Practice Exam Solutions

15-381/781: Artificial Intelligence (Fall 2016)

Name:
Andrew ID:

	17	
	17	
	5	
	5	
	5	
	17	
	17	
	17	
Total		

1 Convex Optimization (17 points)

Consider a standard linear program of the form: minimize $\mathbf{c}^T\mathbf{x}$ such that $A\mathbf{x} \leq \mathbf{b}$. Here $\mathbf{x} \in \mathbb{R}^n$ is the vector of variables, and $\mathbf{c} \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, and $\mathbf{b} \in \mathbb{R}^m$ are constants. Prove from the definitions that this is a convex program.

Solution: First, we show that the objective function is linear, which we denote by f. Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, and let $0 \leq \theta \leq 1$. We need to show $f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \leq \theta f(\mathbf{x}) + (1-\theta)f(\mathbf{y})$. We have

$$\begin{aligned} f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) &= \mathbf{c}^T(\theta\mathbf{x} + (1-\theta)\mathbf{y}) = \mathbf{c}^T(\theta\mathbf{x}) + \mathbf{c}^t((1-\theta)\mathbf{y}) \\ &= \theta\mathbf{c}^T\mathbf{x} + (1-\theta)\mathbf{c}^T\mathbf{y} = \theta f(\mathbf{x}) + (1-\theta)f(\mathbf{y}) \ . \end{aligned}$$

Thus, we conclude that the desired inequality holds (in fact, it holds with equality). Next, we show that the feasible region $\mathcal{F} = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$ is convex. For this, let $\mathbf{x}, \mathbf{y} \in \mathcal{F}$ and let $0 \leq \theta \leq 1$. We need to show that $\theta\mathbf{x} + (1-\theta)\mathbf{y} \in \mathcal{F}$ as well, which amounts to showing $A(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \leq \mathbf{b}$. We have

$$\begin{aligned} A(\theta\mathbf{x} + (1-\theta)\mathbf{y}) &= A(\theta\mathbf{x}) + A((1-\theta)\mathbf{y}) = \theta A\mathbf{x} + (1-\theta)A\mathbf{y} \\ &\leq \theta\mathbf{b} + (1-\theta)\mathbf{b} = (\theta + 1 - \theta)\mathbf{b} = \mathbf{b} \ . \end{aligned}$$

This completes the proof that $\mathcal{F}$ is convex, and hence the proof that a linear program is a convex program.

2 Extensive-Form Games (17 points)

Prove that in Chess, exactly one of the following statements is true:

1. White has a winning strategy, i.e., a strategy such that no matter what Black does, White wins.
2. Black has a winning strategy.
3. Both players can force a tie.

You may use the fact that in an extensive-form game of perfect information, bakward induction gives a subgame perfect Nash equilibrium.

Solution: Consider the game tree for Chess, represented as a zero-sum game. The payoffs are 1 (White wins), -1 (Black wins), or 0 (tie).

Let us apply backward induction to this game tree. We claim that the value of the root must be 1, -1, or 0. This true since the value of each node is one of the values of its children, and the values at the leaves are 1, -1, or 0.

Now suppose that the value of the root is 1. We know that this value is obtained in Nash equilibrium. That is, the strategy of White is such that if Black plays a different strategy, the value is at least 1. This means that White's strategy is a winning strategy. The case of value -1 is symmetric.

When the value is 0, the argument is similar. The two strategies are in equilibrium. This means that for any strategy of Black, the value is at least 0, that is, White can force a tie; and for any strategy of White, the value is at most 0, that is, Black can force a tie.

3 Stackelberg Games (5 points)

Suppose two players are playing a *Stackelberg* game with the payoffs given below, with player 1 (the row player) as the leader. What is the payoff of player 1 under the optimal mixed strategy to commit to?

	L	R
U	$(0,1)$	$(4,0)$
D	$(-1,0)$	$(2,1)$

Solution: Let player 1 play U with probability p and play D with probability $1-p$. Then the expected payoff of player 2 for playing L is p and for playing R is $(1-p)$. So, player 2 picks R iff $p \leq \frac{1}{2}$.

When player 2 plays L, the expected payoff of player 1 is $-(1-p)$ and when player 2 plays R the expected payoff of player 1 is $4p+2(1-p)$. The optimal strategy for player 1 involves player 2 choosing R, so $p = 0.5$.

Player 1 gets expected payoff of $4p + 2(1-p) = 3$ in this strategy.

4 VC-Dimension (5 points)

Prove or disprove: If $\mathcal{C}$ and $\mathcal{C}'$ are two concept classes, then VC-dim($\mathcal{C} \cup \mathcal{C}'$) = VC-dim($\mathcal{C}$)+VC-dim($\mathcal{C}$).

Solution: False. Let $\mathcal{C} = \mathcal{C}'$, then VC-dim($\mathcal{C} \cup \mathcal{C}'$) = VC-dim($\mathcal{C}$) $\neq$ VC-dim($\mathcal{C}$) + VC-dim($\mathcal{C}'$). This statement does not hold even for disjoint concept classes.

5 Uninformed Search (5 points)

Consider the problem of uninformed search on a *finite* search space with a binary tree structure as shown in the following figure (the actual space could be larger).

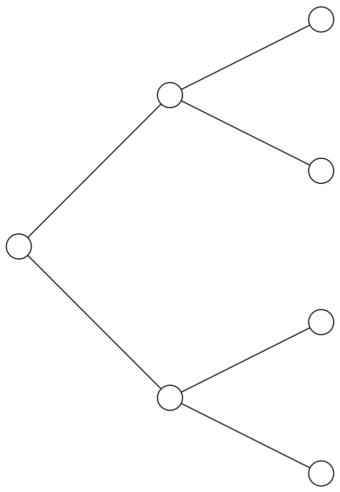

Figure 1: The structure of the search space

Suppose there are multiple goal states, we want the shortest path to the closest goal, and we want to minimize time. We have unlimited memory. Discuss which of BFS, DFS, or IDS would be best suited and why. Briefly explain why the other options are not as well suited as your choice.

Solution: We should use BFS. IDS would also find the closest goal, but uses a constant factor more time than BFS. DFS may not find the closest goal.

6 Informed Search (17 points)

The 8-Puzzle game is a single-player board game which consists of a 3×3 board with 8 tiles and 1 blank slot. A tile can move horizontally or vertically to its adjacent blank slot (if it neighbors it). The objective of the game is to start from a given arrangement of tiles and move tiles to achieve a given goal arrangement.

Here, we discuss some heuristics to be used with A* graph search. Recall, from slides 18 and 19 of Lecture 2, that heuristic $h_1(\cdot)$ returns the number of tiles that are in the wrong position and $h_2(\cdot)$ returns the sum of Manhattan distances of tiles from their goal positions. We introduce a third heuristic, $h_3(\cdot)$, that is the *minimum* number of moves necessary to get to the goal state if each action could move any tile to the blank slot. This is another *relaxed problem* heuristic.

1. (9 points) Prove that h_3 is consistent.

   Solution:

   Let x and y represent any two states of the problem. Define $H(x, y)$ to be the minimum number of moves necessary to go from x to y where each action could move any tile to the blanks slot. Note that this is the same as value of $h_3(x)$ when y is the goal state.

   Sliding a tile is the same as switching a tile with its adjacent blank space (if it neighbors the blank space). Since H allows any tile to be switched with the blank slot regardless of where it is located, H is a relaxation of the 8-puzzle game. So, $H(x, y) \leq \text{cost}(x, y)$.

   Now consider any x and y. One way to go from x to t using the moves defined by H (and h_3) is to first use $H(x, y)$ moves to go from x to y, and then use $h_3(y)$ moves to go from y to t. Using the inequality from the previous paragraph, we have

   $$h_3(x) \leq h_3(y) + H(x, y) \leq h_3(y) + \text{cost}(x, y)$$

   So, h_3 is consistent.

2. (4 point) Prove or disprove: h_3 dominates h_1.

   Solution:

   Yes. h_3 can only place a tile on a blank slot without changing the order of the other tiles. So, for any misplaced tile, h_3 takes at least one move. In fact, if the blank slot is in its correct position, then h_3 makes two moves to take a tile to its correct location. Since, h_1 is the number of misplaced tiles, h_3 is at least as large as h_1, so it dominates h_1.

3. (4 point) Prove or disprove: h_3 dominates h_2.

   Solution: No. Consider a goal state where the blank space is in the bottom right corner and the tiles increase from left to right and top to bottom. Now consider a start state where the blank space and tile 1 are switched. h_3 can move tile 1 to its location in 1 move, whereas h_2 computes the Manhattan distance between the current location of tile 1 and its desired location, which is 4. So h_3 does not dominate $h1$.

7 Deep Learning and Computer Vision (17 points)

1. (8 points) Consider a layer in a convolutional neural network that takes in one 100×100 feature map (e.g., a gray-scale image), and outputs 100 feature maps. In each of the following cases, give the number of parameters that must be learned for this layer. Remember that we include a bias value for each output map.

   (a) The layer is a convolution layer where the filters are the same size as the input feature map.

   Solution: Each filter is 100×100, and there are 100 filters (since there are 100 output maps), and there are 100 biases. Therefore there are $(100^3 + 100)$ parameters

   (b) The layer is a convolution layer with 10×10 filters and a stride of 5.

   Solution: Each filter is 10×10, and there are 100 filters (since there are 100 output maps), and there are 100 biases. Therefore there are $(10 * 10 * 100 + 100)$ parameters.

   (c) The layer is a locally-connected layer with 10×10 tiles and a stride of 5.

   Solution: Since the stride is 5, that means there will be 19x19 = 361 tiles. Each tile is 10×10, so that means $(100 * 361 + 1)$ for each output map. Thus in total there are $(100 * 100 * 361 + 100)$ parameters

   (d) The layer is a convolution layer with 1×1 filters and a stride of 1.

   Solution: There will be 100 filters since there are 100 output maps. Each filter has 1 parameter. Thus there are $100 + 100 = 200$ parameters.

2. (5 points) What is the trade-off with having more or fewer parameters?

   Solution: More parameters leads to a more complex model which is more difficult to train. Fewer parameters leads to a model that is easier to train, and requires less data, but less complex. Too many parameters can also cause overfitting, especially when you have less data, but with too few, the model won't be expressive enough. Therefore, you want to balance the complexity of model, which includes making sure you don't overfit, with how easily you want to train the model (and how much data you need to train).

3. (4 points) Explain an additional reason (besides the number of parameters) convolution layers with filters that are smaller than the input map but larger than 1×1 are well-suited for image-related tasks.

   Solution: Using receptive fields that are smaller than the input provides translation invariance as the same filter is applied to each receptive field. Making the receptive fields larger than 1×1 allows each neuron in the output to be more robust to patterns that shift within the local area. A good receptive field size helps the network exploit the 2D structure of the image and provide robustness to certain differences (e.g., translation, slight rotation) that are trivial in the domain of images.

8 (From Homework) Choosing the Value of $R_{\max}$ (17 points)

This question refers to the R-MAX algorithm (to be covered in lecture the week of 10/10).

The R-MAX algorithm makes the assumption that we know the maximum possible reward. In the real world, this might not always be the case. In this problem we explore some possible modifications to the algorithm when we don't know the maximum reward.

8.1 Setting an upper bound [3 points]

Suppose we have a known upper bound for the reward and we set $R_{\max}$ to be this upper bound. When this bound is very loose, i.e., the bound is much greater than the true maximum possible reward, how will the algorithm behave?

Solution: If the bound is very loose, the algorithm will visit every state-action pair the threshold number of times since any state-action pair that remains unknown will appear to give a much higher reward than those that are known.

8.2 Working up to the true bound [8 points]

Instead, suppose we initialize $R_{\max} = 0$ (assume this is the minimum possible reward), and every time a state-action pair becomes known with a reward $\rho > R_{\max}$, we set $R_{\max} = \rho$ (and modify our unknown states accordingly). While intuitively this may seem like it should work, in some cases it does not. Give a simple MDP where this fails, i.e., where we will never find the optimal policy. Show that we never find the optimal policy on your example under balanced wandering.

Solution: Imagine a very long 1D grid of states. You start at the very left. The actions are T to teleport back to the leftmost state or R to move right and are both deterministic. Action T has a reward of 1. Action R moves right with reward 0, except for the right most state, where it is a self-loop with reward M, a very very large number. Suppose γ is 0.9.

Consider balanced wandering where it breaks ties when counts are the same by taking the lowest index action. Here we let T be the lowest index action. Let's label the states starting from the left with $s_0, s_1, \ldots, s_H$, where s_H is the rightmost state. Note that the only time R-MAX can advance to the right is if R-MAX picks action R. However for every state, balanced wandering will alternate between T and R, meaning R-MAX will be teleported back to the leftmost state very often. This means that an exponential number of steps is required to reach s_i, i.e. $\Omega(2^i)$, before any counts reach the threshold. This means that the counts for s_0 will reach the threshold before R-MAX will ever reach close to s_H. When the counts for s_0 reach the threshold, R-MAX will learn that action T has reward 1 and self-loops, whereas R has reward 0. Then R-MAX will set $Rmax = 1$. However since R has reward 0, the value for R will be $0 + \gamma V(s_1)$ which is not as good as just always taking T for s_1. Thus R-MAX will now always take T and keep getting reward 1 for all eternity.

For balanced wandering with random tie-breaking, the same example will work, except now we note that the probability that R-MAX reaches s_i is exponentially low i.e. $O(2^{-i})$. Thus R-MAX will most likely never reach s_H before the counts for s_0 reach the threshold.

8.3 Optimistically increasing $R_{\max}$ (6 points)

Now suppose we use the same algorithm as in (1.1.2), but instead we set $R_{\max} = \rho + \delta$, for some δ, which may depend on the discount factor, γ. Either give a brief description of why this won't fail as it did in (1.2.2), or show that for any value of δ, you can construct an MDP where this fails.

Solution: We use the same example as for the previous question. The only difference is that once s_0 reaches the threshold, due to δ, R-MAX may not believe T is better than R, so R-MAX may continue to try R. Then we continue R-MAX until the counts for s_1 reaches the threshold, until $s_2, s_3, \ldots, s_k$ reaches the threshold. At this point, for large enough k, the value of taking R at state s_0 will drop, because it has discovered that no reward can be gotten from all the other states $s_1, \ldots, s_k$. Thus any policy that tries to keep taking R will have value $0 + 0 + 0 + \cdots + \gamma^k V(s_k)$. For large enough k, taking action T will be better than action R at s_0. Chances are that R-MAX will teleport back to s_0 at some point and from then on be stuck taking action T for all eternity. For large enough H, s_H will still almost never be reached, before the intermediates states reach the threshold.