MONTANA STATE UNIVERSITY

SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

EELE 491 Spring 2026
Problem Set #3

Assigned: January 30, 2026
Due Date: February 6, 2026

This assignment is worth 50 points. It is due at 7:30 p.m. on the posted due date. Late assignments will not be accepted.

1. This MATLAB-based problem will help solidify your understanding of the Single-layer Perceptron.

   Write a MATLAB function called `SLP_forward_pass` that performs the forward pass of a single-layer perceptron, given an input vector, a weight matrix, and a bias. The inputs (in order) should be

   - `in`, a 1xN vector of input values
   - `weights`, an NxM matrix of weights, where `weight(i,j)` represents the strength of the connection between input `i` and output `j`
   - `bias`, a scalar value representing the bias of the nonlinear step function

   and the output should be

   - `out`, a 1xM vector of output values. Each entry in this vector should be either 0 or 1.

2. This MATLAB-based problem will help solidify your understanding of the Miltilayer Perceptron Forward Pass.

   Write a MATLAB function called `MLP_forward_pass` that performs the forward pass of a multilayer perceptron, given an input vector, two weight matrices, and a bias vector. The inputs (in order) should be

   - `x`, a 1xN vector of input values
   - `W1`, an NxD matrix of weights, where `weight(n,d)` represents the strength of the connection between input `n` and hidden node `d`
   - `W2`, a DxM matrix of weights, where `weight(d,m)` represents the strength of the connection between hidden node `d` and output `m`
   - `b`, a 1x2 vector representing the bias of the logistic function for the hidden layer (`b(1)`) and the output layer (`b(2)`)

   and the outputs should be

   - `y`, a 1xM vector of output values
   - `h`, a 1xD vector of hidden node values

   Your function should work for any size of input (N), hidden layer (D), and output (M). For this problem, verify the functionality of the forward pass using the example inputs from the written problem done in class (N = D = M = 2):

   ```
   x = [0.05, 0.10];
   W1 = [0.15, 0.25; 0.20, 0.30];
   W2 = [0.40, 0.50; 0.45, 0.55];
   b = [0.35, 0.60];
   ```

   The answers should be `y = [0.7514 0.7729]` and `h = [0.5933 0.5969]`.

3. This MATLAB-based problem will help solidify your understanding of backprop for the Miltilayer Perceptron.

   Write a MATLAB function called `MLP_backprop` that performs one iteration of backprop for a multilayer perceptron, given an input vector, a target output vector, two weight matrices, a bias vector, and a learning rate. The inputs (in order) should be

   - `x`, a 1xN vector of input values
   - `t`, a 1xM vector of target output values
   - `W1`, an NxD matrix of weights, where `weight(n,d)` represents the strength of the connection between input `n` and hidden node `d`
   - `W2`, a DxM matrix of weights, where `weight(d,m)` represents the strength of the connection between hidden node `d` and output `m`
   - `b`, a 1x2 vector representing the bias of the logistic function for the hidden layer (`b(1)`) and the output layer (`b(2)`)
   - `alpha`, a scalar representing the learning rate for backprop

   and the outputs should be

   - `W1_new`, an updated NxD matrix of weights, where `weight(n,d)` represents the strength of the connection between input `n` and hidden node `d`
   - `W2_new`, an updated DxM matrix of weights, where `weight(d,m)` represents the strength of the connection between hidden node `d` and output `m`
   - `b_new`, an updated 1x2 vector representing the bias of the logistic function for the hidden layer (`b(1)`) and the output layer (`b(2)`)

   Your function should work for any size of input (N), hidden layer (D), and output (M). For this problem, verify the functionality of backprop using the example values from the written problem done in class (N = D = M = 2):

```
x = [0.05, 0.10];
t = [0.01, 0.99];
W1 = [0.15, 0.25; 0.20, 0.30];
W2 = [0.40, 0.50; 0.45, 0.55];
b = [0.35, 0.60];
alpha = 0.5;
```

   From class, you know that the answers should be:

```
W1_new = [0.1498 0.2498; 0.1996 0.2995]
W2_new = [0.3589 0.5113; 0.4087 0.5614]
b_new = [0.3406 0.5498]
```

   Note: The first thing you should do in your `MLP_backprop` is to call your `MLP_forward_pass` from Problem 2 in order to get values for the hidden nodes and output nodes!

4. Run 200 iterations of backprop using the input, target, initial weights, initial biases, and learning rate as the previous problem. Make a plot of the error as a function of backprop iteration.

5. Repeat problem 4 using a random input with 8 dimensions and random binary targets of six dimensions:

   - The input should be `x = randn(1,8);`
   - The target output should be `t = round(rand(1,6));`

   The initial weights and biases should also be initialized randomly, and you should use 12 hidden nodes.

   - `W1 = randn(8,12);`
   - `W2 = randn(12,6);`
   - `b = randn(1,2);`

   Use a learning rate of `alpha = 0.5` and run the code for 200 iterations. Make a plot of the error as a function of iteration.

6. Now let's see if your homemade neural network can work on a 'real' problem! Download the 'hw03_problem06_data.mat' file from Canvas. This file contains the 4-dimensional Fisher Iris data (4 measurements of 150 flowers: 50 each of setosa, versicolor, and virginica) in the `meas` variable. It also contains three label variables:

   - `species` gives the labels as a text value.
   - `t` gives each label as a one-hot encoded three-dimensional vector (setosa = [1 0 0], versicolor = [0 1 0], virginica = [0 0 1]). This is the variable that is used to train the weights and biases in your multilayer perceptron.
   - `y` gives each label as a unique number (setosa = 1, versicolor = 2, virginica = 3). This is the variable that is used to generate the confusion matrix.

   The simplest way to train the MLP using more than one data point is to select one measurement/target pair at random for each iteration of backprop.

   As the AI Engineer, you get to select the learning rate, the size of the hidden layer, and the number of iterations you run.

   Track the error as a function of iteration to make sure that the error is indeed decreasing.

   After training the weights and biases in your MLP, run each of the 150 measurements through the MLP to get the predicted outputs for each flower. Recall that the MLP is a regression model, so an output might look something like `[0.0023, 0.9987, 0.0045]`. You will need to write code that can translate this type of output to a predicted class (1, 2, or 3).

   Once you have a prediction for each flower, create a confusion matrix to compare the predicted labels to the actual labels. Since we did not do any validation scheme, this will be the training performance of your algorithm.