Problem Set 2

Please submit a typed PDF addressing all problems below. This problem set contains 3 questions and is worth 25 points. **All responses MUST be in *your* own words.** Justification must be provided for **all** written answers. Statements made without any supporting explanation/justification will receive **no credit**. For mathematical derivations and plots, you may insert pictures of handwritten work if you find this easier. The required weekly readings and lecture slides should be helpful in completing the assignment. You can find these on our course website.

1. **Parameters vs. Hyper-Parameters [6 points]:**

   (a) Define and explain the fundamental difference between parameters and hyper-parameters. How is this difference related to training and validation dataset splits?

   (b) For each of the rows in the below table, indicate which items are parameters (P) and which are hyper-parameters (HP). Justify your answers.

Item	P or HP?
A weight matrix **w**	
The learning rate	
A bias term **b**	
The minibatch size	
The non-linear activation function	
The optimizer	

   (c) Provide examples of two hyper-parameters not present on the table in part (b). Justify your answers.

2. **Overfitting versus Underfitting [4 points]:**

   (a) You're training a deep learning classification model and observe the following:

      i. The validation accuracy increases quickly during the first 5 epochs, then begins to decrease.

      ii. The training loss decreases to zero during the first 10 epochs.

   Is the model overfitting or underfitting? Suggest a possible modification to the hyper-parameters that would improve model generalization.

   (b) You're training a deep learning classification model and observe the following:

      i. The validation accuracy increases slowly during the entire training run of 100 epochs.

      ii. The training loss decreases continually over the entire training run, but never approaches zero.

   Is the model overfitting or underfitting? Suggest a possible modification to the hyper-parameters that could improve model generalization.

3. **Gradient Backpropagation on Computational Graphs [15 points]**: For parts (a) and (b), a computational graph (with input values) is given. Your task for each part is threefold:

   i: Feed the provided input values forward through the computational graph to obtain the predicted value, $\hat{Y}$.

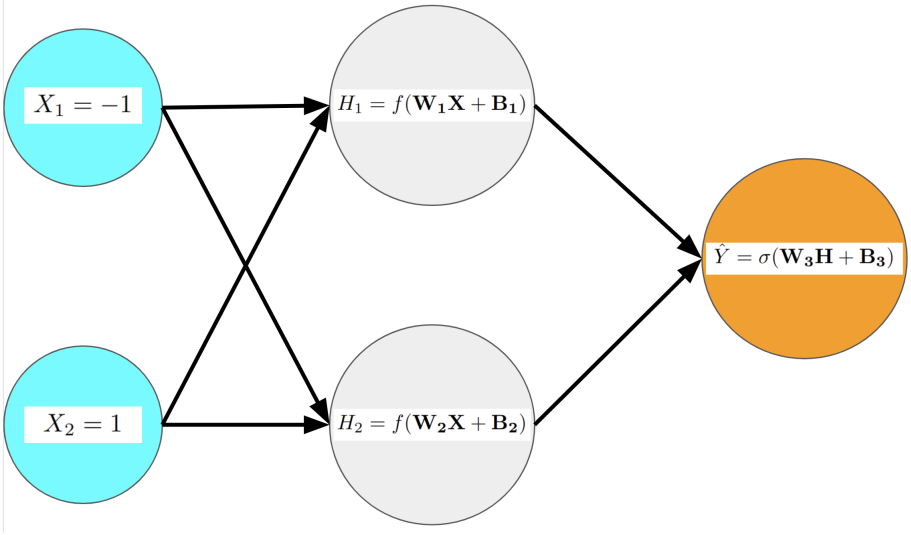
   ii: Compute the binary cross-entropy loss between $\hat{Y}$ and the ground truth value Y.

   iii: Backpropagate the loss computed in part (ii) to obtain gradients for all parameters (W_i, B_i), hidden nodes (H_i), and input nodes (X_i).

   You are only required to perform one forward/backward pass on each graph. Recall that the gradients for parameters/hidden nodes/etc. are partial derivatives of the loss function. In other words, your task is to find the quantities $\frac{dL}{dH}$, $\frac{dL}{dX}$, $\frac{dL}{dW}$, and $\frac{dL}{dB}$ for all relevant H, X, W, B. L denotes the loss. You will need to employ the chain rule for derivatives in order to accomplish this. For all mathematical derivations, you must **show your work**. Note that this is **NOT a programming assignment**; submissions based on automatic differentiation tools (such as PyTorch) will receive **no credit**. The details for each computational graph begin on the next page.

(a) **Standard Neural Network**

Consider the computational graph below:



With the following parameter values:

$$\begin{aligned}\mathbf{W_1} &= [1, -1] \\ \mathbf{B_1} &= 0 \\ \mathbf{W_2} &= [-1, 0.5] \\ \mathbf{B_2} &= 0.5 \\ \mathbf{W_3} &= [1, 1] \\ \mathbf{B_3} &= -1\end{aligned}$$

σ denotes the sigmoid activation function. Use the "Leaky ReLU" non-linear activation function for $f(\cdot)$. Leaky ReLU is widely used in deep learning, and is defined as follows:

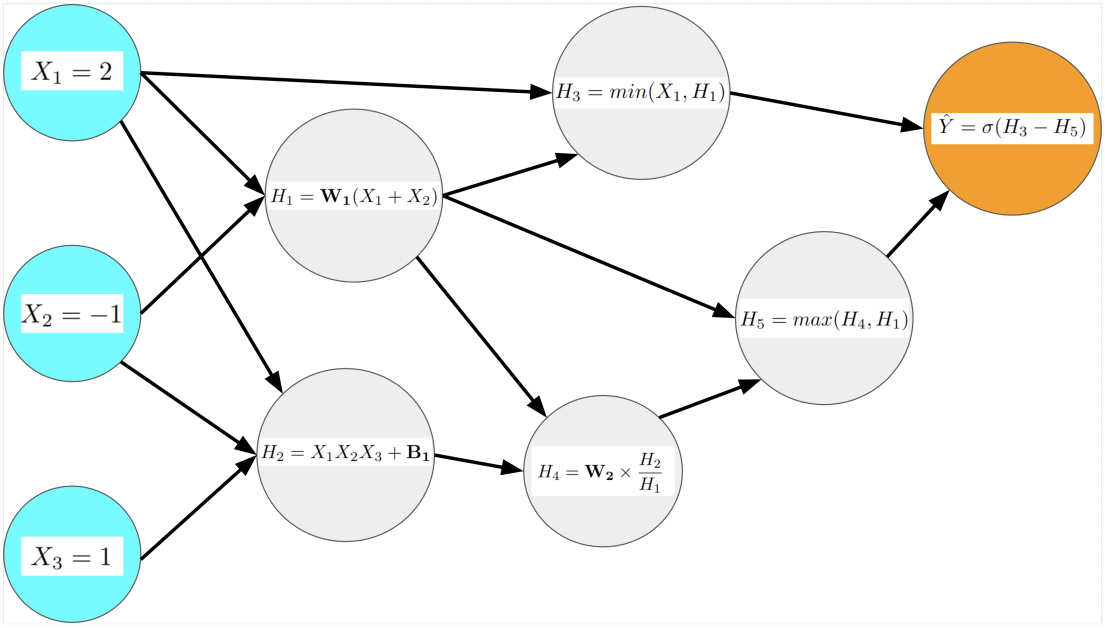
$$f(x) = max(x, 0.1x) \tag{1}$$

For the provided sample, the ground truth label is $Y = 0$. Follow steps (i-iii) to populate the below table with the appropriate values (**Show your work!**):

$\hat{Y}$	L	$\bigtriangledown \mathbf{W_3}$	$\bigtriangledown \mathbf{B_3}$	$\bigtriangledown \mathbf{W_2}$	$\bigtriangledown \mathbf{B_2}$	$\bigtriangledown H_2$	$\bigtriangledown \mathbf{W_1}$	$\bigtriangledown \mathbf{B_1}$	$\bigtriangledown H_1$	$\bigtriangledown \mathbf{X_1}$	$\bigtriangledown \mathbf{X_2}$

(b) **A More "Interesting" Graph**

Consider the computational graph below:



With the following parameter values:

$$\mathbf{W_1} = 1.5$$
$$\mathbf{B_1} = 1$$
$$\mathbf{W_2} = -3$$

σ denotes the sigmoid activation function. For the provided sample, the ground truth label is $Y = 1$. Follow steps (i-iii) to populate the below table with the appropriate values (**Show your work!**):

$\hat{Y}$	L	$\bigtriangledown H_5$	$\bigtriangledown H_4$	$\bigtriangledown H_3$	$\bigtriangledown H_2$	$\bigtriangledown H_1$	$\bigtriangledown \mathbf{W_2}$	$\bigtriangledown \mathbf{W_1}$	$\bigtriangledown \mathbf{B_1}$	$\bigtriangledown X_3$	$\bigtriangledown X_2$	$\bigtriangledown X_1$

You may find the following information useful:

The cross entropy loss function: Like mean-squared error (MSE), the cross entropy function can be used to calculate the loss between the predicted output of a neural network and the target. For the binary classification task, it is defined by:

$$L_{CE}(\hat{Y}, Y) = -Y log(\hat{Y}) - (1 - Y) log(1 - \hat{Y})$$

where $\hat{Y}$ is the prediction from the model, and Y is the target label (0 or 1). The first derivative is given by:

$$\frac{d}{d\hat{Y}} L_{CE} = \frac{\hat{Y} - Y}{\hat{Y}(1 - \hat{Y})}$$

The sigmoid function: The logistic sigmoid function is often used as a non-linearity in neural networks. It is defined by:

$$\sigma(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}}$$

Its first derivative is given by:

$$\frac{d}{dx}\sigma(x) = \frac{1}{1 + e^{-x}} \left(1 - \frac{1}{1 + e^{-x}}\right) = \sigma(x)(1 - \sigma(x))$$

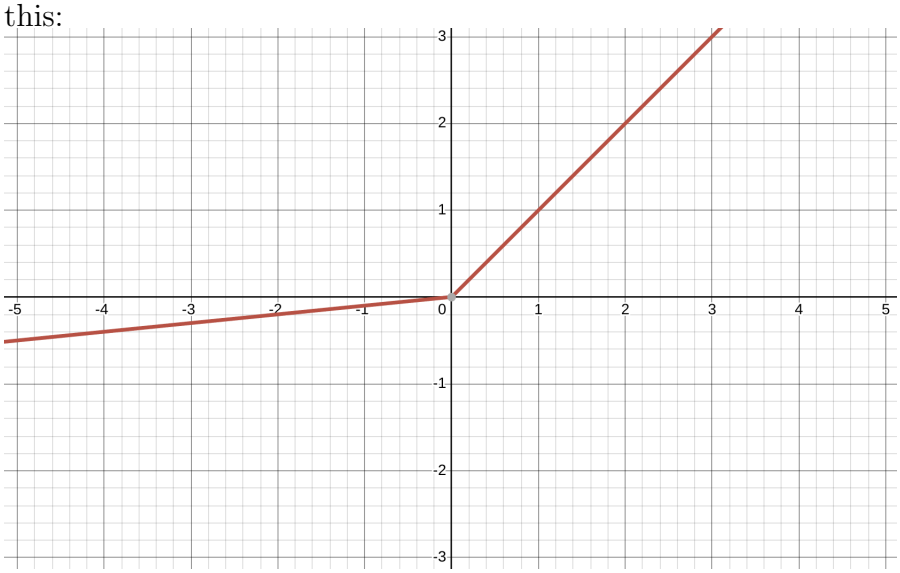
This function is closely related to the hyperbolic tangent function, which is another popular non-linear activation used in deep learning. *tanh* is defined by:

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The two functions are related by the identity:

$$tanh(x) = 2\sigma(2x) - 1 \tag{2}$$

The Leaky ReLU activation function: Leaky ReLU, or (leaky) rectified linear unit, non-linearly modifies the input value depending on its sign. Equation (1) is shorthand for a piece-wise linear function which looks like this:



Examining the graph reveals that Leaky ReLU's derivative is given by:

$$\begin{aligned} f'(x) &= 1, \ x \geq 0 \\ f'(x) &= 0.1, \ \text{otherwise.} \end{aligned} \tag{3}$$

Note that while the slope for negative inputs is commonly 0.1, other variants exist. ReLU (not leaky) uses 0 for this slope, and sometimes other values are used as well.

4. **Extra Credit [2.5 points]:**

   (a) Verify the relationship between *tanh* and σ given by equation (2).

   (b) For both computational graphs, replace σ with *tanh* in the final output "neuron". Compute a new loss using the mean squared error function, which is defined below:

$$MSE(Y, \hat{Y}) = \frac{1}{2}(\hat{Y} - Y)^2 \tag{4}$$

   For the computational graph in Question 3, part (a), use a ground truth value of $Y = -1$ instead of $Y = 0$. Why is this change needed when we are using *tanh* activation?

   (c) Show the steps of back-propagating the new loss through the final output "neuron". You do **NOT** need to back-prop the loss through the entire graphs, just show how the equations change when using the new activation function.

Collaboration versus Academic Misconduct: Collaboration with other students (or AI) is permitted, but the work you submit must be your own. Copying/plagiarizing work from another student (or AI) is not permitted and is considered academic misconduct. For more information about University of Colorado Boulder's Honor Code and academic misconduct, please visit the course syllabus.