CS221 Midterm Review Solutions

Week 5

Midterm Prep

This review only covers the **general machine learning** and **games** topics, which are of course only a subset of the topics covered so far in the course. The problems in this handout aim to (i) gather all the general machine learning topics that we covered at the start of the quarter into one problem and (ii) go over the most recent topic, games, which was just introduced this week.

If time permits during the problem session, then we may also go over Problem 2 from Week 3's problem session on **search** (see the handout for that week's Friday session on the website).

Besides **general machine learning**, **search**, and **games**, make sure you also refresh yourself on the **k-means clustering algorithm** as well as **Markov decision processes (MDPs)**. For MDPs, we recommend looking back at the second half of Week 4's problem session, during which we went over some comprehensive problems regarding MDPs and when each MDP algorithm is useful (the handout and slides for that week's Friday session are also on the website).

For more practice problems, see the Files on this class's Canvas, which should include three exams with solutions from previous years.

Practice Problems

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 for some generic function g means that $g(x + y) = g(x) + g(y)$ and $g(\alpha x) = \alpha g(x)$ for all parameters α.

Solution Linear regression models are linear both in $\phi(x)$ and $\mathbf{w}$, but not in x. A simple example of this is using $\phi(x) = [1, x, x^2, x^3]$, which can be used to fit non-linear cubic features of x.

(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 $\mathbf{w} \cdot \phi(x)y = -1000$ imply about how well our model classified the point (x, y)? What does $\mathbf{w} \cdot \phi(x)y = 0.1$ imply about how well our model classified the point (x, y)?

Additionally, you consider using the following loss function

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

for gradient descent. Explain why using this loss function is a bad idea.

Solution The decision boundary is $w \cdot \phi(x) = 0$.
In the first case for $\mathbf{w} \cdot \phi(x)y = -1000$, our model is confident in the classification (notice how $\mathbf{w} \cdot \phi(x)$ is large and thus, the point is likely classified far from the decision boundary), but incorrect (because the sign is wrong, as the product $\mathbf{w} \cdot \phi(x)y$ can only be negative if the prediction and y have opposite signs).

In the second case for $\mathbf{w} \cdot \phi(x)y = 0.1$, our model is not very confident (as $\mathbf{w} \cdot \phi(x)y = 0.1$ is small, so the point is classified near the decision boundary), but at least the sign is correct, meaning the point is classified correctly.

The loss function is the zero-one loss function, which has zero gradient almost everywhere! Such a function won't do well for gradient descent, which requires meaningful derivatives.

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

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

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 $\frac{\partial L}{\partial \mathbf{w}}$? (Food for thought: how would the derivative change if it were over a summation?)

Solution Yes! We use the chain rule:

$$\begin{aligned}\frac{\partial L(x, y; \mathbf{w})}{\partial \mathbf{w}} &= -y \frac{1}{f(x; \mathbf{w})} \frac{\partial f(x; \mathbf{w})}{\partial \mathbf{w}} + (1 - y) \frac{1}{1 - f(x; \mathbf{w})} \frac{\partial f(x; \mathbf{w})}{\partial \mathbf{w}} \\ &= \left(\frac{f(x; \mathbf{w}) - y}{f(x; \mathbf{w})(1 - f(x; \mathbf{w}))} \right) \frac{\partial f(x; \mathbf{w})}{\partial \mathbf{w}}\end{aligned}$$

A summation in the loss function would simply lead to a summation in the derivative, with the same expression as above inside the summation (but with indices i on (x_i, y_i)).

(d) For your function f in the above loss function, you can't decide between using the sigmoid function,

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

or the 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 place of f. How would the derivative from Part (c) look like with function g above in place of f, and with function h above in place of f?

Solution To make things easier for ourselves, we can first compute $\frac{\partial g}{\partial \mathbf{w}}$ and $\frac{\partial h}{\partial \mathbf{w}}$ and substitute those into $\frac{\partial f}{\partial \mathbf{w}}$ in our solution from (c). Thus

$$\begin{aligned}
\frac{\partial g(x;\mathbf{w})}{\partial \mathbf{w}} &= -(1+e^{-\mathbf{w}^T x})^{-2}\frac{\partial}{\partial \mathbf{w}}\left(1+e^{-\mathbf{w}^T x}\right) \\
&= \frac{x e^{-\mathbf{w}^T x}}{(1+e^{-\mathbf{w}^T x})^2} \quad \text{(this is a valid answer)} \\
&= x\frac{1}{(1+e^{-\mathbf{w}^T x})}\frac{e^{-\mathbf{w}^T x}}{(1+e^{-\mathbf{w}^T x})} \\
&= x g(x;\mathbf{w})(1-g(x;\mathbf{w}))
\end{aligned}$$

and

$$\begin{aligned}
\frac{\partial h(x;\mathbf{w})}{\partial \mathbf{w}} &= \frac{1}{2}\frac{\partial \tanh(\mathbf{w}^T x)}{\partial \mathbf{w}} \\
&= \frac{1}{2}\frac{\partial}{\partial w}\frac{(e^{\mathbf{w}^T x} - e^{-\mathbf{w}^T x})}{(e^{\mathbf{w}^T x} + e^{-\mathbf{w}^T x})} \\
&= \frac{1}{2}\left[\frac{(e^{\mathbf{w}^T x} + e^{-\mathbf{w}^T x})}{(e^{\mathbf{w}^T x} + e^{-\mathbf{w}^T x})} - \frac{(e^{\mathbf{w}^T x} - e^{-\mathbf{w}^T x})^2}{(e^{\mathbf{w}^T x} + e^{-\mathbf{w}^T x})^2}\right]x \\
&= \frac{1}{2}(1-\tanh(\mathbf{w}^T x)^2)x
\end{aligned}$$

and we can plug these into our solution from (c). For sigmoid g:

$$\begin{aligned}
\frac{\partial L(x,y;\mathbf{w})}{\partial \mathbf{w}} &= \left(\frac{g(x;\mathbf{w}) - y}{g(x;\mathbf{w})(1-g(x;\mathbf{w}))}\right)\frac{\partial g(x;\mathbf{w})}{\partial \mathbf{w}} \\
&= \left(\frac{g(x;\mathbf{w}) - y}{g(x;\mathbf{w})(1-g(x;\mathbf{w}))}\right) g(x;\mathbf{w})(1-g(x;\mathbf{w}))x \\
&= x(g(x;\mathbf{w}) - y)
\end{aligned}$$

which looks surprisingly similar to the gradient of squared error loss. For tanh h:

$$\begin{aligned}
\frac{\partial L(x,y;\mathbf{w})}{\partial \mathbf{w}} &= \left(\frac{h(x;\mathbf{w}) - y}{h(x;\mathbf{w})(1-h(x;\mathbf{w}))}\right)\frac{\partial h(x;\mathbf{w})}{\partial \mathbf{w}} \\
&= \left(\frac{h(x;\mathbf{w}) - y}{\frac{1}{2}(\tanh(\mathbf{w}^T x)+1)(1-\frac{1}{2}(\tanh(\mathbf{w}^T x)+1))}\right)\frac{1}{2}(1-\tanh(\mathbf{w}^T x)^2)x \\
&= \left(\frac{h(x;\mathbf{w}) - y}{(\tanh(\mathbf{w}^T x)+1)\frac{1}{2}(1-\tanh(\mathbf{w}^T x))}\right)(1-\tanh(\mathbf{w}^T x)^2)x \\
&= 2x(h(x;\mathbf{w}) - y)
\end{aligned}$$

Isn't that neat?!

(e) Assume that 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 functions. Explain why writing the derivative of the loss function in the form of $cx(f(x;\mathbf{w})-y)$ is very convenient for backpropagation.

Hint: Draw out the backpropagation tree and think about what quantities we would need to compute when evaluating $L(x,y;\mathbf{w})$.

Solution This form of the gradient is convenient because it just uses x (our data), y (our label), and $f(x;\mathbf{w})$, which we have to compute anyways when we compute the loss. All of these quantities are computed during the forward pass, so there is no extra computation needed during backpropagation other than putting the pieces together.

(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 because you've already done the work in differentiating them).

Now, rather than pluging 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

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

(Self check: Do you remember the derivative of the max function?)

The loss function is now:

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

It looks you need to find the derivative again for this new loss function, but you think you can reuse your work from the earlier parts of this problem. Can we we reuse our result from (d) for $\frac{\partial L}{\partial w}$?

(Food for thought: suppose we figure that our model's poor performance was due to overfitting instead. Why might L_2 regularization help, and how would it change our loss function from Part (c)?)

Solution Yes, we can reuse our result for $\frac{\partial L}{\partial \mathbf{w}}$ with just a slight change. We need to replace x with $z=N(x;A,B)$ whenever it appears. But, remember we were differentiating with respect to $\mathbf{w}$, so we treated x as a constant. By replacing x with $z=N(x;A,B)$, we are just redefining a *constant*, which won't change the actual differentiation process.

L_2 regularization penalizes our weights $\mathbf{w}$ from having too much influence in our loss function to avoid overfitting. We'd simply add a $+\frac{\lambda}{2}||\mathbf{w}||_2^2$ term to our loss function.

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:

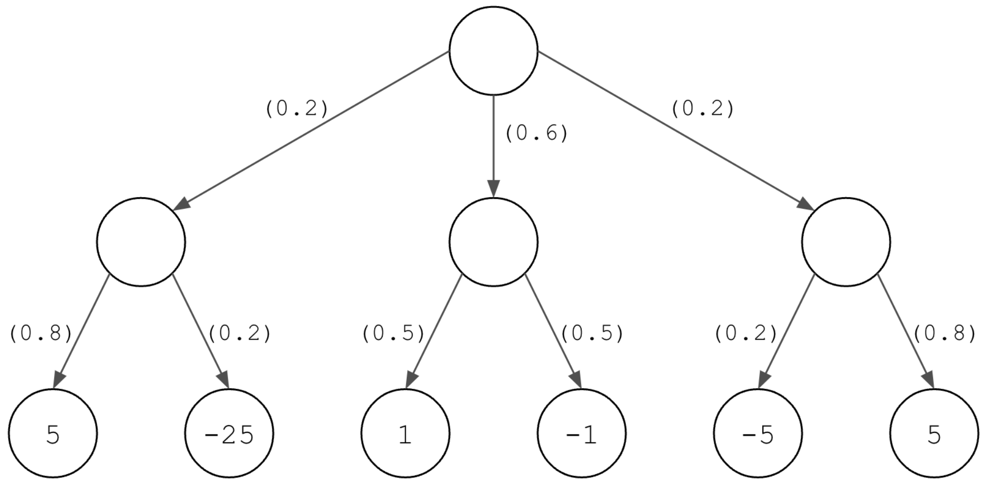


Figure 1

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?

Solution Bottom row left to right: $-1, 0, 3$. Overall value is 0.4. Under expectimax, they would choose the right subtree and get a value of 3.

(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.

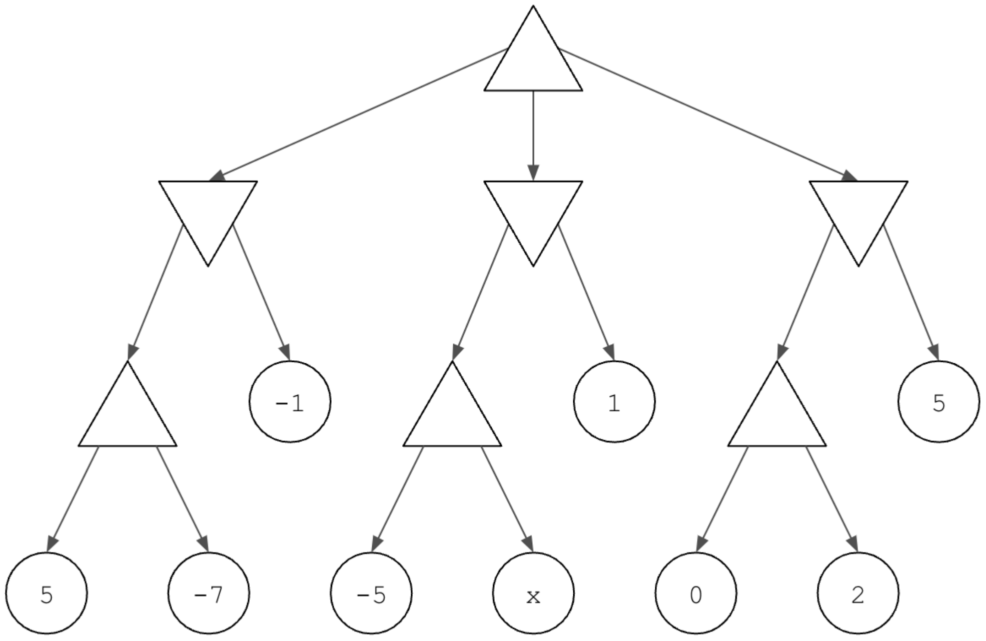


Figure 2

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

Solution Bottom row of maximizing triangles, left to right: 5,-5,2
Second row of minimizing triangles, left to right: -1,-5,2
Top (value of the game): 2.
No, we cannot make the maximizing player lose by changing x. If $x < -5$, then it is ignored by its parent maximizer, and if $-5 \leq x < 1$, then it is chosen by the minimizer, but ignored by the right subtree having a minimax value of 2. If $x \geq 1$, then the 1 in the middle subtree will be chosen by the minimizer.

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

Solution No! Proved in lecture. If we could improve by deviating then it wouldn't be minimax by definition.

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

Solution Bottom row of minimizing triangles, left to right: -20, 50, -5, 5, 0, 0
Expected value of middle row of circles, left to right: -6, 0, 0
Expected value of the game (maximizer at the root): 0

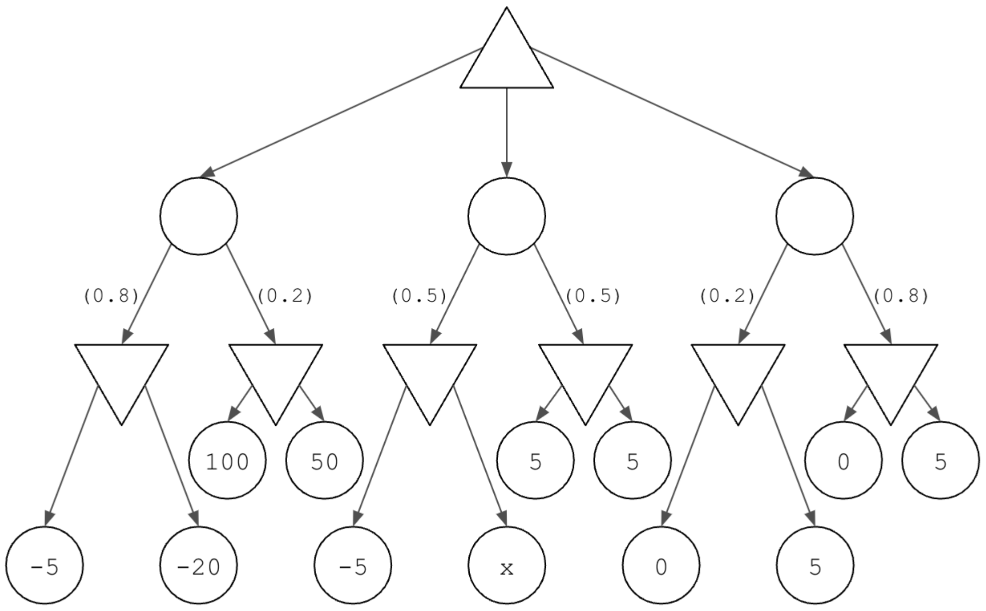


Figure 3

I like to think that the third player playing the randomness in the circles is my lucky penny that I lost in third grade and has been out to get me ever since. (The point is that the circles follow a known stochastic policy. They aren't 'playing' the game the same way as the two players are).

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

Solution No, making $x < -5$ would result in decreasing the expected value of the middle subtree, but the right subtree has an expected value of 0. If we take $x > -5$, then it'll be ignored by its parent minimizer.

(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?

Solution With $x = -5$ both the middle subtree and rightmost subtree have expected value of 0. However, in the right subtree, the expected value has zero variance, and the game will always draw since both minimizing nodes have value 0. The middle subtree is averaging -5 and 5 with equal probability, which is an expected value of 0 as well, but this time with non-zero variance. However, it isn't necessarily better to break the tie towards larger variance. We go from guaranteeing a draw to losing half the time and winning half the time if we take the middle subtree, and one isn't necessarily 'better' than the other given the defined utilities of the game.

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

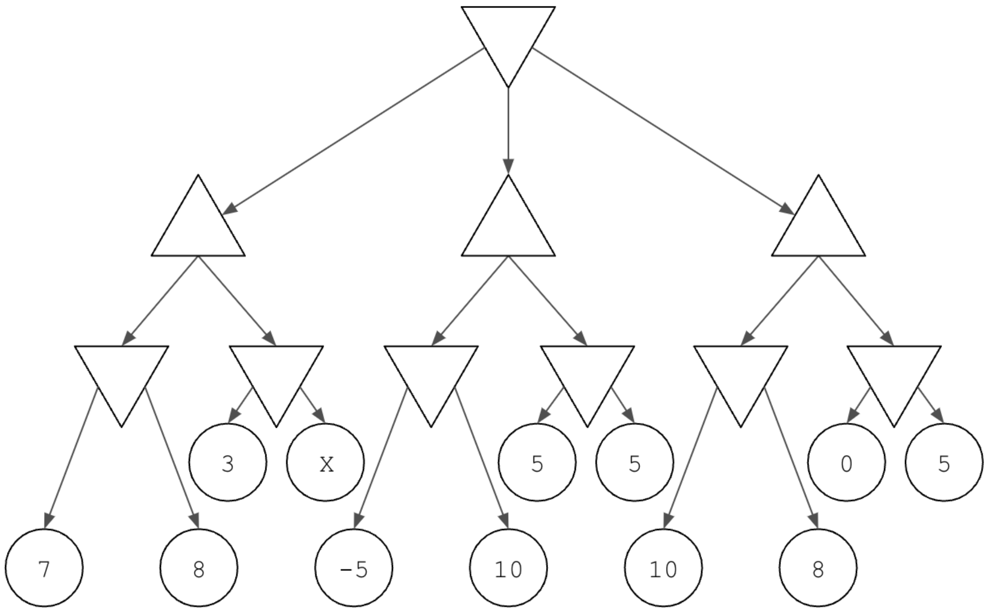


Figure 4

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

Solution The bottom left minimizing node will choose 7. Then, the parent maximizer to that minimizer will choose at least 7. Going to the second leftmost minimizer, the first value is a 3, meaning the minimizer will choose at most 3. Since the maximizer already has access to a 7, it isn't going to care if the second minimizing node offers a 3 or something smaller. Thus we don't care about the value of x since it won't change our decision in the left subtree.

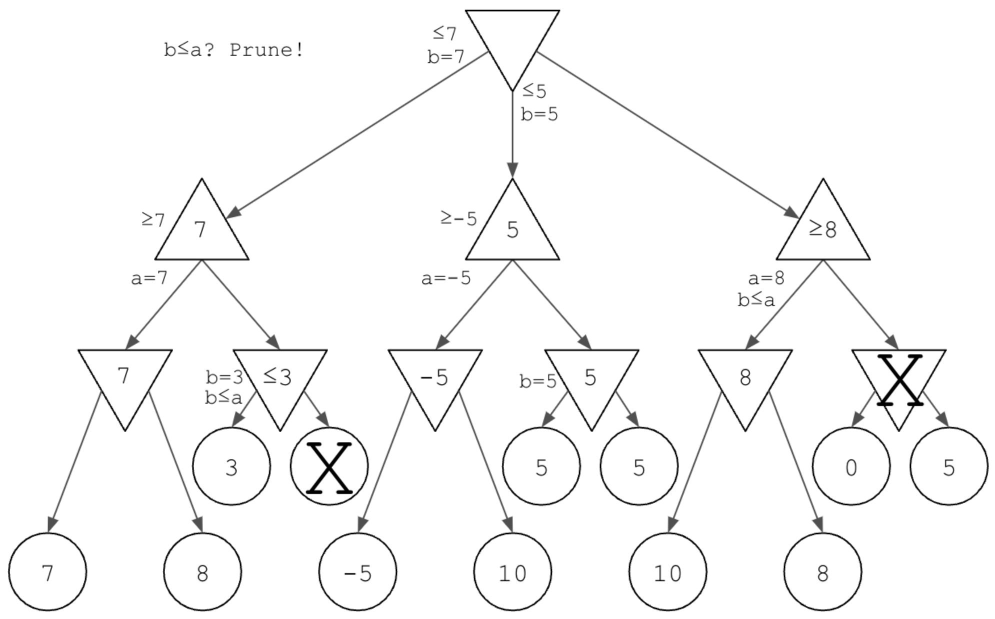
(h) To run alpha-beta pruning we:

- 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.



Solution Solution:

See we can confirm that we cannot prune anything in the middle subtree since we would be pruning the optimal value of 5.