MSAI-349, Fall 2024
Homework #3: Neural Networks
Due Date: Tuesday, November 19th @ 11:59PM
Total Points: 10.0

In this assignment, you will work with your group to implement and train a feed-forward neural network (FFNN) to perform binary classification on four datasets {`xor`, `center_surround`, `two_gaussians`, `spiral`} using two-dimensional coordinates as features. Example code `example.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. You may discuss the homework with other groups, but do not take any written record from these discussions. Also, do not copy any source code from the Web. Your submission must be your own.

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 FFNN.
- 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 a single Jupyter notebook.
- Your code should use parameters to control all functionality needed to complete specific tasks (see below).

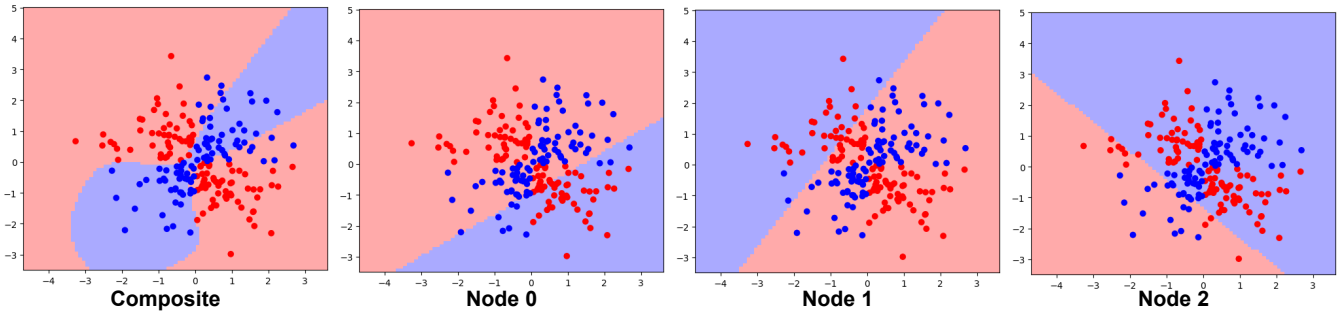
1. **PyTorch Implementation with MCE (2.0 points):** Implement and train a FFNN as specified above using PyTorch and multi-class cross entropy as the cost function. Experiment with each of the four datasets to find the best number of nodes k from {2, 3, 5, 7, 9} in the hidden layer. For the best k for ***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 text 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.

2. **PyTorch Implementation with MSE (2.0 points):** 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.

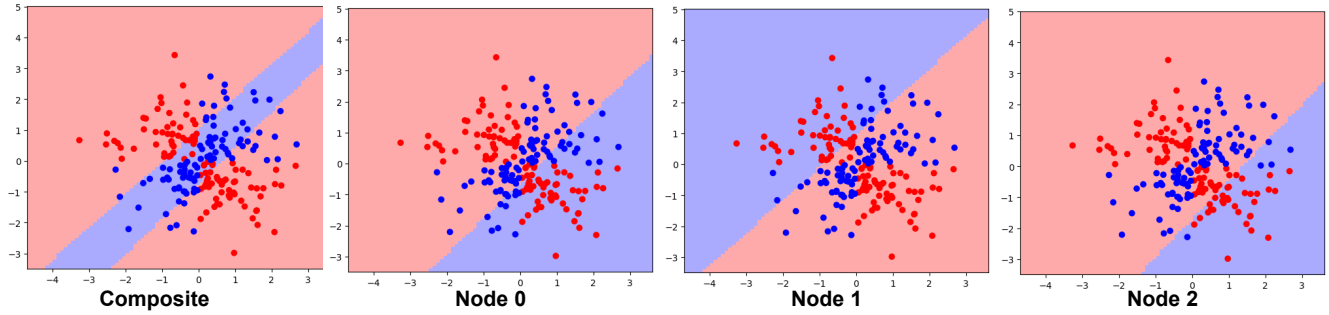
3. **Manual Gradients and Updates (3.0 points):** Repeat either Step 1 or Step 2 ***without using a deep learning platform***. You can use symbolic differentiation tools like WolramAlpha, Mathematica, etc., to compute gradients. You can also calculate the gradients by hand. You may want to calculate, 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.

4. **Regularizers (3.0 points):** Repeat either Step 1 or 2 above, adding two regularizers to your PyTorch implementation. The first regularizer should minimize the norm of the input layer weight matrix. The second regularizer should 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 FFNN (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 a single Jupyter Notebook in a single ZIP file under Homework #3 in Canvas.

Good luck, and have fun!