

BME 548L: Homework 2

February 18, 2026

Due date: March 6, 2026 by 11:59pm

Please write up solutions to all problems (however you'd like) and submit them to Gradescope by the above due date. It is important that you show all of your steps – points will be deducted for simply writing the answer without showing any intermediate steps on how you arrived at your answer. You may work together to solve these problems, but please write up your own answer in your own way.

Remember that there is a coding component to this homework assignment as well, which can be found on the website: <https://deepimaging.github.io/homework/hw2>

Problem 1: The magic of the max operator. In this problem, we're going to try to classify a set of 3 different training data points in two dimensions. Here are the example points and their associated labels:

$$\mathbf{x}_1 = \begin{bmatrix} 3/5 \\ 3/5 \end{bmatrix}, y_1 = +1; \mathbf{x}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, y_2 = -1; \mathbf{x}_3 = \begin{bmatrix} -3/5 \\ -3/5 \end{bmatrix}, y_3 = +1$$

- (a) **(5 points)** Let's try to classify these points with a linear classifier, $y^* = \text{sign}(\mathbf{w}^T \hat{\mathbf{X}})$, where \mathbf{w} is the unknown 3×1 weight vector that we'd like to solve for. Remember, we set the first entry of each 3×1 data vector \mathbf{x} to 1 to fold the usual constant offset term b into the inner product between \mathbf{w} and \mathbf{x} . To solve for \mathbf{w} , please construct a 3×3 data matrix $\hat{\mathbf{X}}$ that holds each of our 3 example points, expressed as a vector, in each matrix column. Then, solve for the optimal linear classifier weights \mathbf{w} by computing the pseudo-inverse of $\hat{\mathbf{X}}$, and then by multiplying this pseudo-inverse, \mathbf{X}^\dagger , with a 3×1 vector containing the labels. You can either compute \mathbf{X}^\dagger and the resulting \mathbf{w} by hand, or use a computer to help. If you use the computer, please note that the `np.linalg.pinv` function will be the most helpful here, given the challenging form of $\hat{\mathbf{X}}$.
- (b) **(5 points)** How well do the optimal weights \mathbf{w} do at classifying the original 3 data points? To check this, it is helpful to solve for $\mathbf{y}^* = \text{sign}(\mathbf{w}^T \hat{\mathbf{X}})$, where $\hat{\mathbf{X}}$ is a matrix that holds each of the training data points in its 3 *columns*, and $\mathbf{y}^* \in \mathbb{R}^{1 \times 3}$ is a vector that holds the three classification score estimates. Please sketch the 3 points in 2D space, and discuss why the above classification attempt was or was not so successful.

- (c) **(5 points)** Maybe it'll help to do a convolutional classification instead of a straight linear classification? To test this, let's assume that we've heard that a good convolutional filter to mix this data up a bit is $\mathbf{c} = [-2, 2]$ (written as a row vector here to save space). First, construct a convolution matrix $\mathbf{W}_1 \in \mathbb{R}^{4 \times 3}$ with \mathbf{c} across the diagonal. Then, we'll try out the following classifier,

$$y^* = \text{sign}(\mathbf{w}_2^T \mathbf{W}_1 \hat{\mathbf{X}}), \quad (1)$$

where \mathbf{w}_2 is a 4×1 weight vector of unknown weights. Is it possible to determine a \mathbf{w}_2 that can correctly classify the three points?

- (d) **(5 points)** Finally, let's try to add a non-linearity in there, to see if that will help. Using the same \mathbf{W}_1 as above, let's solve,

$$\mathbf{y}^* = \text{sign} \left(\mathbf{w}_2^T \text{ReLU}(\mathbf{W}_1 \hat{\mathbf{X}}) \right), \quad (2)$$

where \mathbf{w}_2 is again a 4×1 weight vector of unknown weights, and the ReLU operation is the $\max()$ operation that sets all values less than 0 equal to 0. Is it possible to determine a \mathbf{w}_2 that can accurately classify the three points contained in the columns of $\hat{\mathbf{X}}$?

Problem 2: Cost function for logistic classification. We showed in class that the cross-entropy cost function works well for classifying images into outputs that are treated as probability distributions, as opposed to binary "yes/no" categories. This cost function takes the following form for the case of logistic classification:

$$L_{in}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N \ln \left(1 + e^{-y_n \mathbf{w}^T \mathbf{x}_n} \right) \quad (3)$$

- (a) **(6 points)** Noting that the standard logistic function $\theta(x) = e^x / (1 + e^x)$, please compute the gradient of L_{in} and show that,

$$\nabla_{\mathbf{w}} L_{in}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N -y_n \mathbf{x}_n \theta(-y_n \mathbf{w}^T \mathbf{x}_n) \quad (4)$$

- (b) **(6 points)** Based on this computation, please show that a misclassified input contributes more to the gradient than a correctly classified input. This is an important thing to remember - the inputs that are misclassified will be important in driving your gradient descent solver!
- (c) **(bonus 6 points)** Compute the Hessian of $L_{in}(\mathbf{w})$ with respect to \mathbf{w} . Then, use this Hessian to write an expression for the optimal step size to use for a steepest descent solver for the logistic classification problem.

Problem 3: Perceptron Learning Algorithm. A simple iterative method related to solving the logistic classification problem is termed the Perceptron Learning Algorithm. In the simplest form of this approach, we'll start by guessing an initial separation boundary between training data points, where the data points are assigned known labels in one of two categories. The separation boundary is defined by a vector of weights \mathbf{w} , just like we've been considering in class. At iteration t , where $t = 1, 2, \dots$, the weight vector will take on a value $\mathbf{w}(t)$. Then, the algorithm will pick one of the *currently misclassified* training data points $(\mathbf{x}(t), y(t))$ and will use it to update $\mathbf{w}(t)$. Since the example is misclassified, we have $\text{sign}(y(t)) \neq \text{sign}(\mathbf{w}^T(t)\mathbf{x}(t))$. The update rule is,

$$\mathbf{w}(t+1) = \mathbf{w}(t) + y(t)\mathbf{x}(t) \quad (5)$$

The PLA algorithm continues through the iterative loop until there are no longer any misclassified points in the training data set, and will eventually converge to a linear separator for linearly separable data.

- (a) **(6 points)** To show that the weight update rule above moves in the direction of correctly classifying $\mathbf{x}(t)$, first show that

$$y(t)\mathbf{w}^T(t)\mathbf{x}(t) < 0 \quad (6)$$

Hint: $\mathbf{x}(t)$ is misclassified by $\mathbf{w}(t)$

- (b) **(6 points)** Then, show that,

$$y(t)\mathbf{w}^T(t+1)\mathbf{x}(t) > y(t)\mathbf{w}^T(t)\mathbf{x}(t) \quad (7)$$

Hint: Use the update rule here

- (c) **(6 points)** As far as classifying $\mathbf{x}(t)$ is concerned, argue that move from $\mathbf{w}(t)$ to $\mathbf{w}(t+1)$ is a move in the right direction. Feel free to draw a picture of a simple example in 2D if you find that helpful.