CS221 Midterm Review Problem Set

Week 5

Note that this is only a subset of topics covered so far in the course. Make sure you still study the lecture material, previous problem sessions (week 4 is great for MDPs which are not covered here), and use the released back exams to your advantage.

1) *"The fear of loss [functions] is a path to the dark side."* - Yoda

(a) We have a trained linear regression model $f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$. In your own words, explain why we call this model linear. Is it linear in x? Linear in $\phi(x)$? Linear in $\mathbf{w}$? Note that linearity means that $g(x+y) = g(x) + g(y)$ and $g(\alpha x) = \alpha g(x)$ for all α.

(b) We are working with a classification model $f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$. What is the decision boundary? What does $\mathbf{w} \cdot \phi(x)y = -1000$ mean? How about $\mathbf{w} \cdot \phi(x)y = 0.1$? You consider using the loss function

$$\mathbb{1}[(\mathbf{w} \cdot \phi(x))y \leq 0]$$

for gradient descent. Explain why this is a bad idea.

(c) After solving the prior problem you realize zero-one loss is bad and decide to use logistic loss. Your data is $y \in \{0, +1\}$ so you define the logistic loss:

$$L(x, y; \mathbf{w}) = -y \log(f(x; \mathbf{w})) - (1 - y) \log(1 - f(x; \mathbf{w})) \tag{1}$$

where f has a range of $[0, 1]$. Before picking f, you'd like to differentiate L with respect to $\mathbf{w}$. Is this possible, and if so, what is $\frac{\partial L}{\partial \mathbf{w}}$?

(d) You can't decide between using the sigmoid function,

$$g(x; \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}^T x}}$$

and a shifted tanh function,

$$h(x; \mathbf{w}) = \frac{1}{2} \tanh(\mathbf{w}^T x) + \frac{1}{2} \quad \text{with} \quad \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

in your loss in place of f. Compute the loss from Equation 1 using g for f and then h for f.

(e) Assume you were able to format $\frac{\partial L(x,y;\mathbf{w})}{\partial \mathbf{w}}$ as $cx(f(x;\mathbf{w})-y)$ in the previous problem, where c is a constant and f is the corresponding sigmoid g or tanh h. (if you couldn't get it like this, don't worry, it's just some algebra). Explain why this loss function's gradient is very convenient for backpropagation. Hint: think about what quantities we compute when evaluating $L(x, y; \mathbf{w})$.

(f) Unfortunately your model has poor performance for both sigmoid and tanh. You think that it's because your model isn't expressive enough. You decide to make your model a neural network to hopefully fix that. You decide to keep the loss function and still use either sigmoid or tanh as your final activation (mostly since you've already differentiated them). Thus rather than plug x directly into your choice for f, you plug x into a neural network $N(x; A, B) = z$ and then take $f(z; \mathbf{w})$. Let

$$N(x; A, B) = B \max\{Ax, 0\} = z$$

And the loss is now:

$$L(x, y; A, B, \mathbf{w}) = -y \log(f(N(x; A, B); \mathbf{w})) - (1 - y) \log(1 - f(N(x; A, B); \mathbf{w}))$$

You figure you need to do some more differentiating, but you think you can reuse some things from earlier problems. Can we we reuse our result from (d) for $\frac{\partial L}{\partial w}$?

2) "I am the Lorax who speaks for the [game] trees, which you seem to be [alpha-beta pruning] as fast as you please!" - The Lorax

(a) Evaluate the following game (Figure 1) where the edges are probabilities:

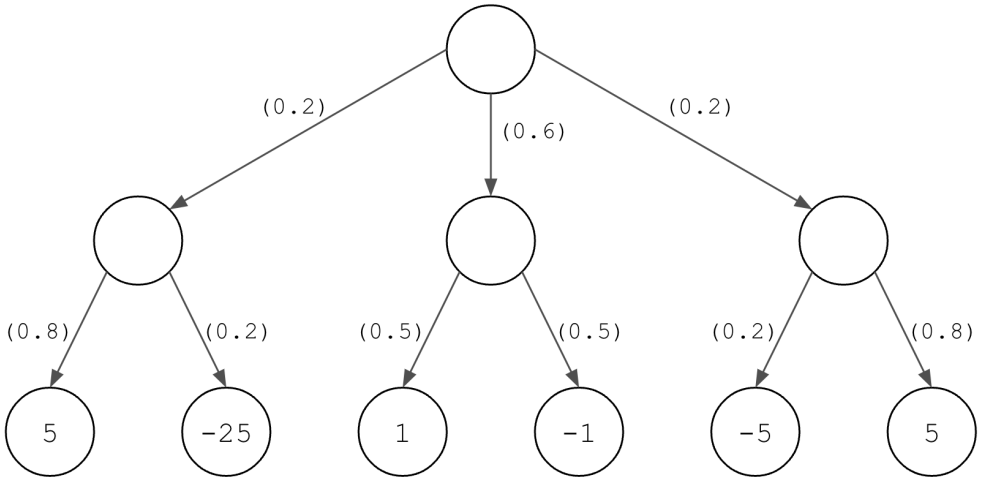

Figure 1

Pretend the top node is now a maximizing player. Under expectimax which action should they take (left, center, or right) and what is the value of the game.

(b) Evaluate the game in Figure 2 using the minimax strategies for both players, with $x = -5$. Recall that upwards pointing triangles is the maximizing player and downwards pointing is the minimizing player.

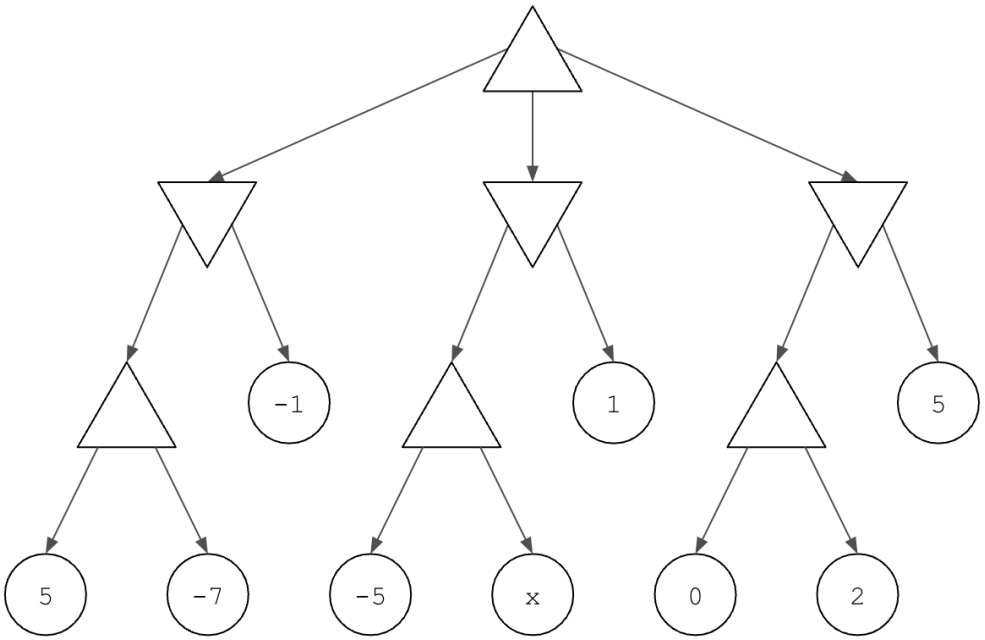

Figure 2

Can we pick x so that the maximizing player loses? Why or why not.

(c) Can either player do better by deviating from minimax assuming the other stays?

(d) Evaluate the game in Figure 3 under the expectiminimax strategy, using $x = -5$. Write down a funny answer for who the third player playing the circles is.

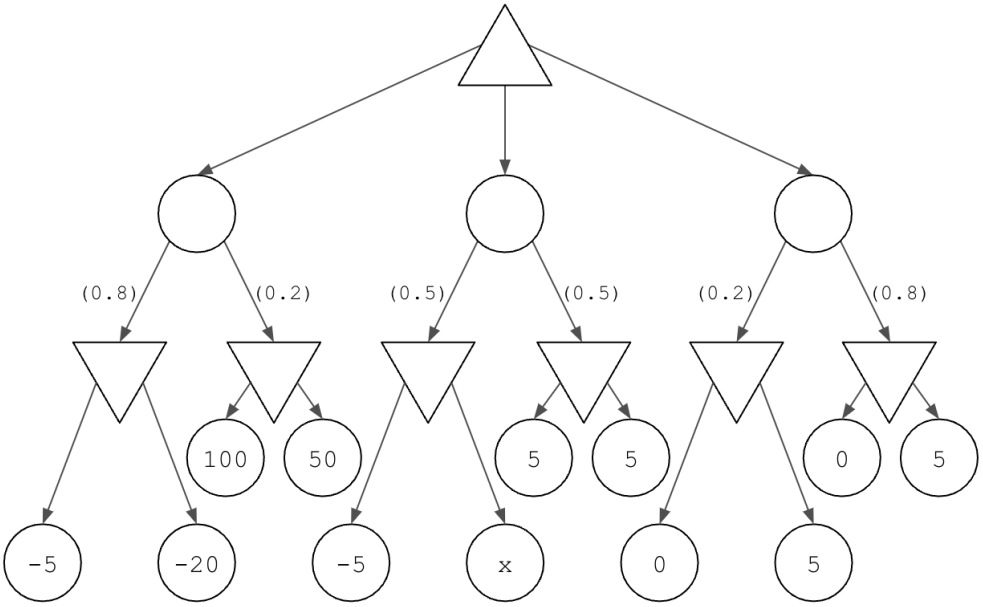

Figure 3

(e) In the previous problem, is there a value of x we can choose so that the game does not end in a draw?

(f) Assume that in the case of a tie in the value of multiple options, the maximizing player chooses the rightmost tied-value action. Still referring to (d) and Figure 3 with $x = -5$, explain, in your own words, why expectiminimax always chooses to draw the game given this choice of tie-breaking. Is there a better way of breaking ties?

(g) Let's develop some intuition for alpha-beta pruning. Look at the minimax tree in Figure 4, specifically the left subtree.

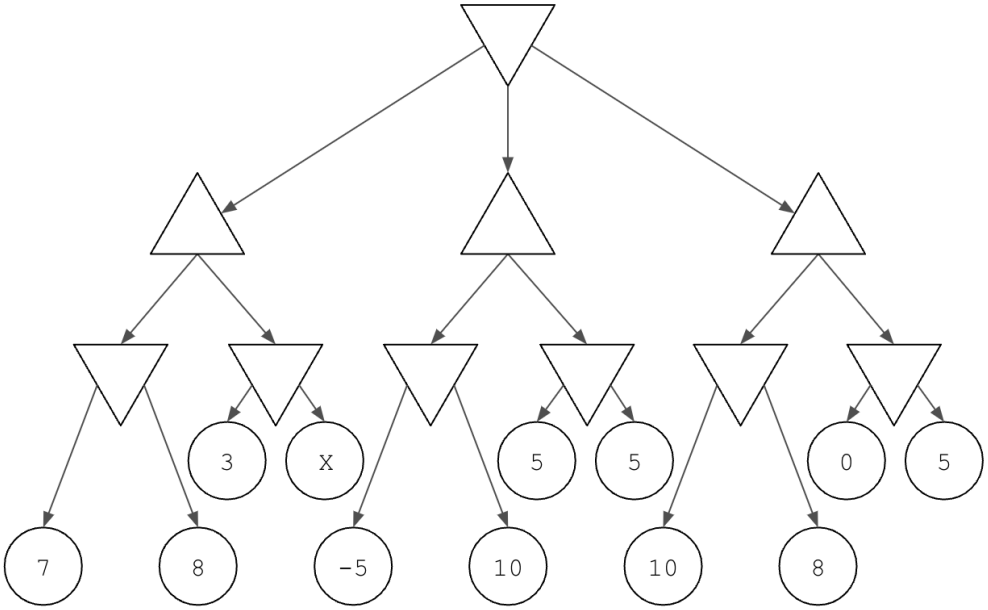

Figure 4

Explain why we don't care about the value of x.

(h) To run alpha-beta pruning we:

- Find a_s, the lower bound on value at max node s.
- Find b_s, the upper bound on value at min node s.
- To prune, calculate the following (where $s' \leq s$ indicates ancestors):

$$\alpha_s = \max_{s' \leq s} a_{s'} \quad \text{and} \quad \beta_s = \min_{s' \leq s} b_{s'}$$

noting that we are maximizing over lower bounds and minimizing over upper bounds.

Run alpha-beta pruning on the tree from the previous question, Figure 4.

(i) You recall learning about Nash Equilibrium and the Prisoner's Dilemma in lecture, but you can't remember exactly what the numbers were. You remember there were three conditions:

- If both testify, then both are sentenced to a years in jail.
- If both refuse, then both are sentenced to b years in jail.
- If only one testifies, they get 0 years and the other gets c years.

And you remember that a Nash Equilibrium is when no player has incentive to change their strategy. First, write down the payoff matrix for this game in terms of a, b, c, and 0. The matrix should look symmetric. Then, write down the conditions (or just values for a, b, and c) so that both testify is a Nash Equilibrium.

3) Ikea is a BFS (Big Furniture Store)

Oh no! You're trapped in an Ikea! Let's use search to get out. You're located at $(0, 0)$ and the exit is at $(1, 1)$. Between you and the exit are a significantly large number of boxes (denoted $b_i \in B$), situated somewhere in the square defined by corners $(0, 0) - (1, 0) - (1, 1) - (0, 1)$, blocking your path. You can remove boxes that are in your way, but you'd like to minimize your effort.

For this problem, consider the algorithms: Backtracking, BFS, DFS, DFS-ID, Dynamic Programming, UCS, and A^*.

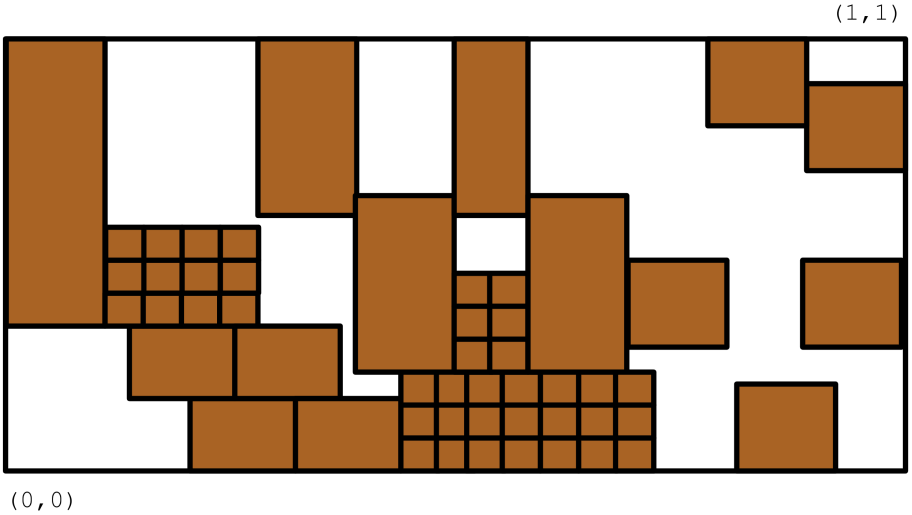

Figure 5: Example scenario, with start at bottom left and end at top right.

(a) Let's formalize this as a search problem:

- $s_{\text{start}} =$
- IsEnd$(s) =$
- States $S =$
- Actions$(s) =$
- Succ$(s, a) =$
- Cost$(s, a) =$ unknown for now

(b) Consider that the boxes are all empty and you don't care how many you have to remove. Write a cost function for this scenario. What algorithm would you use to find a path to the exit if you only care about *how long* your algorithm takes to run? What is the exact worst case length of your path (number of boxes traversed)? Is it possible for DFS to return the shortest path?

(c) After throwing hundreds of tiny little boxes out of your way, you realize you do in fact care how many boxes you have to remove. Write a cost function for this scenario. What algorithm would you use to find the shortest path to the exit? If there are $O(b)$ boxes reachable from each box, and the shortest path is length d, what is the complexity of this algorithm? What is the memory cost?

(d) Inside one of the boxes you've found a nifty little drone that can fly up, look at the boxes, and perform a sort of DFS up to a given depth. This means that given a depth $\tilde{d}$, the drone reports whether there is a path of length at most $\tilde{d}$ or not (there is no information about a shorter path if one is found). Let's say you know there exists a path of length d_0 from part (b). You don't know how long the battery in your drone is going to last, so to conserve it you'd like to have it run the algorithm for as few $\tilde{d}$ as possible. What should you do (can be a rough description of your method) and how many runs will you need at most?

(e) You've come to realize that not all boxes are equally easy to move, and you'd prefer to move small boxes over large boxes. For simplicity, let's assume that all boxes are equally dense, so the effort to move them is **nonzero** and proportional to their area. What is our new cost function? Can we use dynamic programming to minimize the the effort along our path to the exit? What about UCS? If $|B| \to \infty$ will UCS work with some restrictions on the problem? If so what are the restrictions?

(f) UCS seems to be taking too long so you decide to give A^* a try. Your friend suggests you use distance (either Euclidean/ℓ_2 or Manhattan/ℓ_1) to the goal as a heuristic. You can use the center of each box as the 'location' of the box. Prove or disprove that your friend's heuristic is consistent.

(g) Since distance didn't work you need a new heuristic. Propose one and prove that it works.