Q1. Perceptron

We would like to use a perceptron to train a classifier for datasets with 2 features per point and labels +1 or -1.

Consider the following labeled training data:

<table>
<thead>
<tr>
<th>Features</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1, x_2)</td>
<td>y^*</td>
</tr>
<tr>
<td>(-1,2)</td>
<td>1</td>
</tr>
<tr>
<td>(3,-1)</td>
<td>-1</td>
</tr>
<tr>
<td>(1,2)</td>
<td>-1</td>
</tr>
<tr>
<td>(3,1)</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) Our two perceptron weights have been initialized to \( w_1 = 2 \) and \( w_2 = -2 \). After processing the first point with the perceptron algorithm, what will be the updated values for these weights?

For the first point, \( y = g(w_1 x_1 + w_2 x_2) = g(2 \cdot -1 + -2 \cdot 2) = g(-5) = -1 \), which is incorrectly classified. To update the weights, we add the first data point:

\[
\begin{align*}
  w_1 &= 2 + (-1) = 1 \\
  w_2 &= -2 + 2 = 0
\end{align*}
\]

(b) After how many steps will the perceptron algorithm converge? Write “never” if it will never converge.

Note: one step means processing one point. Points are processed in order and then repeated, until convergence.

The data is not separable, so it will never converge.

(c) Instead of the standard perceptron algorithm, we decide to treat the perceptron as a single node neural network and update the weights using gradient descent on the loss function.

The loss function for one data point is \( \text{Loss}(y, y^*) = (y - y^*)^2 \), where \( y^* \) is the training label for a given point and \( y \) is the output of our single node network for that point.

(i) Given a general activation function \( g(z) \) and its derivative \( g'(z) \), what is the derivative of the loss function with respect to \( w_1 \) in terms of \( g, g', y^*, x_1, x_2, w_1, \) and \( w_2 \)?

\[
\frac{\partial \text{Loss}}{\partial w_1} = 2(g(w_1 x_1 + w_2 x_2) - y^*) g'(w_1 x_1 + w_2 x_2) x_1
\]

(ii) For this question, the specific activation function that we will use is:

\[
g(z) = 1 \text{ if } z \geq 0 \text{ and } = -1 \text{ if } z < 0
\]

Given the following gradient descent equation to update the weights given a single data point. With initial weights of \( w_1 = 2 \) and \( w_2 = -2 \), what are the updated weights after processing the first point?

Gradient descent update equation: \( w_i = w_i - a \frac{\partial \text{Loss}}{\partial w_i} \)

Because the gradient of \( g \) is zero, the weights will stay \( w_1 = 2 \) and \( w_2 = -2 \).

(iii) What is the most critical problem with this gradient descent training process with that activation function?

The gradient of that activation function is zero, so the weights will not update.
Q2. Neural Nets

Consider the following computation graph for a simple neural network for binary classification. Here $x$ is a single real-valued input feature with an associated class $y^\star$ (0 or 1). There are two weight parameters $w_1$ and $w_2$, and non-linearity functions $g_1$ and $g_2$ (to be defined later, below). The network will output a value $a_2$ between 0 and 1, representing the probability of being in class 1. We will be using a loss function $Loss$ (to be defined later, below), to compare the prediction $a_2$ with the true class $y^\star$.

1. Perform the forward pass on this network, writing the output values for each node $z_1$, $a_1$, $z_2$ and $a_2$ in terms of the node’s input values:

   $z_1 = x \ast w_1$
   $a_1 = g_1(z_1)$
   $z_2 = a_1 \ast w_2$
   $a_2 = g_2(z_2)$

2. Compute the loss $Loss(a_2, y^\star)$ in terms of the input $x$, weights $w_i$, and activation functions $g_i$:

   Recursively substituting the values computed above, we have:

   $Loss(a_2, y^\star) = Loss(g_2(w_2 \ast g_1(w_1 \ast x)), y^\star)$

3. Now we will work through parts of the backward pass, incrementally. Use the chain rule to derive $\frac{\partial Loss}{\partial a_2}$. Write your expression as a product of partial derivatives at each node: i.e. the partial derivative of the node’s output with respect to its inputs. (Hint: the series of expressions you wrote in part 1 will be helpful; you may use any of those variables.)

   \[
   \frac{\partial Loss}{\partial w_2} = \frac{\partial Loss}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial w_2}
   \]
4. Suppose the loss function is quadratic, \( \text{Loss}(a_2, y^\star) = \frac{1}{2}(a_2 - y^\star)^2 \), and \( g_1 \) and \( g_2 \) are both sigmoid functions \( g(z) = \frac{1}{1+e^{-z}} \) (note: it’s typically better to use a different type of loss, cross-entropy, for classification problems, but we’ll use this to make the math easier).

Using the chain rule from Part 3, and the fact that \( \frac{\partial g(z)}{\partial z} = g(z)(1 - g(z)) \) for the sigmoid function, write \( \frac{\partial \text{Loss}}{\partial w_2} \) in terms of the values from the forward pass, \( y^\star, a_1, \) and \( a_2 \):

First we’ll compute the partial derivatives at each node:

\[
\frac{\partial \text{Loss}}{\partial a_2} = (a_2 - y^\star) \\
\frac{\partial a_2}{\partial z_2} = \frac{\partial g_2(z_2)}{\partial z_2} = g_2(z_2)(1 - g_2(z_2)) = a_2(1 - a_2) \\
\frac{\partial z_2}{\partial w_2} = a_1
\]

Now we can plug into the chain rule from part 3:

\[
\frac{\partial \text{Loss}}{\partial w_2} = \frac{\partial \text{Loss}}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial w_2} = (a_2 - y^\star) * a_2(1 - a_2) * a_1
\]

5. Now use the chain rule to derive \( \frac{\partial \text{Loss}}{\partial w_1} \) as a product of partial derivatives at each node used in the chain rule:

\[
\frac{\partial \text{Loss}}{\partial w_1} = \frac{\partial \text{Loss}}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_1}
\]

6. Finally, write \( \frac{\partial \text{Loss}}{\partial w_1} \) in terms of \( x, y^\star, w_i, a_i, z_i \): The partial derivatives at each node (in addition to the ones we computed in Part 4) are:

\[
\frac{\partial z_2}{\partial a_1} = w_2 \\
\frac{\partial a_1}{\partial z_1} = \frac{\partial g_1(z_1)}{\partial z_1} = g_1(z_1)(1 - g_1(z_1)) = a_1(1 - a_1) \\
\frac{\partial z_1}{\partial a_1} = x
\]

Plugging into the chain rule from Part 5 gives:

\[
\frac{\partial \text{Loss}}{\partial w_1} = \frac{\partial \text{Loss}}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_1} = (a_2 - y^\star) * a_2(1 - a_2) * w_2 * a_1(1 - a_1) * x
\]

7. What is the gradient descent update for \( w_1 \) with step-size \( \alpha \) in terms of the values computed above?

\[
w_1 \leftarrow w_1 - \alpha(a_2 - y^\star) * a_2(1 - a_2) * w_2 * a_1(1 - a_1) * x
\]
Q3. [Timed: 12mins] Neural Networks: Representation

For each of the piecewise-linear functions below, mark all networks from the list above that can represent the function exactly on the range $x \in (-\infty, \infty)$. In the networks above, $\text{relu}$ denotes the element-wise ReLU nonlinearity: $\text{relu}(z) = \max(0, z)$. The networks $G_i$ use 1-dimensional layers, while the networks $H_i$ have some 2-dimensional intermediate layers.

(a) The networks $G_3, G_4, G_5$ include a ReLU nonlinearity on a scalar quantity, so it is impossible for their output to represent a non-horizontal straight line. On the other hand, $H_3, H_4, H_5$ have a 2-dimensional hidden layer, which allows two ReLU elements facing in opposite directions to be added together to form a straight line. The second subpart requires a bias term because the line does not pass through the origin.

(b) These functions include multiple non-horizontal linear regions, so they cannot be represented by any of the networks $G_i$ which apply ReLU no more than once to a scalar quantity.

The first subpart can be represented by any of the networks with 2-dimensional ReLU nodes. The point of nonlinearity occurs at the origin, so nonzero bias terms are not required.
The second subpart has 3 points where the slope changes, but the networks \( H_i \) only have a single 2-dimensional ReLU node. Each application of ReLU to one element can only introduce a change of slope for a single value of \( x \).

Both functions have two points where the slope changes, so none of the networks \( G_j; H_1, H_2 \) can represent them.

An output bias term is required for the first subpart because one of the flat regions must be generated by the flat part of a ReLU function, but neither one of them is at \( y = 0 \).

The second subpart doesn’t require a bias term at the output: it can be represented as \(-\text{relu}(\frac{-x+1}{2}) - \text{relu}(x + 1)\). Note how if the segment at \( x > 2 \) were to be extended to cross the x axis, it would cross exactly at \( x = -1 \), the location of the other slope change. A similar statement is true for the segment at \( x < -1 \).