CS-449, Spring 2025
Homework #1: Multi-layer Perceptrons
Due Date: Monday, April 28th @ 11:59PM
Total Points: 15.0

You may discuss the homework with other groups, but do not take any written record from the discussions. Each group must submit their own work. Do not submit another group's work as your own, and do not allow another group to submit your work as their own. Also, do not copy any source code from the Web or use code generated by AI.

Part 1

In this part of the assignment, you will work with your group to explore various aspects perceptrons.

Question #1 (2.0 points)

Consider the linear classifier with the weights and decision boundary below. Define $g(X) = X \cdot w$. We'll define our classification prediction $h(X) = sign(g(X))$. You can assume the sign (not sine) function is implemented as in numpy. Using the parameters $w = \langle 1, 0.9, -1.3 \rangle$ shown below, we can calculate $g(X)$ and $h(x)$ as:

$\mathbf{X}$	y	$g(\mathbf{X})$	$h(\mathbf{X})$
$(1, 2)$	1	-0.7	-1
$(2, 1)$	-1	1.5	1
$(2, 3)$	-1	-1.1	-1
$(4, 3)$	1	0.7	1
$(10, 3)$	1	6.1	1

Right now, the classifier gets 60% accuracy. We want to think about how different loss functions could help the model improve. In class, we've discussed the perceptron loss and mean squared error as ways of evaluating the quality of model predictions. First, read up on Hinge loss and Binary Cross Entropy loss to understand how those work. We include definitions below, but it'll be helpful to understand the "margin" aspect of hinge loss and the probabilistic interpretation of BCE loss. Using our $g(X)$ definition above, we can define each loss as the following:

1. Perceptron (zero-one) loss:

$$L(x, y, \theta) = \begin{cases} 0 & \text{if } y \cdot g(x) > 0 \\ 1 & \text{otherwise} \end{cases}$$

2. Squared error loss:

$$L(x, y, \theta) = (y - g(x))^2$$

3. Binary cross-entropy loss (written in an atypical way):

$$L(x, y, \theta) = \begin{cases} \ln(1 + \exp(-g(x)) & \text{if } y > 0 \\ \ln(1 + \exp(g(x)) & \text{otherwise} \end{cases}$$

4. Hinge loss:

$$L(x, y, \theta) = \max(0, 1 - y \cdot g(x))$$

For each loss, answer the following questions. You must provide an explanation for full points.

a. For this loss, which point(s) have the highest loss value? Why?

b. For this loss, which point(s) have the lowest loss value? Why?

c. With this dataset and loss, is it possible to find a different set of model parameters w` that would have a lower total loss? Don't calculate the loss for any specific w` values; just look at a plot and use your intuition. You may answer "Yes", "No", or "Maybe".

   If you answer "Yes", give a general description of what the new decision boundary would look like and why that would decrease the loss (e.g., "if you rotated the current boundary clockwise, then …").

   If you answer "No", give a general argument for why there is no decision boundary that could decrease the loss.

   If you answer "Maybe", give a general description of a new decision boundary that might decrease the loss, but it's hard to tell without doing the calculations. Also, provide a general argument for why there's no decision boundary that obviously decreases the loss.

d. Is this loss function a good choice for training a multilayer perceptron on a binary classification task? Why or why not?

Question #2 (4.0 points)

Augment the dataset used in Question #1 with at least 100 additional observations that are representative of the original dataset. Sample from the augmented dataset to construct train, validation and test splits. For each of the cost functions listed in Question #1, train a Single Layer Perceptron using gradient descent without using PyTorch or any other deep learning platform. You should:

- Discuss all design choices (e.g., architectural, equations, hyper-parameters, hyper-parameter values, etc.) and explain why you made these choices,
- Discuss whether it is possible or not possible to use another learning method (e.g, the perceptron algorithm), why or why not, and the benefits and disadvantages of doing so,
- Use the plotting functions in `part1.ipynb` to generate a graphic of the augmented dataset and the learned decision boundary,
- Add code associated with using gradient descent for each cost function to `part1.ipynb`,
- Include you augmented dataset as a comma-separated values file, and
- Include detailed results on the training, validation and test datasets.

Part 2

In this part of the assignment, you will work with your group to implement and train a multi-layer perceptron (MLP) to perform binary classification on four datasets {`xor, center_surround, two_gaussians, spiral`} using two-dimensional coordinates as features. Example code `part2.ipynb` is provided to read and encode each dataset, construct a toy model, illustrate decision boundaries and display the dataset. You may ignore this code entirely or modify it to use in your submission.

General Guidelines

- Your architecture should consist of a single hidden layer with up to *k* nodes.
- You can use any activation function (*e.g.*, sigmoid, tanh, etc.) in the hidden nodes.
- Your model must use a bias term at the input and hidden layers. It can be a standalone term or be incorporated in the weight matrices.
- You should use gradient descent to train your MLP.
- You may find it helpful to use random number seeds for reproducibility when debugging.
- You do not need to use a GPU for this assignment, and your models should train in less than one minute each.
- You are responsible for selecting hyperparameters (e.g., number of hidden nodes, learning rate, training epochs, batch sizes, early stopping criteria, lambda, etc.). The goal is to get "good" performance from your model, but an exhaustive hyper-parameter search is unnecessary.
- All code, exhibits and answers to free-response questions must be in the `part2.ipynb` Jupyter notebook.
- Your code should use parameters to control all functionality needed to complete specific tasks (see below).

Question #3 (1.0 points) PyTorch Implementation with MCE: Implement and train an MLP as specified above using PyTorch and multi-class cross entropy as the cost function. Experiment with each of the four datasets to find the best number of nodes k from {2, 3, 5, 7, 9} in the hidden layer. For the best k for ***each dataset***:

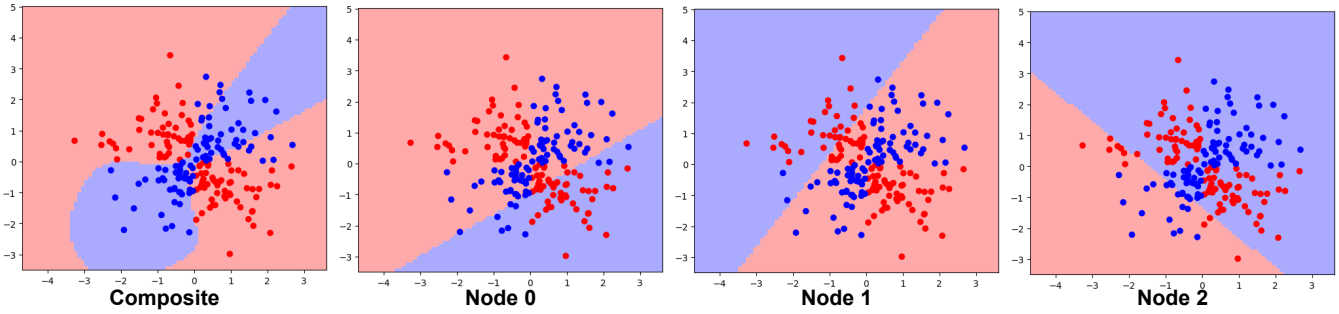
a. list hyper-parameters used in the model,
b. plot the learning curves for training and validation loss as a function of training epochs,
c. provide the final test accuracy, defined as the number of correct classifications divided by the total number of examples,
d. plot the learned decision surface along with observations from the test set (see example below), and
e. discuss your design choices and comment on how they impact performance.

Question #4 (1.0 points) PyTorch Implementation with MSE: Repeat Step 1 using mean-squared error as the cost function. You may want to use a sigmoid function as the output layer of your network and apply a threshold of 0.5 to assign labels at test time.

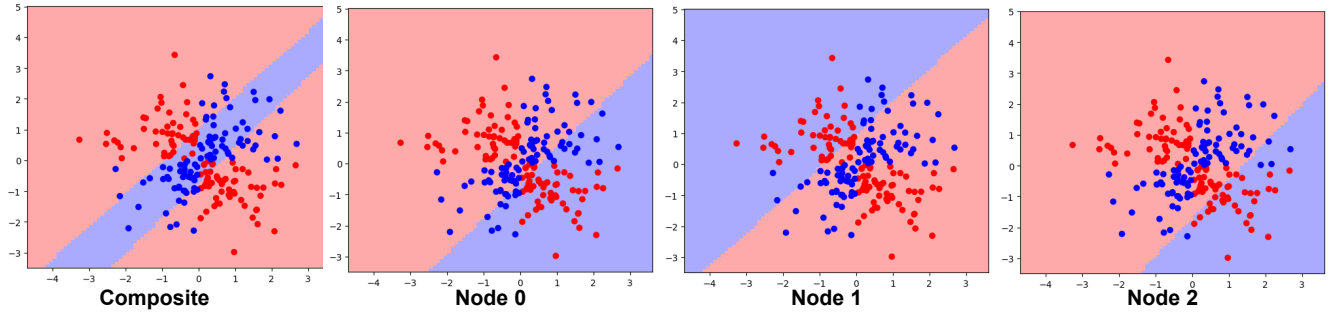
Question #5 (4.0 points) Manual Gradients and Updates: Repeat either Step 1 or Step 2 ***without using a deep learning platform***. You can use symbolic differentiation tools like WolramAlpha, Mathematica, etc., to compute gradients. You can also calculate the gradients by hand. You may want to calculate, code and verify gradients for individual components of your model and use the chain rule to build the gradients for specific weights and biases. You may also want to consider using `for loops` instead of matrix algebra in some parts of your code to avoid the ambiguity of broadcasting. You may find it helpful to "calibrate" intermediate quantities in your implementation against your PyTorch implementation from Step 1 or 2.

Question #6 (3.0 points) Regularizers: Repeat **either** Step 1 or 2 above using a regularizer to encourage *orthogonality* in the intermediate decision boundaries learned in the first layer (note: we discussed this in class as interpreting the output of each hidden node as a single layer perceptron where the weight matrix feeding into a hidden node can be thought of as the *g(x)* in the perceptron). In addition to items a through e listed under step one, plot the intermediate decision boundaries of the regularized and unregularized MLP (see example below). You may find it helpful to work with either the `xor` data with $k = 3$ for MSE, or the `spiral` dataset with $k = 3$ for MCE.

Regularized Dataset: XOR, Nodes: 3, Cost Function: MSE



Unregularized Dataset: XOR, Nodes: 3, Cost Function: MSE



Suggestion: You may find it helpful to look at https://karpathy.github.io/2019/04/25/recipe/ . Despite the blog's title, it is not meant to provide a step-by-step guide to completing this assignment. Instead, it introduces some general principles that you should consider.

Submission Instructions

Turn your Jupyter Notebooks in a single ZIP file under Homework #1 in Canvas.

Good luck, and have fun!