CSE7850/CX4803 - Spring 2024 - Homework 2

1. *Machine Learning Basics [12 pts]:*

1. [6pts] Consider the polynomial regression we discussed in our lecture, in which we have the independent variable (feature) x and the dependent variable (target) y and fit a polynomial of degree p, i.e., $y = \theta_p x^p + \theta_{p-1} x^{p-1} + \cdots + \theta_1 x + \theta_0$. How will the training, test, and validation errors change as p increases from 0?

   In polynomial regression, 'p' is the hyperparameter that ultimately influences the complexity of the model. As p increases, the polynomial regression model goes from a linear model (degree 1) that most likely doesn't fit the data as well (underfitting) to a model that better fits the ground truth becoming more flexible as it fits a greater variety of shapes in the data. In terms of training error, we need to keep in mind that if 'p' increases, the model's complexity will increase causing it to better fit the training data. Hence, training error will generally decrease as 'p' increases because the model will fit the training data better. However, the decrease in training error will plateau if 'p' keeps increasing past the optimal 'p' value, and the model may begin to capture the noise in the training data. With too high of a 'p' value, the model will become highly irregular in shape and lead to overfitting. Too high of a 'p' in a polynomial regression model will lead to a highly expressive model that is unable to generalize and in turn makes highly incorrect predictions outside the training dataset.

   In terms of test and validation errors, for very low values of 'p', the model would probably be too simple and may not capture the underlying trend in the data as well. Hence, the model will underfit the data and not generalize will on the training data. Both the validation and test errors will be high because the model doesn't generalize as well. As 'p' starts to increase, the hyperparameter will reach an optimal level causing the model to generalize better to new data. This will in turn decrease validation and test errors. As previously mentioned, if 'p' grows too big beyond the model's optimal complexity, the model will begin overfitting the data and the validation and test errors will begin to increase again. An optimal value of 'p' is crucial in polynomial regression so that the model can generalize well on both training, validation, and test data.

   (2) [6pts] Imagine you are a reviewer for a scientific journal that has received submissions comparing the performance of two machine learning algorithms, A and B, with respective hyperparameters α and β. Each submission makes a claim about the relative performance of these algorithms under certain evaluation settings. Your task is to assess these claims based on the provided evaluation settings and decide whether to recommend acceptance or rejection of the paper.

   1. (a) Claim: Algorithm A outperforms Algorithm B.
      Evaluation Setting: The optimal value of hyperparameter α for Algorithm A was selected by 10-fold cross-validation on the training set, while the default value was used for hyperparameter β of Algorithm B.

Recommendation: Reject

Justification: The claim is rejected because the hyperparameter of algorithm B is not optimized using 10-fold cross validation creating a bias towards algorithm A.

2. (b) Claim: Algorithm A outperforms Algorithm B.
   Evaluation Setting: The performance comparison is based on test errors, with hyperparameters for both algorithms determined through 10-fold cross-validation on the training set.

   Recommendation: Accept

   Justification: The claim is accepted because the claim is fair and rigorous as both algorithms undergo the same method for hyperparameter optimization.

3. (c) Claim: Algorithm A outperforms Algorithm B.
   Evaluation Setting: The results reported correspond to the choices of hyperparameters α for Algorithm A and β for Algorithm B that each produces the best test error.

   Recommendation: Reject

   Justification: Optimizing hyperparameters based on only the best test error risks overfitting and bias, as it does not ensure model generalization beyond the specific test set used, ultimately weakening the rigor of the mode evaluation.

For each scenario, consider the fairness and rigor of the evaluation process described. For each claim and its associated setting listed below, provide a recommendation of "accept" or "reject" and a one- sentence justification for your decision.

2. Gradient Descent [13 pts]

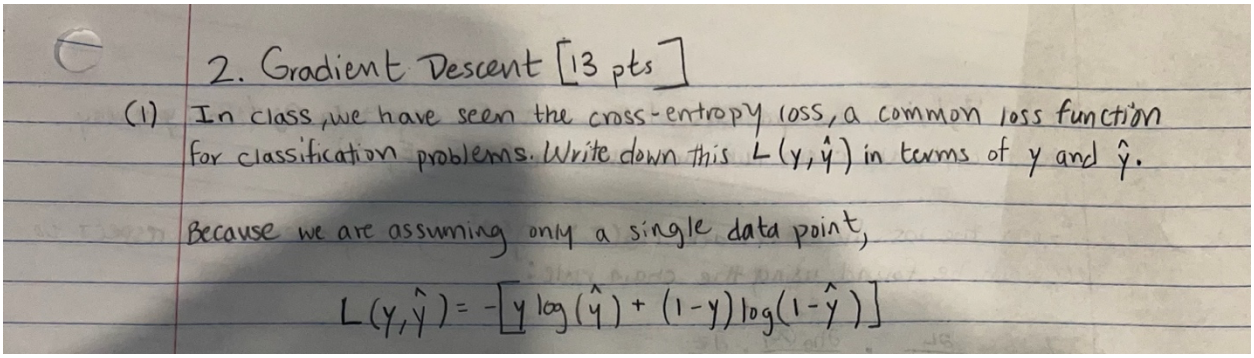
Logistic regression is one of the widely-used model for classification. The logistic regression model $h_\theta(x)$ has

the form

$$\hat{y} = h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}, \quad (1)$$

where the vector x is the input, $\hat{y}$ is the output predicted by the model, and the vector θ is the weights of the model. We simplify the setting such that there is only one training data point (x,y). Note that here $x = [x1,x2,...,xm]^T$ is a m-dimensional features vector of the input and $\theta = [\theta1,\theta2,...,\theta m]^T$ is a vector of weights.

1. (1) [1pt] In class, we have seen the cross-entropy loss, a common loss function for classification problems. Write down this loss $L(y, \hat{y})$ in terms of y and $\hat{y}$.

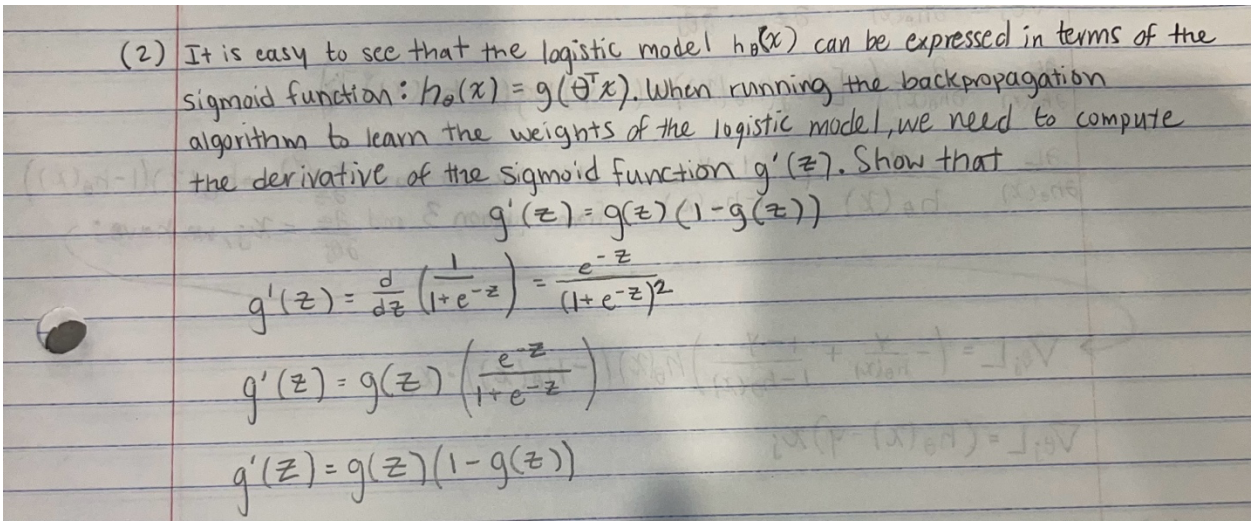


2. (2) [4pts] The logistic regression is closely related to the sigmoid function, which is given by

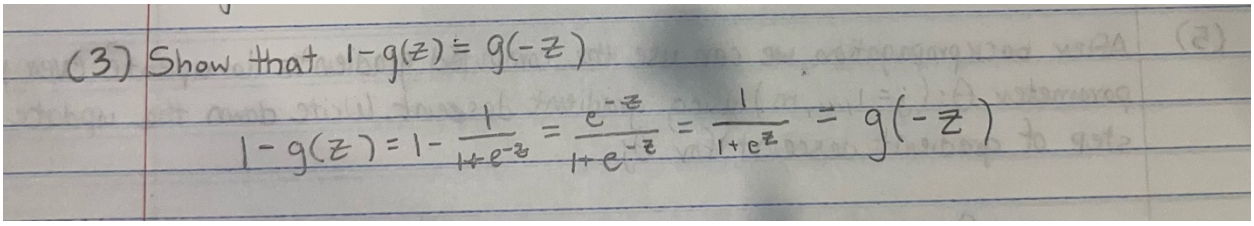
   $g(z) = \frac{1}{1+e^{-z}}$. (2)

   It is easy to see that the logistic model $h\theta(x)$ can be expressed in terms of the sigmoid function: $h\theta(x) = g(\theta^T x)$. When running the backpropagation algorithm to learn the weights of the logistic model, we need to compute the derivative of the simoid function $g'(z)$. Show that

   $g'(z) = g(z)(1 - g(z))$ (3)

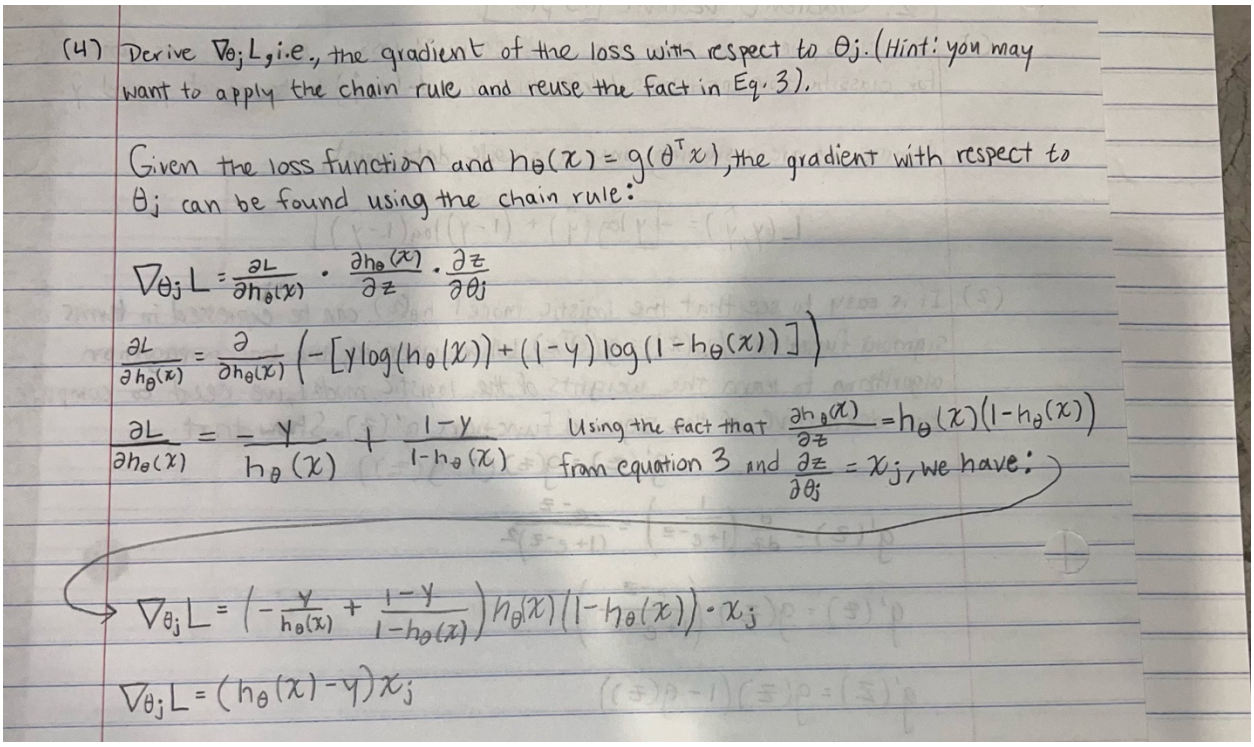


3. (3) [3pts] Show that 1 − g(z) = g(−z).



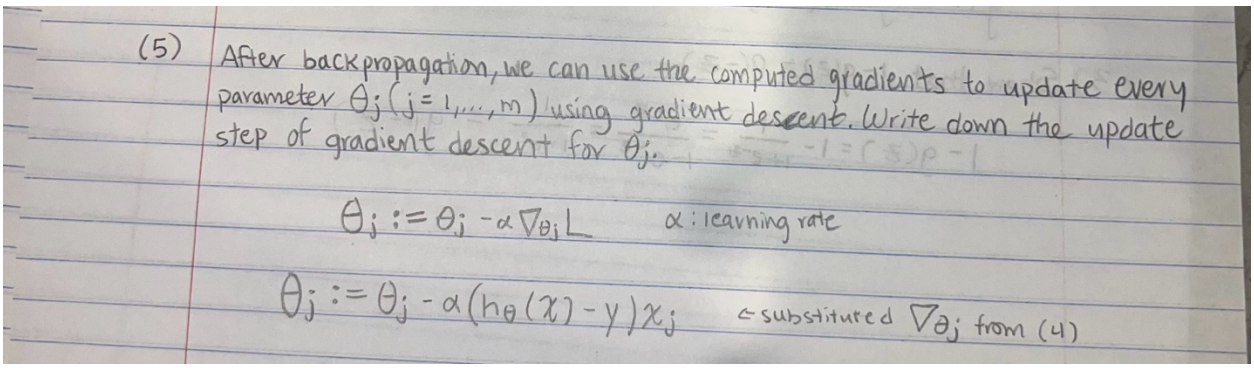
4. (4) [4pts] Derive ∇θj L, i.e., the gradient of the loss with respect to θj . (Hint: you may what to apply the

   chain rule and reuse the fact in Eq. 3).



5. (5) [1pt] After backpropagation, we can use the computed gradients to update every parameter θj (j =

   1, . . . , m) using gradient descent. Write down the update step of gradient descent for θj .



3. *Number of Optimal Alignments [15 pts]*

From our lecture slides or HW1, you may have noticed that for a given input pair of sequences, the optimal alignment is not unique, i.e., there may exist more than one alignment that gives the optimal score. Please give an efficient algorithm that computes the total number of optimal alignments. (Note: you are expected to give an algorithm that is more efficient than the brute-force exhaustive search.) Hint: use dynamic programming!

To accomplish this algorithm, we need to go one step further than the classic Needleman-Wunsch algorithm so that it not only computes the optimal score, but also counts the number of distinct alignments that can achieve that score. The algorithm will consist of creating and filling two matrices that record the best scores and the count of the best scores at each step. I will break up the algorithm into different steps so that it resembles a dynamic programming approach.

We will first start with an initialization process where we first initialize matrix S of size $(m+1) \times (n+1)$ to store scores. The m and n represent the lengths of the first and second sequence to be aligned and the $+1$ represents the additional row and column that store the initial state of the alignment where the sequences are not yet aligned. We will also initialize a matrix C of the same size to store counts of alignments. Set $S[0][0] = 0$ as the base case for building up the rest of the matrix and because it is the score of aligning two empty sequences. $C[0][0] = 1$ is set as the base case for the number of alignments and the alignment of two empty sequences. As the algorithm progresses, the initial base case values will change depending on the cell in the matrices being calculated. For each cell in the first row and column of S, assign gap penalties cumulatively. Similarly, set matrix C values in the first row and column to 1 since there's only one way to align a sequence with gaps up to that cell. To continue the algorithm, we will then have to proceed with the matrix filling steps. Compute and fill the S and C matrices at the same time for each cell (i, j) and calculate the score based on these moves:
For each cell (i, j), calculate the score based on the three possible moves:

- Diagonal (match/mismatch): $S[i-1][j-1] + score\_for\_matching$
- Up (gap in sequence 1): $S[i-1][j] + gap\_penalty$
- Left (gap in sequence 2): $S[i][j-1] + gap\_penalty$
- For $C[i][j]$, if the cell's optimal score comes from:
- Diagonal, increment $C[i][j]$ by $C[i-1][j-1]$
- Up, increment $C[i][j]$ by $C[i-1][j]$
- Left, increment $C[i][j]$ by $C[i][j-1]$

The $score\_for\_matching$ variable is the score added to the current total when the current characters from both sequences being aligned match. The $gap\_penalty$ is the score deducted when a gap is introduced into the alignment.

To consider the tie handling and finding the optimal path, if multiple paths give the same optimal score, add the counts from all paths that lead to the optimal score. For example, if more than one direction (diagonal, top, left) has contributed to the optimal score at $S[i][j]$, then the count at $C[i][j]$ should be the sum of counts from all contributing directions. Since we are trying to compute the count of optimal alignments, no backtracking is required. Therefore, we end the algorithm by referencing to the bottom right cell of the $C[m][n]$ matrix where the final count of optimal alignments will be computed. This algorithm works by making sure all paths which lead to an optimal alignment are counted without duplication.

4. Design a Neural Network by Hand [10 pts]

You have seen in our lectures that the non-linearity and multi-layer structure of a neural network increase the capacity to model complex data. For example, a neural network is able to classify points which cannot be perfectly classified by a linear classifier. In this problem, you will need to design a neural network that is more “powerful” than a linear classifier.

x_1 x_2 y 1 1 -1 1 0 +1 0 1 +1 0 0 -1

x_1	x_2	y
1	1	-1
1	0	+1
0	1	+1
0	0	-1

Table 1: Input data

Table 1: Input data Suppose you a given four points on a 2D plane, $\{x^{(i)},x^{(i)}\}^4$

. Each point is associated with a binary label $y^{(i)} \in \{-1,+1\}$, as shown in Table 1. First plot these points and convince yourself that there is no linear classifier (linear line) that can perfectly classify the four points. Now we want to show that a simple

2

1 2 i=1

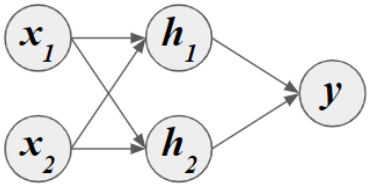


Figure 1: One-layer neural network

neural network, even with only one layer of hidden units, can correctly classify those points. Consider the neural network in Figure 1, which has 2 inputs (x_1 and x_2), 2 hidden nodes (h_1 and h_2), and one output. The network can be summarized by the following functions.

$z_1 = h_1(x_1, x_2) = \text{sgn}(W_{11}x_1 + W_{12}x_2 + b_1)$ $z_2 = h_2(x_1, x_2) = \text{sgn}(W_{21}x_1 + W_{22}x_2 + b_2)$ $o = \text{sgn}(W_{31}z_1 + W_{32}z_2 + b_3)$,

where W_{k1}, W_{k2}, and b_k (k = 1, 2, 3) are the weights and biases of the neural network. Here, we use the sign function sgn as the activation function (sgn(x) returns +1 if x is positive, and -1 otherwise). Please find a setting of all parameters (W_{k1}, W_{k2}, and b_k, k = 1,2,3) for this neural network that will perfect classify the four points.

Submission: Please use a text file "weights.txt" to save your answers of this question. There should be nine numbers in the file, separated by spaces (in one line). The order is W_{11}, W_{12}, b_1, W_{21}, W_{22}, b_2, W_{31}, W_{32}, b_3. Compress the file weights.txt together with other code files that need to be submitted into a zip and upload it to gradescope HW2-Code part.