MONTANA STATE UNIVERSITY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

EELE 491 Spring 2026
Problem Set #4

Assigned: February 6, 2026
Due Date: February 13, 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 problem is about activation functions. Consider the following activation functions:

   - $\sigma(x) = \frac{1}{1+e^{-x}}$
   - $\tanh(x)$
   - $\text{ReLU}(x) = \max(0, x)$
   - $\text{swish}(x) = x \cdot \sigma(x)$

   Note that $\frac{\mathrm{d}}{\mathrm{d}x}\sigma(x) = \sigma(x)\left(1 - \sigma(x)\right)$, $\frac{\mathrm{d}}{\mathrm{d}x}\tanh(x) = \text{sech}^2(x)$, and $\frac{\mathrm{d}}{\mathrm{d}x}\text{ReLU}(x) = u(x)$, where $u(x)$ is the unit step function. You can determine $\frac{\mathrm{d}}{\mathrm{d}x}\text{swish}(x)$ on your own using $\frac{\mathrm{d}}{\mathrm{d}x}\sigma(x)$ and the product rule.

   Now, for each of these activation functions:

   (a) Plot the activation function.

   (b) Plot the gradient (derivative) of the activation function.

   (c) Determine if the range of the activation function is bounded or unbounded.

   (d) Determine the regions for which the gradient is likely to be small.

2. This problem is about loss functions. Consider the targets (y) and model outputs ($\hat{y}$) for the following four datasets:

   - Dataset 1 (Regression): $y = [-0.3, 0.5, 0.9]$ and $\hat{y} = [-0.1, 0.4, 1.5]$.
   - Dataset 2 (Regression): $y = [-3, 5, 9]$ and $\hat{y} = [-1, 4, 15]$.
   - Dataset 3 (Classification): $y = [0, 0, 1]$ and $\hat{y} = [0.1, 0.7, 0.9]$.
   - Dataset 4 (Classification): $y = [-1, -1, 1, 1]$ and $\hat{y} = [0.1, -0.7, 0.9, 2.3]$.

   (a) For Dataset 1, compute the MSE, MAE, and Huber Loss (with $\delta = 1.5$).

   (b) For Dataset 2, compute the MSE, MAE, and Huber Loss (with $\delta = 1.5$).

   (c) For Dataset 3, compute the Cross Entropy or Log Loss.

   (d) For Dataset 4, compute the Hinge Loss.

3. This problem is about parameter updates. We are going to perform one single parameter update using three different methods. First off, here are the mathematical definitions of the updates.

   Gradient Descent is defined as $w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w_t}$, where α is a user-defined 'learning rate' parameter.

   Momentum is calculated in two steps:

   - Momentum update: $m_t = \beta m_{t-1} + (1 - \beta) \frac{\partial L}{\partial w_t}$
   - Weight update: $w_{t+1} = w_t - \alpha m_t$

   Momentum requires an initial momentum ($m_0 = 0$) and two user-defined parameters: α (learning rate) and β (momentum rate).

   Adam is a bit more involved, requiring five steps for the update:

   - First moment (mean) estimate: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial w_t}$
   - Second moment (variance) estimate: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left(\frac{\partial L}{\partial w_t} \right)^2$
   - Bias correction (mean): $\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$
   - Bias correction (variance): $\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$
   - Final weight update: $w_{t+1} = w_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$

   Adam requires an initial first moment ($m_0 = 0$) and second moment ($v_0 = 0$), as well as four user-defined parameters: α (learning rate), β_1 (gradient decay rate), β_2 (squared gradient decay rate), and ϵ, a small positive constant used to avoid a divide-by-zero error in the final update.

   For this problem, assume $\alpha = 0.1$, $\beta = 0.9$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $t = 1$, $w_1 = 1.0$, and $\frac{\partial L}{\partial w_1} = 0.5$. Using these hyperparameters, compute w_2 using each of the three methods. Show your work.

4. Last week, you implemented backprop for a multilayer perceptron (MLP) with a single hidden layer, and arbitrary dimensions for the input/hidden/output layers. This week, extend your implementation to support ONE of the following three options:

   - **Option 1:** Multiple hidden layers. Rather than take two weight matrices as an input, backprop should take a cell array of matrices. Consider the example of an input dimension of 3, three hidden layers of dimension [8 16 32], and an output dimension of 2. The cell array should contain four weight matrices of dimensions 3x8, 8x16, 16x32, and 32x2 and a 1x4 vector of bias values. The algorithm should perform a single forward pass (which you will also need to modify), and then a single back-prop, culminating with the update of all weight matrices and the bias vector. Show that your algorithm works by training it on the Fisher Iris dataset from last week using multiple numbers of hidden layers.
   - **Option 2:** Training using minibatches. The user should provide a dataset of training inputs and targets, and the user should specify the minibatch size. Your algorithm should divide the data into minibatches and then train over one single epoch. Then you should implement multiple epochs of training. Show that your algorithm works by training it on the Fisher Iris dataset from last week using a variable minibatch size (that is not a divisor of 150) and 1000 epochs. Plot the error as a function of epoch.
   - **Option 3:** A choice of activation function for the hidden layer (sigmoid, ReLU, tanh, and swish), a choice of activation function for the output layer (sigmoid, ReLU, tanh, and swish), and a choice of loss function (MSE, MAE, Huber Loss). For simplicity, you may have the user select each option using hardcoded integer values (e.g. `hidden=1, output=2, loss=3` could be how the user selects sigmoid, ReLU, and Huber, respectively).

     Note that you will need to implement both the forward-pass and the backprop (gradient) for the activation and loss functions. You may also find it helpful to modify the forward pass so that it returns the inputs to the hidden and output nodes, and not just their outputs. You'll need the node inputs to compute the gradients (unless you use the sigmoid activation function.) Show that your algorithm works by training it on the Fisher Iris dataset from last week using a different activation or cost function than what was used last week.

     When I did this problem, I got terrible results using any activation function other than sigmoid. I also got bad results using MAE as the loss function. However, if I changed only one option at a time, I was able to see that the algorithm was trying to converge. You may want to try a few different scenarios and comment on your results. You should also consider changing your gradient descent step size (α) for the different activation functions.

5. The Research Cyber Infrastructure (RCI) group at MSU has developed a baseline example that creates and trains a convolutional neural network in MATLAB using the MNIST handwritten dataset:

   `https://www.montana.edu/uit/rci/tempest/tempest-examples/matlab-mnist.html`.

   Step 1: Download the data and run the code. The only thing you might need to change is `ExecutionEnvironment="gpu"` to `ExecutionEnvironment="cpu"`, and you only need to do this if your computer does not have a GPU set up for deep learning. (Or if you want to compare GPU and CPU performance...)

   Step 2: Now, alter the network architecture. This is in the long, multiline `layers = [ .... ];` command. Add layers, remove layers, change layers, etc. Keep track of some of the variations you try. What performs the best? What performs the worst? Don't restrict yourself to the layers that already exist in the example code. A full list of deep learning layers supported by MATLAB can be found at `https://www.mathworks.com/help/deeplearning/ug/list-of-deep-learning-layers.html`.

   Be careful on this step. You can easily spend lots of time on this and play around forever. Before training any single alteration, consider thinking about and writing down 10 different combinations you want to try. Then only try those 10 for this HW assignment. If a handful of your combinations fail to compile, that's fine. Report these failures and see if you can understand why. You should be able to report the results for 6-8 successful attempts for this step.

   Step 3: Consider an architecture you found that got an accuracy in the 98% range (not 99.9% like the default option). Can you make it better (higher test score) by changing the training options? These are in the `options = trainingOptions(...);` command, below the layer definition command.

   (If you didn't find an option in the 98% range, just comment out lines 51-59 in the template code—the 16-filter convolutional layer and the 32-filter convolutional layer, as well as their batch normalization, relu, and max pooling layers. Running this gave me a test accuracy of 98.45%.)

   Again, *be careful on this step!* Consider thinking about and writing down 5 parameter combinations you want to try. Then based on those results, select another 5 parameter options. Report your results for at least 10 parameter combinations.

6. Use your favorite LLM to help generate code that trains a basic LSTM. One possible input prompt to the LLM might be: "Please help me write MATLAB code for a very basic Long Short Term Memory neural network. The example should define and use meaningful synthetic input and target time-series data. The code should output visualizations about the training process. The code should also produce plots of the validation target output and the model-predicted output."

   Be sure your code actually runs! You may need to share error messages with your LLM partner. After everything is working, submit your code, as well as important plots of the training process, predicted outputs, and target outputs.