**CAP 6619 Deep Learning**

**2023 Fall**

Homework 1 [14 Pts, Due: Sept 16 2023. Late Penalty: -2/day]

[If two homework submissions are found to be similar to each other, both submissions will receive 0 grade]

[Homework solutions must be submitted through Canvas. No email submission is accepted. If you have multiple files, please include all files as one zip file, and submit zip file online (only zip, pdf, or word files are allowed). You can always update your submissions. Only the latest version will be graded.]

**Question 1 [1 pt]** Show artificial neuron (perceptron) structure and explain function of each component, including inputs, weights, summing function, activation function, and output [1 pt].

**Ans:**

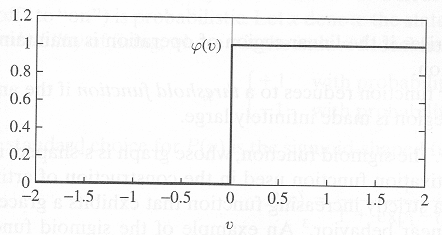
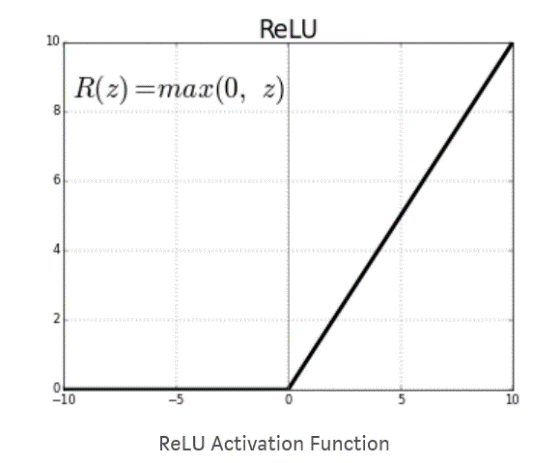
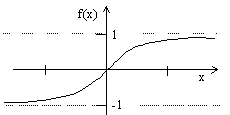
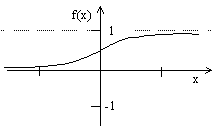
Undoubtedly, a perceptron is one of the most basic artificial neural network models, with just a few parts. Here is a description of its composition and what each part does:

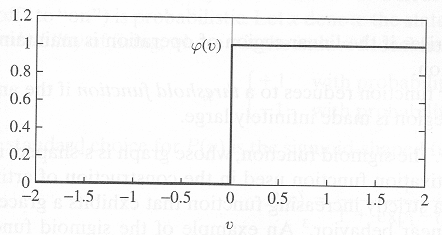
1. **Inputs (x1, x2, ..., xn):** These are the input signals or features that the perceptron receives as inputs which could represent different measures or qualities of the data you are attempting to categorize or analyze in a real-world application.
2. **Weights (w1,w2,…,wn):** Each input has a weight assigned to it, denoted by the letters w1, w2,..., wn. The strength of the link between the input and the perceptron is determined by weights. Consider these to be parameters that the perceptron is trained to learn. They can be either positive

or negative, and they can change throughout training to affect the behavior of the perceptron.

1. **Summation Function (Σ):** The perceptron performs a weighted sum of the inputs and weights. This is often represented as Σ(xi \* wi), where xi is the input and wi is the corresponding weight. The summation function calculates the weighted sum of all inputs.
2. **Activation Function (f):** The summation function's output is then passed via an activation function, denoted by f. Non-linearity is introduced into the perceptron by the activation function, allowing it to learn complicated patterns and make non-binary decisions.
3. **Output (y):** The activation function's output represents the perceptron's ultimate output. In the case of a step function binary classification problem, y will be either 0 or 1, indicating the class to which the input belongs.

**Question 3 [1pt]** Figure 1 shows four activation function in the neural network, please show the mathematical formulation of each activation function [0.5], and explain the characteristics of each activation function [0.5 pt].

1. **** (b) (c) (d)

Figure 1 Activation Functions.

**ANS:**

1. **Mathematical Formulation:**

**Characteristics:**

1. **Mathematical Formulation:**

**V bgHyperbolic Tangent (Tanh) Activation Function:**

**Characteristics:**

Range: The tanh function returns numbers in the range (-1, 1), centered around 0, making it appropriate for applications requiring zero mean outputs.

S-shaped Curve: It has an S-shaped curve, similar to the sigmoid, but with outputs ranging from -1 to 1.

  Zero centered: Tanh is zero-centered, unlike the sigmoid, which can assist minimize the vanishing gradient problem to some extent.

Vanishing Gradient: Deep networks, like sigmoid networks, can suffer from the disappearing gradient problem.

1. **Mathematical Equation:**

**Sigmoid Function:**

**Characteristics:**

Range: The sigmoid function outputs values in the range (0, 1), making it suitable for binary classification problems where the output represents probabilities.

S-shaped Curve: It has an S-shaped curve, and it smoothly squashes input values between 0 and1.

Smooth Gradient: The sigmoid function has a smooth gradient, which can be useful for gradient-based optimization methods like gradient descent.

Vanishing Gradient: It suffers from the vanishing gradient problem, which can slow down training in deep networks

1. **Mathematical Equation:**

**The ReLU activation function:**

The ReLU activation function is defined as: ReLU ( x ) = max ( 0 , x ) ReLU(x)=max(0,x)

Simplicity: ReLU is simple and computationally efficient.

Sparsity: It induces sparsity by setting negative values to zero.

No Vanishing Gradient: ReLU addresses the vanishing gradient problem by having a constant

gradient for positive inputs (unlike sigmoid and tanh).

Dead Neurons: However, ReLU neurons can become "dead" during training if they always

output zero for all inputs (gradient becomes zero).

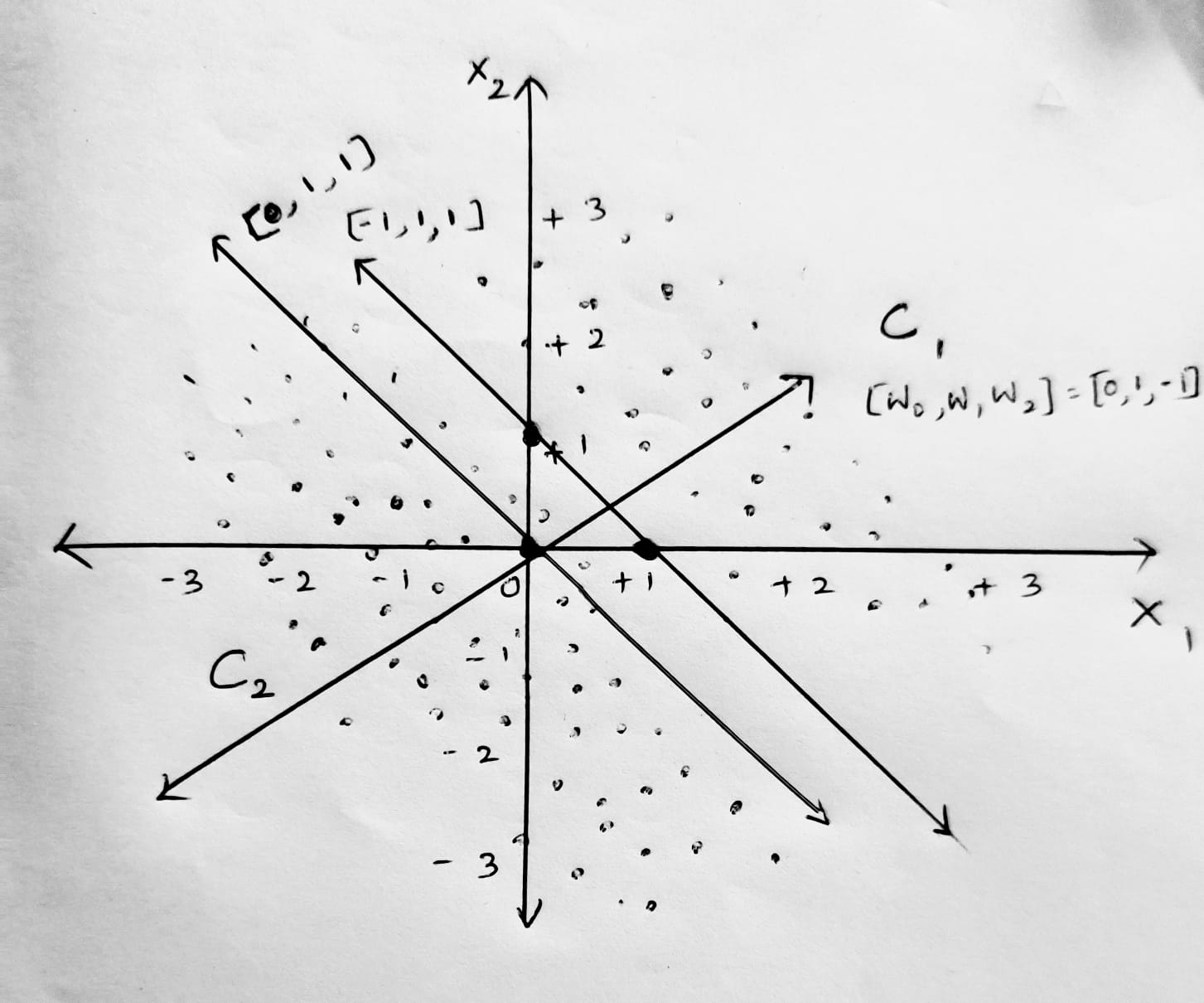
**A diagram of a graph

Description automatically generated**

**Question 4 [2 pts]** Figure 2 shows a set of samples (dots) which are labeled as red and green (red dots belong to class C2, and green dots belong to class C1).

* What are the roles of the weight values of the neuron [0.5 pt]

1. They give us Relative significance of input features. for a particular perceptron.by combining weights. We can normally change or update the weights to get what required. we can get the desired decision line by updated weights.



* Assume a neuron with weight values [w0, w1, w2] is used to learn decision surface to separate the two group instances (w0 is the weight value for bias), Draw decision surfaces corresponding to [0, 1, 1], [0, 1, -1], and [-1, 1, 1], respectively (mark each line on the plot) [0.5 pt]
* Explain how does each weight values [w0, w1, w2] control the decision surface, respectively [0.5 pt]

1. Here [w0, w1, w2] can collectively adjust the position of the line with respective to the origin.
2. Wo is the bias which we will introduce to manipulate the intercept of the decision line.

* Among the three decision surfaces, which line is the best decision surface to separate instances, why? [0.5 pt]

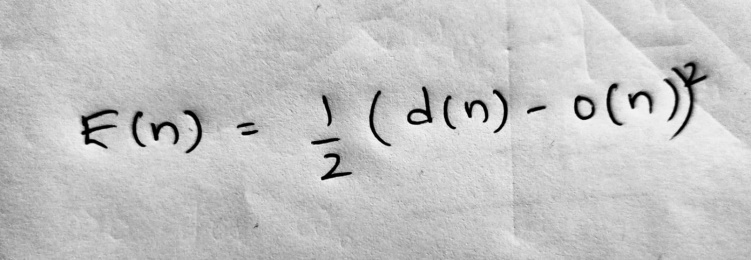
1. The best line to separate the instances is [w0, w1, w2] =[-1,1,1] compared to the other lines .as we can see in the result images as it separates the classes without any errors.

Figure 2. Examples of a linearly separable classification task with two feature dimensions

**Question 5 [1 pt]** Figure 3 shows a single layer neural network with three weight values (including bias). Given a training instance x(n), assume desired label of the instance is d(n),

1. Define squared error of the instance with respect to the network [0.5 pt].

* The squared error for a single-layer neural network is typically defined as the squared difference between the predicted output and the desired target label for a training instance. Let's denote the predicted output as:



b)

1. Use gradient descent learning to derive weight updating rules for w0, w1, and w2, respectively [0.5 pt]

x1

1

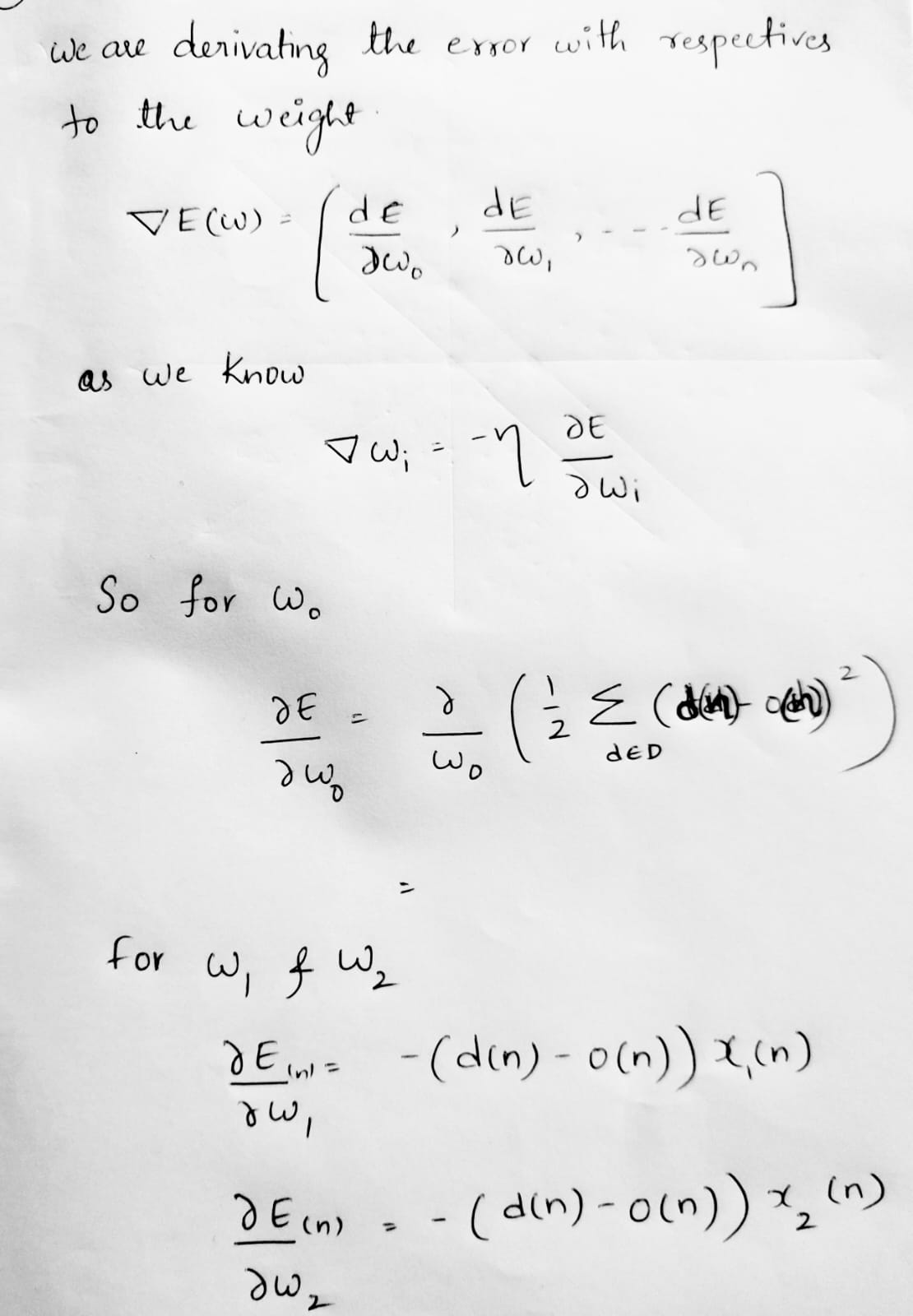
x2

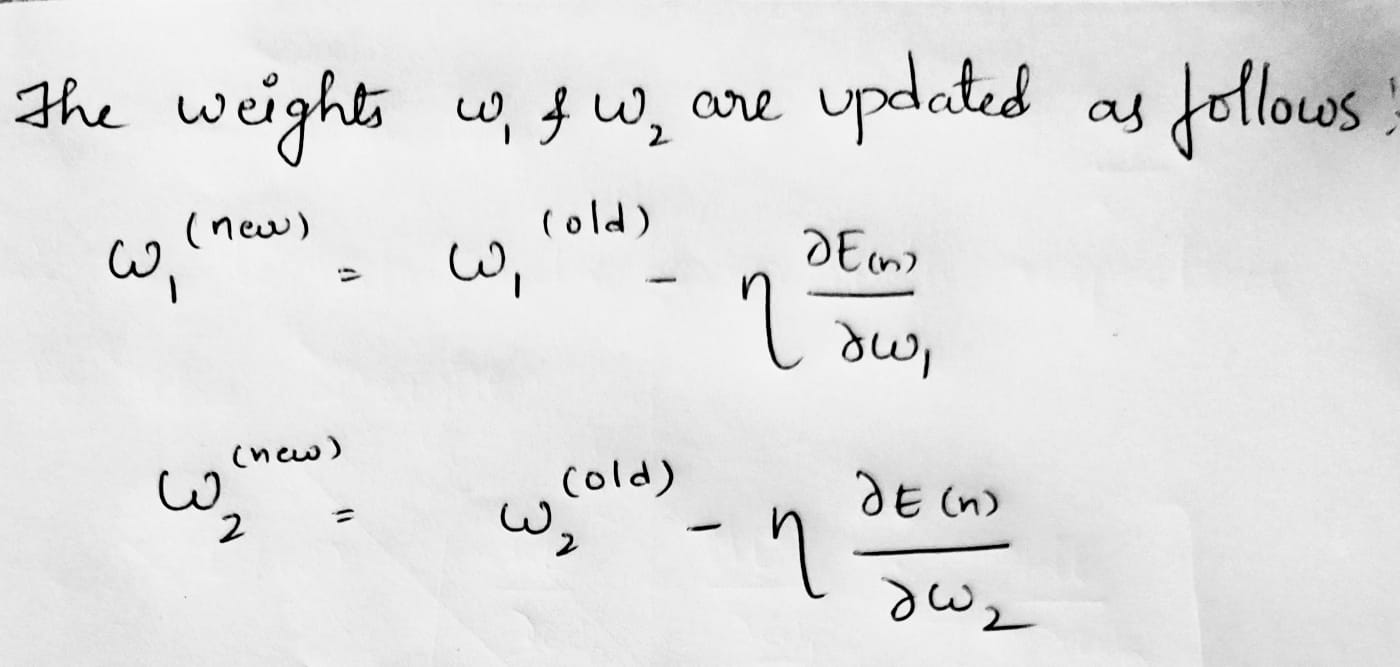
w2

w1

w0

Figure 3: Single layer neural network



****

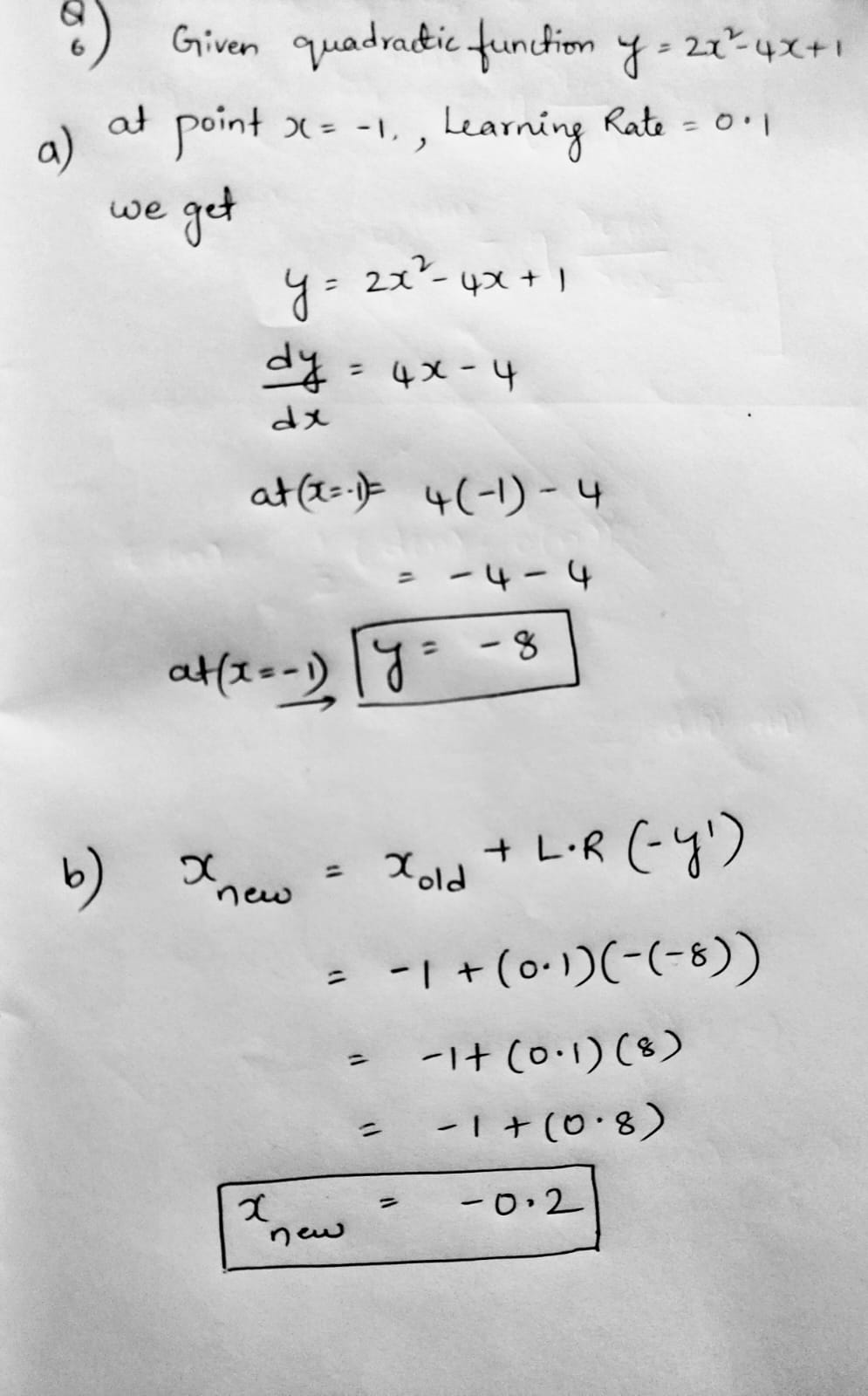
**Question 6 [1 pt]:** The following figure shows a quadratic function y=2x2-4x+1. Assume we are at the point x=-1, and is searching for the next movement to find the minimum value of the quadratic function using gradient descent (the learning rate is 0.1).

* What is the gradient at point x=-1? (Show your solutions) [0.5 pt]
* Following gradient decent principle, find the next movement towards the global minimum [0.5 pt]

Chart, line chart

Description automatically generated

Figure 4. A quadratic function



**Question 7 [1.5 pts]** Assuming we have two sets of instances, which belong to two classes, with each class containing three instances. C1={(1, 0), (1, 1), (0, -1)}; C2={(0, 1), (-1, 0), (-1, -1)}. Assuming the class label for C1 and C2 are 1 and 0, respectively, the learning η=0.1, and the initial weights are *w0*=1, *w1*=1, and *w2*=1. Please use gradient learning rule to learn a linear decision surface to separate the two classes. List the results in the first two rounds by using tables in the following form (Report the mean squared errors of all instances with respect to the initial weight values, and the mean squared errors E(W) AFTER the weight updating for each round).

Mean squared errors E(W) corresponding to the initial weights:

First Round

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input** | **Weight** | **v** | **Desired** | **Output** | **Δw** |
| **(1,1,0)** |  |  |  |  |  |
| **(1,-1,0)** |  |  |  |  |  |
| **(1,0,-1)** |  |  |  |  |  |
| **(1,0,1)** |  |  |  |  |  |
| **(1,1,1)** |  |  |  |  |  |
| **(1,-1,-1)** |  |  |  |  |  |

New weight after first round:

Mean squared errors E(W) after the first-round weight updating:

Second Round

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input** | **Weight** | **v** | **Desired** | **Output** | **Δw** |
| **(1,1,0)** |  |  |  |  |  |
| **(1,-1,0)** |  |  |  |  |  |
| **(1,0,-1)** |  |  |  |  |  |
| **(1,0,1)** |  |  |  |  |  |
| **(1,1,1)** |  |  |  |  |  |
| **(1,-1,-1)** |  |  |  |  |  |

New weight after second round:

Mean squared errors E(W) after the second-round weight updating:

**Question 8 [1.5 pts]** Assuming we have two sets of instances, which belong to two classes, with each class containing three instances. C1={(1, 0), (1, 1), (0, -1)}; C2={(0, 1), (-1, 0), (-1, -1)}. Assuming the class label for C1 and C2 are 1 and 0, respectively, the learning rate η=0.1, and the initial weights are *w0*=1, *w1*=1, and *w2*=1. Use Delta rule (AdaLine) to learn a linear decision surface to separate the two classes. List the results in the first round by using tables in the following form (Please report the mean squared errors of all instances with respect to the initial weight values, and also report the mean squared errors E(W) AFTER the weight updating of the last instance).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Weight** | **v** | **Desired** | **Output** | **Δw** | **New Weight** |
| **(1,1,0)** |  |  |  |  |  |  |
| **(1,-1,0)** |  |  |  |  |  |  |
| **(1,0,-1)** |  |  |  |  |  |  |
| **(1,0,1)** |  |  |  |  |  |  |
| **(1,1,1)** |  |  |  |  |  |  |
| **(1,-1,-1)** |  |  |  |  |  |  |

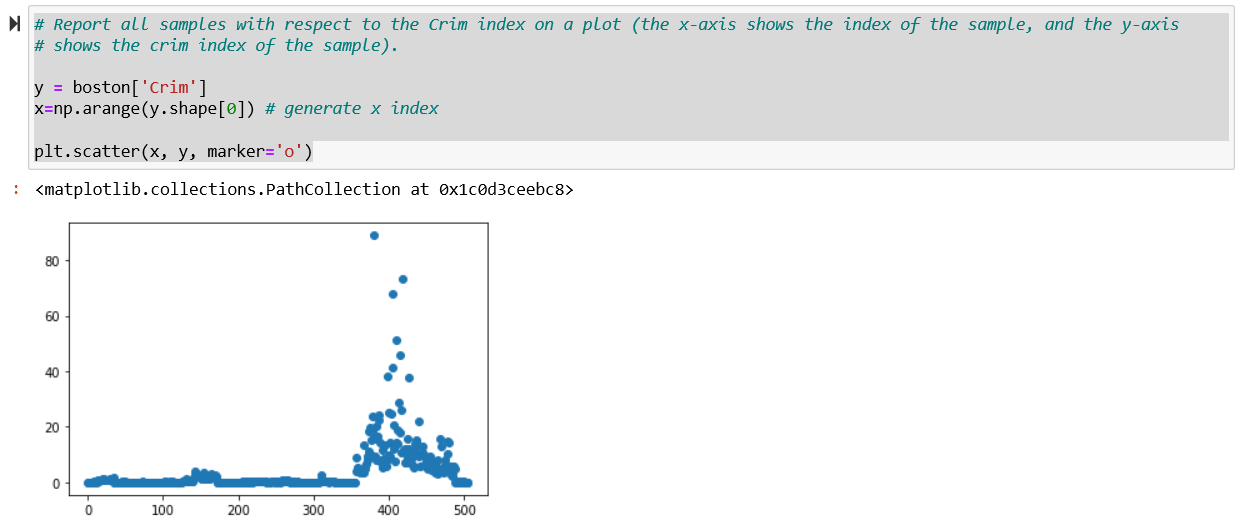
Mean squared errors E(W) after the weight updating of the last instance:

Questions 9 and Question 10 are programming tasks

**For all programming tasks, solutions must be submitted as notebook (html or pdf) files for grading (your submission must include scrips/code and the running results of the script).**

If you are not familiar with Python programming (and want to use Python for the coding tasks), please check Python Plotting notebook and Python Simple Analysis notebook posted in the Canvas, before working the coding tasks.

For each subtask, please use task description (requirement) as comments, and report your coding and results in following format:



**Question 9 [2 pts] programming task:** Please download housing.header.txt dataset from Canvas, and use a programming language (Python, R, etc.) to implement following tasks (a brief description of this dataset is available from the following URL)

<https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html>

1. Read housing.header.txt as a dataframe. Report number of instances and features, and report all samples with respect to the Crim index on a plot (the x-axis shows the index of the sample, and the y-axis shows the crim index of the sample). [0.1 pt]
2. Show both histogram of the Crim index and the density of the Crim index on a 1x2 frame (one row two columns). [0.1 pt]
3. Show following four scatter plots in one frame (1x4), crim-medv, Rm-medv, Age-medv,Tax-medv, and explain how are they (Crim, Rm, Age, Tax) correlated to the medium house value (Medv) [0.1 pt]
4. Create a subset which only includes properties with Crim less than 1 (inclusive), and Rm greater than 6 (inclusive). [0.1 pt]
5. Show a scatter plot between Rm and Medv (x-axis shows Rm and y-axis denote Medv), please color all properties with 7 or larger Rm values as “red”, and rest properties as “black”. [0.2 pt]
6. Create a scatter plot between Rm and Crim and show all 506 properties on the plot. Color property whose Medv value greater or equal to 24 as red, and the rest as blue. [0.2 pt]
7. Report the pairwise correlation between every two variables (either as a matrix or as a level plot) [0.2 pt]
8. Please explain which variable is mostly positively correlated to Medv (medium house value), and which variable is mostly negatively correlated to Medv. [0.2 pt]
9. Draw scatterplots to show relationship between each attribute and Medv, respectively. [0.2 pt]
10. Explain how to use scatterplots to find attributes which are positively correlated, negatively correlated, or independent of Medv, respectively. [0.2 pt]
11. Please create a new instance with following attribute value

Crim=1.0,Zn=0.2,Indus=6,Chas=0.1,Nox=6.5,Rm=5,Age=100,Dis=4.1,Rad=4.5,Tax=21,Ptratio=20,B=300,Lstat=12,Medv=20.5

include this instance into the original dataframe, and report the number of instances and features of the new dataframe. [0.2 pt]

1. Create a new feature (named “Dummy”), and include the new feature into the new dataframe as the last feature. The values of the Dummy feature for each instances are randomly generated within range [0,5]. [0.2 pt]

**Question 10 [2 pts]: programming task**

Given two functions f1(x,y)=(x-2)2 + (y-3)2, and f2(x,y)=(1 - (y-3))2 + 20((x+3) - (y-3)2)2, please implement gradient descent learning to search for global minimum value for each function. Your code needs to implement following components:

* Use two Plots to show f1(x,y) and f2(x,y) values with respect to the two dimensional input x and y (You can specify the range of the x and y values) [0.5 pt]
* Starting from initial value: (x,y)=(0,0), use learning rate η=0.5, report f1(x,y) and f2(x,y) values in T=100 iterations. (your code can report f1(x,y) and f2(x,y) values as tables, or simple print out the values). Explain whether the gradient descent learning is effective finding the solutions for f1(x,y) and f2(x,y), why or why not? [0.5 pt]
* Following step 2, please change your code (e.g., using different learning rates, such as η=0.01) to try to search minimum for f1(x,y) and f2(x,y), respectively. Run algorithms for T=100 iterations Explain the motivation of your changes, and the final minimum values [0.5 pt]
* Explain why gradient descent learning can be used to help search solutions for f1(x,y) and f2(x,y), and what are the impact of the learning rate in the gradient descent learning [0.5 pt]