Q7. [8 pts] So Many Derivatives

Consider the neural network configuration below.

(a) [2 pts] Which of the following decision boundaries can be learned by the neural network? Assume \( w_0, w_1, x_0, x_1, T_a \in \mathbb{R}^n \) and \( z = w_0x_0 + w_1x_1 \). Let \( g(z) \) be the binary step activation function with \( T_a \) as the decision threshold, which is defined as follows:

\[
g(z) = \begin{cases} 
1 & \text{if } z \geq T_a \\
0 & \text{if } z < T_a 
\end{cases}
\]

[A] Graph A  
[B] Graph B  
[C] Graph C  
[D] Graph D  
[E] Graph E  
[F] None of the above

(b) Now let \( g(z) \) be the sigmoid activation function and \( y \) be a real number value between 0 and 1 (we will ignore the threshold \( T_a \) for this part). Recall that the derivative of the sigmoid function is \( \frac{d}{dz} g(z) = g(z) \cdot (1 - g(z)) \). You can represent your answers in terms of \( x_0, x_1, w_0, w_1, z, \) or \( y \).

(i) [2 pts] Calculate the following partial derivatives for backpropagation.

(1) \( \frac{dy}{dz} = \)  

(2) \( \frac{dz}{dw_0} = \)
(ii) [2 pts] Suppose we are running gradient descent on the neural network above. We are trying to minimize the upstream loss $L$ using learning rate $\alpha$. Given the upstream gradient $\frac{\partial L}{\partial y}$ and the two partial derivatives that you computed in the previous part ($\frac{\partial y}{\partial z}$ and $\frac{\partial z}{\partial w_0}$), determine the gradient descent update rule for $w_0$.

\[ w_0 \leftarrow \ldots \]

(c) [2 pts] The Binary Perceptron is defined as the following:

\[ y = \text{classify}(x) = \begin{cases} 
+1 & \text{if } w \cdot f(x) + b \geq 0 \\
-1 & \text{if } w \cdot f(x) + b < 0
\end{cases} \]

where $w$ is a vector of real-valued weights, $w \cdot f(x)$ is the dot product $\sum_{i=1}^{m} w_i f_i(x)$ where $m$ is the number of features, $f_i(x)$ is the $i$th feature of $x$, and $b$ is the bias.

Which of the following are true about the binary perceptron as defined above?

[A] It is possible that the perceptron learns a decision boundary that is nonlinear in terms of the features $f(x)$.

[B] It is possible that the perceptron learns a decision boundary that is nonlinear in terms of the data $x$.

[C] The perceptron algorithm is guaranteed to converge if the data is linearly separable.

[D] The perceptron algorithm is trained using gradient descent.

[E] None of the above
Q9. [8 pts] Higher-Dimensional Perceptrons

Consider a dataset with 6 points on a 2D coordinate grid. Each point belongs to one of two classes. Points $[-1, 0], [1, 0], [0, 1]$ belong to the negative class. Points $[-2, 0], [2, 0], [0, 2]$ belong to the positive class.

(a) [1 pt] Suppose we run the perceptron algorithm with the initial weight vector set to $[0, 5]$. What is the updated weight vector after processing the data point $[0, 1]$?

(b) [1 pt] How many iterations of the perceptron algorithm will run before the algorithm converges? Processing one data point counts as one iteration. If the algorithm never converges, write $\infty$.

In the next few subparts, we’ll consider transforming the data points by applying some modification to each of the data points. Then, we pass these modified data points into the perceptron algorithm.

For example, consider the transformation $[x, y] \rightarrow [x, y, x^2, 1]$. In this transformation, we add two extra dimensions: one whose value is always the square of the first coordinate, and one whose value is always the constant 1. For example, the point at $[2, 0]$ is transformed into a point at $[2, 0, 4, 1]$ in 4-dimensional space.

(c) [2 pts] Which of the following data transformations will cause the perceptron algorithm to converge, when run on the transformed data? Select all that apply.

- $[x, y] \rightarrow [y, x]$
- $[x, y] \rightarrow [x, y, 1]$
- $[x, y] \rightarrow [x, y, x^2, 1]$
- $[x, y] \rightarrow [x, y, x^2 + y^2]$
- None of the above.

(d) [2 pts] Suppose we transform $[x, y]$ to $[x, y, x^2 + y^2, 1]$, and pass the transformed data points into the perceptron. Write one possible weight vector that the perceptron algorithm may converge to.

(e) [2 pts] Construct another transformation (not equal to the ones above) that will allow the perceptron algorithm to converge. Hint: The transformation $[x, y] \rightarrow [x, y, x^2 + y^2, 1]$ allows the perceptron algorithm to converge. Fill in the blank: $[x, y] \rightarrow [x, y, \underline{\quad}, 1]$. 

Exam continues on next page.
Q10. [9 pts] Q-Networks

Consider running Q-learning on the following Pacman problem: the maze is an $x$-by-$x$ square, and each position can contain a food pellet or no food pellet. There are no ghosts or walls. Pacman’s only actions are {up, down, left, right}.

(a) [2 pts] How many Q-values do we need to learn for this problem?

Your answer should be an expression, possibly in terms of $x$.

To learn every Q-value, we could run standard Q-learning, but we decide to try a different approach:

Suppose somebody tells us $N$ exact Q-values for $N$ different state-action pairs: the exact Q-value for the state-action pair $(s_i, a_i)$ is $q_i$, for $1 \leq i \leq N$. ($N$ is less than the total number of state-action pairs.)

We decide to use these exact Q-values to train a neural network, so that we can estimate other Q-values we don’t know. To train this neural network, we need to apply gradient descent to minimize the following loss function:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} (f(\theta, s_i, a_i) - q_i)^2$$

$\theta$ represents the weights of the neural network. $f(\theta, s_i, a_i)$ represents running the neural network with weights $\theta$ on state-action pair $(s_i, a_i)$.

(b) [1 pt] What is the gradient $\frac{\partial L}{\partial \theta}$?

Your answer should be an expression, possibly in terms of $N$, $\frac{\partial f}{\partial \theta}$, and $q_i$.

(c) [1 pt] After running $t$ iterations of gradient descent, our current weights are $\theta_t$. The learning rate is $\alpha$.

What are the weights on the next iteration, $\theta_{t+1}$?

Your answer should be an expression, possibly in terms of $\theta_t$, $\alpha$, and $\frac{\partial L}{\partial \theta}$.

$$\theta_{t+1} =$$

(d) [2 pts] Eventually, gradient descent converges to the weights $\theta^*$.

We use the neural network with weights $\theta^*$ to compute Q-values, and extract a policy out of these Q-values:

$$\pi(s) = \arg \max_a f(\theta^*, s, a)$$

Is $\pi$ the optimal policy for this problem?

- [ ] Yes
- [ ] No
- [ ] Not enough information

Instead of the Pacman problem, consider a different problem where the action space is continuous. In other words, there are infinitely many actions available from a given state.

(e) [3 pts] Can we still use the strategy from the previous subparts (without any modifications) to obtain a policy $\pi$?

- [ ] Yes
- [ ] No

Briefly explain why or why not.