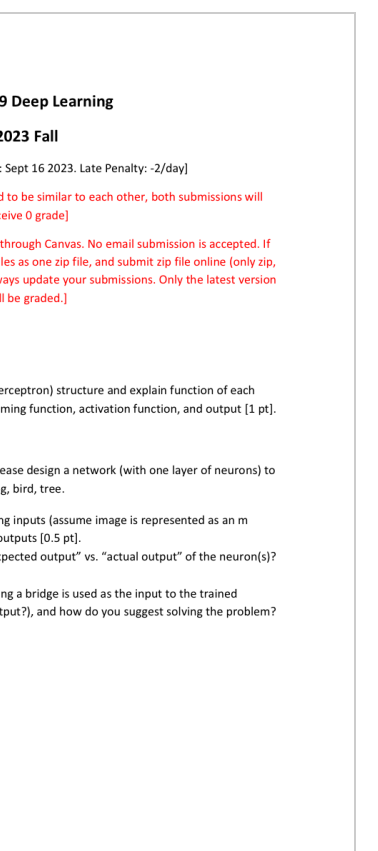
CAP6619_1Homework

Wednesday, September 20, 2023 12:35 AM



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]

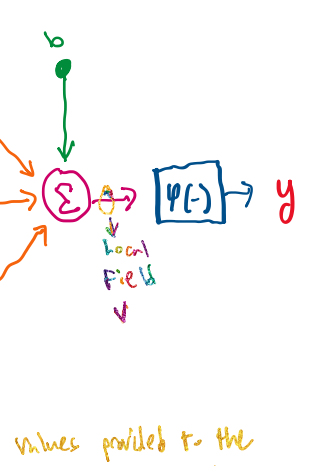
[Homework solutions must be submitted through Canvas. No email submission is accepted. If you have multiple files, please include all files as one zip file, and submit zip file online (only zip, pdf, or word files are allowed). You can always update your submissions. Only the latest version will be graded.]

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].

Question 2 [1 pt]. Given a set of images, please design a network (with one layer of neurons) to classify images into four categories: cat, dog, bird, tree.

- Show network architecture, including inputs (assume image is represented as an m dimensional vector), neurons, and outputs [0.5 pt].
- Given a training sample, what is "expected output" vs. "actual output" of the neuron(s)? [0.25 pt].
- What will happen if an image showing a bridge is used as the input to the trained network? (e.g., What will be the output?), and how do you suggest solving the problem? [0.25 pt].

①



Inputs = Features or values provided to the perceptron. Each input is associated with a weight which determines its contribution to the output. For example, in binary classification, inputs could be features of an object like size, color, weight.

Weights = Weights can be (+) or (−) signifying the direction and magnitude of the influence. Larger weights mean the input has a stronger impact on the output. Adjusting the weights during training is how a perceptron learns and adapts to data.

Bias = Additional parameter in a perceptron that allows it to account for a certain level of activation regardless of the inputs. It serves as an offset, helping to shift the output in the desired direction. The bias can be thought of as a threshold that must be crossed for the perceptron to produce an output.

Summing Function = Computes the weighted sum of the inputs and the bias. It represents the local field, which is the weighted sum of inputs. The formula for computing the local field is:

$$Z = \sum_{i=1}^{n} (\text{input}_i \cdot \text{weight}_i) + \text{bias}$$

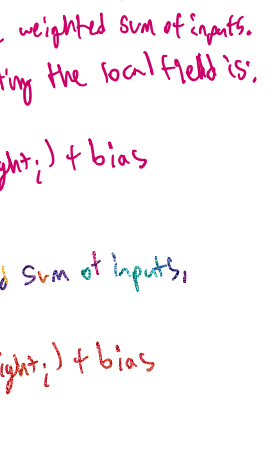
Local Field: Weighted Sum of inputs,

$$Z = \sum_{i=1}^{n} (\text{input}_i \cdot \text{weight}_i) + \text{bias}$$

Activation Function = Determines whether the perceptron should produce an output, and if so, what the output should be. It introduces non-linearity into the perceptron's operation, allowing it to model complex relationships in data. Common activation functions:
1. Sigmoid (logistic) function.
2. Hyperbolic tangent (tanh) function.
3. Rectified linear unit (ReLU) function.

Output = The result of applying the activation function to the local field (Z). If the output exceeds a certain threshold (usually 0 or 1 for binary classification) the perceptron will produce an output indicating one class or another. (e.g. 1 or 0)

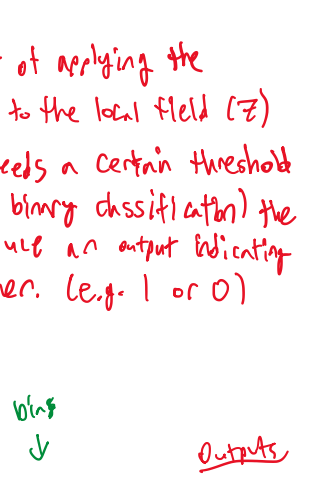
②



Expected output = This is what the input image is labeled as. (i.e. cat, dog, bird, tree)

Actual output = This is what the model predicted. Can be right or wrong and weights can be adjusted by comparing the expected and actual outputs. We want the actual output to match the expected output as often as possible.

If a bridge is fed through the trained network, it will incorrectly classify it as either a dog, cat, bird or tree. Or have equal actual predictions for multiple. (i.e. it is confused). We could either expand the model to include more outputs for larger classification potential, or we could add a 5th output where if there is no clear "winner" from cat, dog, bird, tree it will classify it as the 5th output which could be: "None of these."



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].

(a) (b) (c) (d)

Figure 1 Activation Functions.

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

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

③

(a) Hard-limiter

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

output will be 1 if $v \geq 0$
output will be 0 if $v < 0$

- Only produces two outputs, good for binary classification
- Discontinuity
- Not suitable for gradient descent
- Does not preserve magnitude of input only indicates above or below a threshold.

(b) Sigmoid

$$\varphi(v) = \frac{1}{1+e^{-v}}$$

v is the input to the function, e is the base of the natural logarithm. 2.71828

- Continuous and differentiable everywhere
- Bounded
- Gradient vanish
- Computationally expensive

(c) Hyperbolic Tangent

$$\varphi(v) = \tanh(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}}$$

v is the input to the function, e is the base of natural logarithm

- Continuous and differentiable
- Bounded
- Stronger gradient (steeper derivative values)
- Slightly better than sigmoid at mitigating gradient vanishing
- Zero centered. Avoid bias shift (Eg. All output values become positive)

(d) ReLU

$$\text{ReLU}(x) = \max(0, x)$$

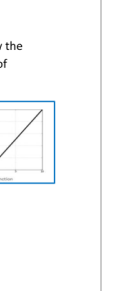
x is the input to function. ReLU returns the maximum of 0 and x

- Does not have vanishing gradient
- Gradient is consistent
- Computationally efficient
- Sparsity, bias from 0
- Continuous, but not differentiable everywhere.
- Not bounded
- Can have "Dead Neurons" – If the input to a ReLU neuron is always less than zero, the neuron will always output 0 and not learn effectively during training.
  Fix this with variations of ReLU (i.e. Leaky ReLU & Parametric ReLU (PReLU))

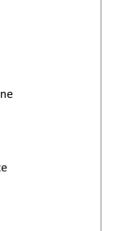
④

- The roles of the weight values of the neuron determine the decision surface. This helps to split the data into different groups.

- [0,1,1]:
  $= 0 \cdot w_0 + 1 \cdot w_1 + 1 \cdot w_2 = 0$
  $\rightarrow w_1 + w_2 = 0$



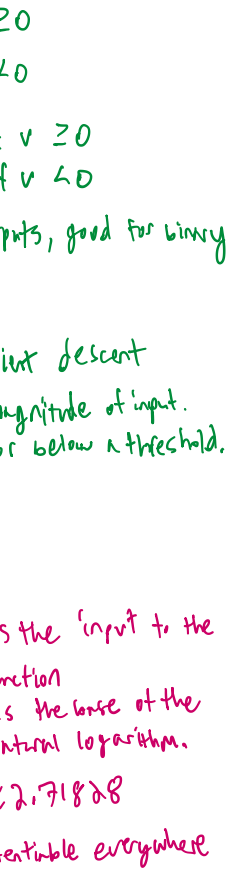
[0,1,-1]:
$= 0 \cdot w_0 + 1 \cdot w_1 - 1 \cdot w_2 = 0$
$\rightarrow w_1 - w_2 = 0$



[-1,1,1]:
$= -1 \cdot w_0 + 1 \cdot w_1 + 1 \cdot w_2 = 0$
$\rightarrow w_1 + w_2 = 1$

- w_0 (bias) = Controls intercepts
  if (+) → shift positive x axis
  if (−) → shift negative x axis
  w_1 & w_2 = Controls rotation (slope) of the decision boundary.

- The best choice for our graph is the weight vector [-1,1,1] because it classifies the data best.



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

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]

Figure 4. A quadratic function

⑤

1) Squared error = $E(n) = \frac{1}{2}[d(n) - y(n)]^2$

$y(n) = 1 \cdot w_0 + x_1 w_1 + x_2 w_2$

$o(x) = y(n)$

This is for a single data point (one set inputs). For multiple data points you would sum these up over entire data set:

$$\frac{1}{2}\sum_{i=1}^{n} [d(n) - (w_0 \cdot 1 + w_1 x_1 + w_2 x_2)]^2$$

*The goal is to update the weights and minimize the MSE using optimization techniques. (i.e. Gradient Descent)

2) *Compute the Gradients of the error (E(n)) with respect to each weight, using chain rule of Calculus

$$\frac{\partial E(n)}{\partial w_0} = -(d(n) - y(n)) \cdot 1$$

$$\frac{\partial E(n)}{\partial w_1} = -(d(n) - y(n)) \cdot x_1$$

$$\frac{\partial E(n)}{\partial w_2} = -(d(n) - y(n)) \cdot x_2$$

α = Learning Rate

*Updating weight rules

$$w_0 = w_0 - \alpha \cdot \frac{\partial E(n)}{\partial w_0}$$

$$w_1 = w_1 - \alpha \cdot \frac{\partial E(n)}{\partial w_1}$$

$$w_2 = w_2 - \alpha \cdot \frac{\partial E(n)}{\partial w_2}$$

⑥

$y = 2x^2 - 4x + 1$

(a) At point $x = -1$

$\alpha = 0.1$

What is the gradient at this point?

$f(x) = 2x^2 - 4x + 1$

$f'(x) = 4x - 4 = 4(x-1)$ ← Gradient formula for the quadratic equation

Plug in x

$\frac{dy}{dx} = 4(-1-1) = -8$ (Gradient)

$\boxed{\text{☒ Gradient at } x = -1 \text{ is } -8}$

(b) ∇ → This symbol means Gradient

Gradient update rule:

$x_{new} = x_{old} - \alpha \cdot \nabla y(x_{old})$

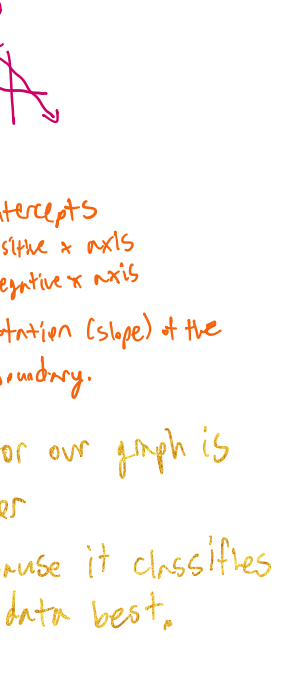
$x_{new} = -1 - 0.1 \cdot 4(-1-1)$

$x_{new} = -1 - 0.1 \cdot -8$

$x_{new} = -1 + 0.8$

$x_{new} = -0.2$

$\boxed{\text{☒ So the next move is to move } x = -1 \text{ to } x = -0.2}$



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: 5.5

First Round

Input	Weight	v	Desired	Output	Δw
(1,1,0)	(1,1,1)	2	1	1	0 0 0
(1,-1,0)	(1,1,1)	0	0	1	0.1 -0.1 0
(1,0,-1)	(1,1,1)	0	1	1	0 0 0
(1,0,1)	(1,1,1)	2	0	1	-0.1 0 -0.1
(1,1,1)	(1,1,1)	3	1	1	0 0 0
(1,-1,-1)	(1,1,1)	-1	0	0	0 0 0

New weight after first round: (0.8, 0.9, 0.9)

Mean squared errors E(W) after the first-round weight updating: 3.8

Second Round

Input	Weight	v	Desired	Output	Δw
(1,1,0)	.8,.9,.9	1.7	1	0	0 0 0
(1,-1,0)	.8,.9,.9	-0.1	0	0	0 0 0
(1,0,-1)	.8,.9,.9	0.1	1	0	-.1 0 0.1
(1,0,1)	.8,.9,.9	1.7	0	1	0.1 0 0.1
(1,1,1)	.8,.9,.9	2.6	1	1	0 0 0
(1,-1,-1)	.8,.9,.9	-1	0	0	0 0 0

New weight after second round: [0.6, 0.9, 0.7]

Mean squared errors E(W) after the second-round weight updating: 3.54

⑦ Round 1

Activation Function:
$v \geq 0 \rightarrow 1$
$v < 0 \rightarrow 0$

① (1,1,1)
$v = 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 = 2 \geq 1$ $\eta = 0.1$ / $\Delta w = \eta \cdot \frac{\partial E}{\partial w_i}$
$\frac{\partial E(n)}{\partial w_0} = -(d_i - v_i) \cdot x_i = -(1-1) \cdot 1 = 0$ $\Delta w_0 = 0.1 \cdot 0$
$\frac{\partial E(n)}{\partial w_1} = -(1-1) \cdot 1 = 0$ $\Delta w_1 = 0.1 \cdot 0$
$\frac{\partial E(n)}{\partial w_2} = -(1-1) \cdot 0 = 0$ $\Delta w_2 = 0.1 \cdot 0$

② (1,1,1)
$v = 1 \cdot 1 + 1 \cdot -1 + 1 \cdot 0 = 0$ $\eta = 0.1$
$\frac{\partial E(n)}{\partial w_0} = -(d_i - v_i) \cdot x_i = -(0-1) \cdot 1 = 1$ $\Delta w_0 = 0.1 \cdot 1 = 0.1$
$\frac{\partial E(n)}{\partial w_1} = -(0-1) \cdot -1 = -1$ $\Delta w_1 = 0.1 \cdot -1 = -0.1$
$\frac{\partial E(n)}{\partial w_2} = -(0-1) \cdot 0 = 0$ $\Delta w_2 = 0.1 \cdot 0 = 0$

③ (1,1,1)
$v = 1 \cdot 1 + 1 \cdot 0 + 1 \cdot -1 = 0 \geq 1$ $\eta = 0.1$
$\frac{\partial E(n)}{\partial w_0} = -(d_i - v_i) \cdot x_i = -(1-1) \cdot 1 = 0$ $\Delta w_0 = 0.1(0) = 0$
$\frac{\partial E(n)}{\partial w_1} = -(1-1) \cdot 0 = 0$ $\Delta w_1 = 0.1(0) = 0$
$\frac{\partial E(n)}{\partial w_2} = -(1-1) \cdot -1 = 0$ $\Delta w_2 = 0.1(0) = 0$

④ (1,1,1)
$v = 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 = 2 \geq 1$ $\eta = 0.1$
$\frac{\partial E(n)}{\partial w_0} = -(d_i - v_i) \cdot x_i = -(0-1) \cdot 1 = 1$ $\Delta w_0 = 0.1(1) = 0.1$
$\frac{\partial E(n)}{\partial w_1} = -(0-1) \cdot 0 = 0$ $\Delta w_1 = 0.1(0) = 0$
$\frac{\partial E(n)}{\partial w_2} = -(0-1) \cdot 1 = 1$ $\Delta w_2 = 0.1(1) = 0.1$

⑤ (1,1,1)
$v = 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 3 \geq 1$ $\eta = 0.1$
$\frac{\partial E(n)}{\partial w_0} = -(d_i - v_i) \cdot x_i = -(1-1) \cdot 1 = 0$ $\Delta w_0 = 0.1(0) = 0$
$\frac{\partial E(n)}{\partial w_1} = -(1-1) \cdot 1 = 0$ $\Delta w_1 = 0.1(0) = 0$
$\frac{\partial E(n)}{\partial w_2} = -(1-1) \cdot 1 = 0$ $\Delta w_2 = 0.1(0) = 0$

⑥ (1,1,1)
$v = 1 \cdot 1 + 1 \cdot -1 + 1 \cdot -1 = -1 \rightarrow 0$ $\eta = 0.1$
$\frac{\partial E(n)}{\partial w_0} = -(d_i - v_i) \cdot x_i = -(0-0) \cdot 1 = 0$ $\Delta w_0 = 0.1(0) = 0$
$\frac{\partial E(n)}{\partial w_1} = -(0-0) \cdot 1 = 0$ $\Delta w_1 = 0.1(0) = 0$
$\frac{\partial E(n)}{\partial w_2} = -(0-0) \cdot 1 = 0$ $\Delta w_2 = 0.1(0) = 0$

$\frac{1}{2}\sum [d-v]^2$
$= \frac{1}{2}[(1-2)^2 + (0)^2 + (1-0)^2 + (0-2)^2 + (1-3)^2 + (0-(-1))^2]$
$= \frac{1}{2}[1 + 0 + 1 + 4 + 4 + 1]$
$= \frac{1}{2}(11) = 5.5 = E(W)$

1(.8) + 1(0.9) + 1(0.9) =
.8 + 0.9 + 0 = 1.7 (1 − 1.7)² = .49
.8 + −0.9 + 0 = −.1 (0 − 0.1)² = .01
.8 + 0 − .9 = −.1 (1 + 0.1)² = 1.21
.8 + 0 + .9 = 1.7 (0 − 1.7)² = 2.89
.8 + 0.9 + .9 = 2.6 (1 − 2.6)² = 2.56
1.2 − 0.9 − .9 = −0.6 (0 − .6)² = .36
7.6
½(7.6) = 3.8

.6 + .9 + 0 (1 − 1.5)² = .25
.6 − .9 + 0 (1 − .3)² = .49
.6 + 0 − .7 = −.1 (1 − .1)² = (.81)
.6 + 0 + .7 (1 − 1.3)² = .09
.6 + .9 + .7 (1 − 2.2)² = 1.44
.6 − .9 − .7 (1 + 1)² = 4
7.08 / 2 = 3.54

Round 2

① (1,1,1)
$v = .8 \cdot 1 + .9 \cdot 1 + .9 \cdot 0 = 1.7 \geq 1$ $\eta = 0.1$ / $\Delta w = \eta \cdot \frac{\partial E}{\partial w_i}$
$\frac{\partial E(n)}{\partial w_0} = -(d_i - v_i) \cdot x_i = -(1-1) \cdot 1 = 0$ $\Delta w_0 = 0.1 \cdot 0$
$\frac{\partial E(n)}{\partial w_1} = -(1-1) \cdot 1 = 0$ $\Delta w_1 = 0.1 \cdot 0$
$\frac{\partial E(n)}{\partial w_2} = -(1-1) \cdot 0 = 0$ $\Delta w_2 = 0.1 \cdot 0$

② (1,1,1)
$v = .8 \cdot 1 + .9 \cdot -1 + .9 \cdot 0 < 0$ $\eta = 0.1$
$\frac{\partial E(n)}{\partial w_0} = -(d_i - v_i) \cdot x_i = -(0-0) \cdot 1 = 0$ $\Delta w_0 = 0.1 \cdot 0 = 0$
$\frac{\partial E(n)}{\partial w_1} = -(0-0) \cdot -1 = 0$ $\Delta w_1 = 0.1 \cdot 0 = 0$
$\frac{\partial E(n)}{\partial w_2} = -(0-0) \cdot 0 = 0$ $\Delta w_2 = 0.1 \cdot 0 = 0$

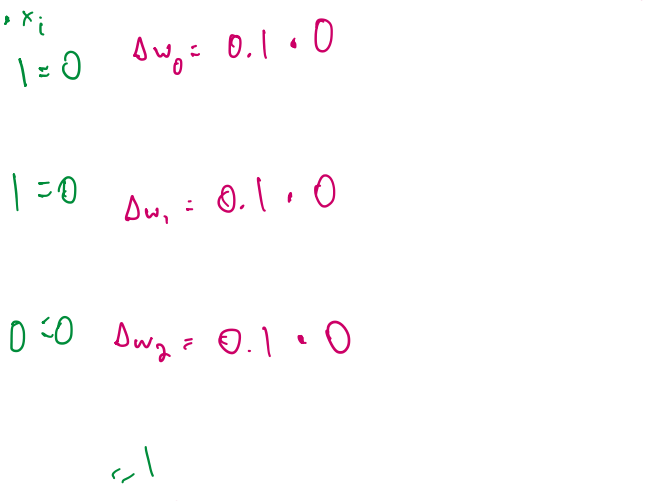
③ (1,1,1)
$v = .8 \cdot 1 + .9 \cdot 0 + .9 \cdot -1 < 1 = 0$ $\eta = 0.1$
$\frac{\partial E(n)}{\partial w_0} = -(d_i - v_i) \cdot x_i = -(1-0) \cdot 1 = -1$ $\Delta w_0 = 0.1(-1) = -0.1$
$\frac{\partial E(n)}{\partial w_1} = -(1-0) \cdot 0 = 0$ $\Delta w_1 = 0.1(0) = 0$
$\frac{\partial E(n)}{\partial w_2} = -(1-0) \cdot -1 = 1$ $\Delta w_2 = 0.1(1) = 0.1$

④ (1,1,1)
$v = .8 \cdot 1 + .9 \cdot 0 + .9 \cdot 1 = 1.7 \geq 1$ $\eta = 0.1$
$\frac{\partial E(n)}{\partial w_0} = -(d_i - v_i) \cdot x_i = -(0-1) \cdot 1 = 1$ $\Delta w_0 = 0.1(1) = 0.1$
$\frac{\partial E(n)}{\partial w_1} = -(0-1) \cdot 0 = 0$ $\Delta w_1 = 0.1(0) = 0$
$\frac{\partial E(n)}{\partial w_2} = -(0-1) \cdot 1 = 1$ $\Delta w_2 = 0.1(1) = 0.1$

⑤ (1,1,1)
$v = .8 \cdot 1 + .9 \cdot 1 + .9 \cdot 1 = 2.6 \geq 1$ $\eta = 0.1$
$\frac{\partial E(n)}{\partial w_0} = -(d_i - v_i) \cdot x_i = -(1-1) \cdot 1 = 0$ $\Delta w_0 = 0.1(0) = 0$
$\frac{\partial E(n)}{\partial w_1} = -(1-1) \cdot 1 = 0$ $\Delta w_1 = 0.1(0) = 0$
$\frac{\partial E(n)}{\partial w_2} = -(1-1) \cdot 1 = 0$ $\Delta w_2 = 0.1(0) = 0$

⑥ (1,1,1)
$v = .8 \cdot 1 + .9 \cdot -1 + .9 \cdot -1 < -1 = 0$ $\eta = 0.1$
$\frac{\partial E(n)}{\partial w_0} = -(d_i - v_i) \cdot x_i = -(0-0) \cdot 1 = 0$ $\Delta w_0 = 0.1(0) = 0$
$\frac{\partial E(n)}{\partial w_1} = -(0-0) \cdot 1 = 0$ $\Delta w_1 = 0.1(0) = 0$
$\frac{\partial E(n)}{\partial w_2} = -(0-0) \cdot 1 = 0$ $\Delta w_2 = 0.1(0) = 0$

$\frac{1}{2}\sum [d-v]^2$
$= \frac{1}{2}[(1-1.7)^2 + (.1)^2 + (.9)^2 + (1.7)^2 + (1.6)^2 + (1)^2]$
$= \frac{1}{2}[.49 + .01 + .81 + 2.89 + 2.56 + 1]$
$= \frac{1}{2}(7.76) = 3.88 = E(W)$



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)	111	2	1	1	0,0,0	111
(1,-1,0)	111	0	0	1	-0.1,0.1,0	0.9,1.1,1
(1,0,-1)	111	0	1	1	0,0,0	111
(1,0,1)	111	2	0	1	-0.1,0,-0.1	0.9,1,0.9
(1,1,1)	111	3	1	1	0,0,0	111
(1,-1,-1)	111	-1	0	0	0,0,0	111

Mean squared errors E(W) after the weight updating of the last instance: 5.5

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.

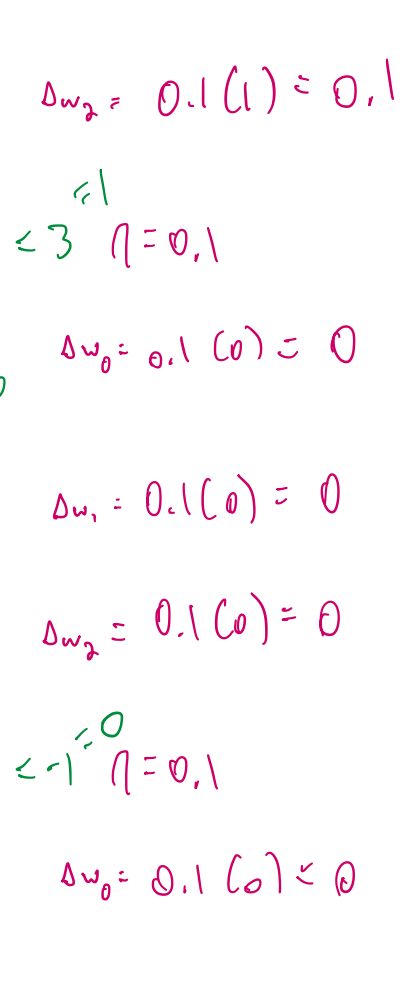
For each subtask, please use task description (requirement) as comments, and report your coding and results in following format:

$\Delta w = 0.1(1-1) * 1 = 0$

$\Delta w = (0.1)(0-1)(1) = -0.1 + 1 = 0.9$

(−0.1) (−1) = 0.1 + 1 = 1.1
(−0.1) (0) = 0 + 1 = 1
(−0.1) (1) = −0.1 + 1 = 0.9
(−0.1) (0) = 0 + 1 = 1
(−0.1) (1) = −0.1 + 1 = 0.9

$\frac{1}{2}\sum (d-v)^2 = (-1)^2 + 0 + 1 + 4 + 4 + 1 = \frac{11}{2}$

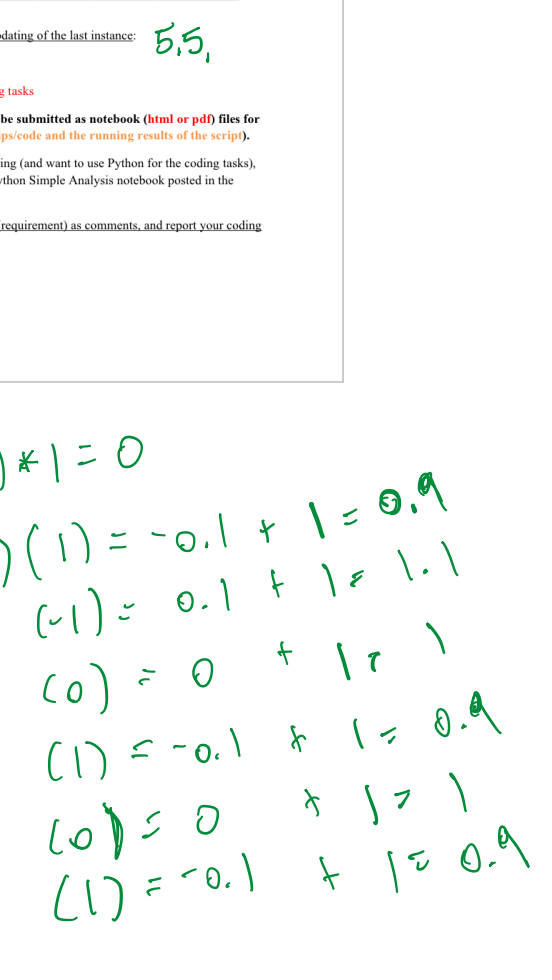


```
# Report all samples with respect to the Crim index on a plot (the x-axis shows the index of the sample, and the y-axis
# shows the crim index of the sample).
y = boston['crim']
x=np.arange(y.shape[0]) # generate x index
plt.scatter(x, y, marker='o')
<matplotlib.collections.PathCollection at 0x...>
```

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]



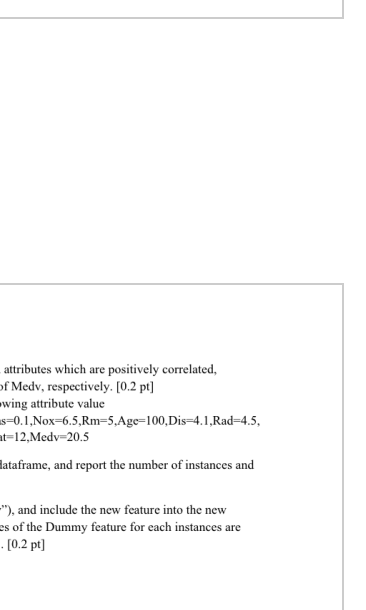
10. Explain how to use scatterplots to find attributes which are positively correlated, negatively correlated, or independent of Medv, respectively. [0.2 pt]
11. Please create a new instance with following attribute value
    Crim=1.0,Zn=0.2,Indus=6,Chas=0.1,Nox=6.5,Rm=5,Age=100,Dis=4.1,Rad=4.5, Tax=21,Ptratio=20,B=300,Lstat=12,Medv=20.5

    include this instance into the original dataframe, and report the number of instances and features of the new dataframe. [0.2 pt]
12. 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]

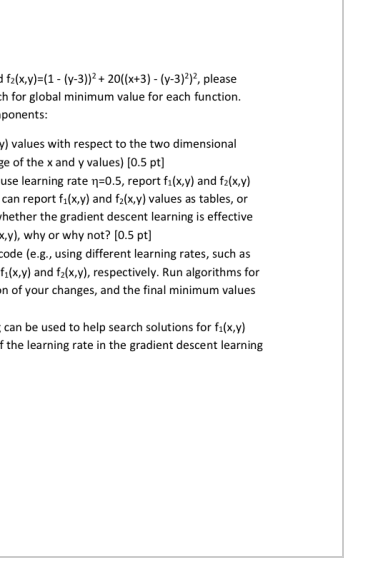
Question 10 [2 pts]: programming task

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:

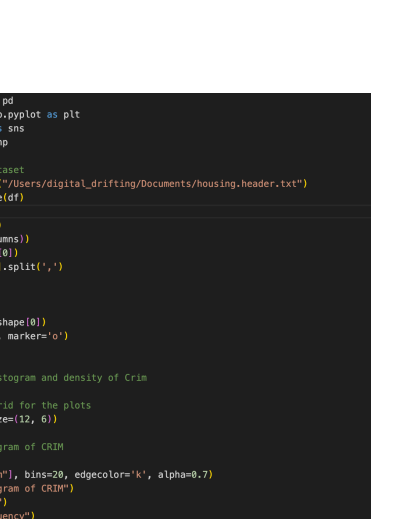
1. Use two Plots to show f₁(x,y) and f₂(x,y) values with respect to the two dimensional input x and y (You can specify the range of the x and y values) [0.5 pt]
2. Starting from initial value: (x,y)=(0,0), use learning rate η=0.5, report f₁(x,y) and f₂(x,y) values in T=100 iterations. (your code can 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]
3. 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]
4. 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]



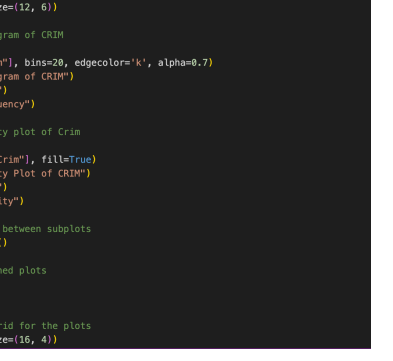
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# 1: Read the dataset
df = pd.read_csv("/Users/digital_drifting/Documents/housing.header.txt")
df = pd.DataFrame(df)
print(df)
print(df.columns)
print(len(df.columns))
print(df.columns[0])
x = df.columns[0].split(',')
print(len(x))
y = df["Crim"]
print(y)
r = np.arange(y.shape[0])
plt.scatter(r, y, marker='o')
#plt.show()

# 2: Plot the histogram and density of Crim

# Create a 1x2 grid for the plots
plt.figure(figsize=(12, 6))

# Plot the histogram of CRIM
plt.subplot(121)
plt.hist(df["Crim"], bins=20, edgecolor='k', alpha=0.7)
plt.title("Histogram of CRIM")
plt.xlabel("CRIM")
plt.ylabel("Frequency")

# Plot the density plot of Crim
plt.subplot(122)
sns.kdeplot(df["Crim"], fill=True)
plt.title("Density Plot of CRIM")
plt.xlabel("CRIM")
plt.ylabel("Density")

# Adjust spacing between subplots
plt.tight_layout()

# Show the combined plots
#plt.show()


# Create a 1x4 grid for the plots
plt.figure(figsize=(16, 4))
```



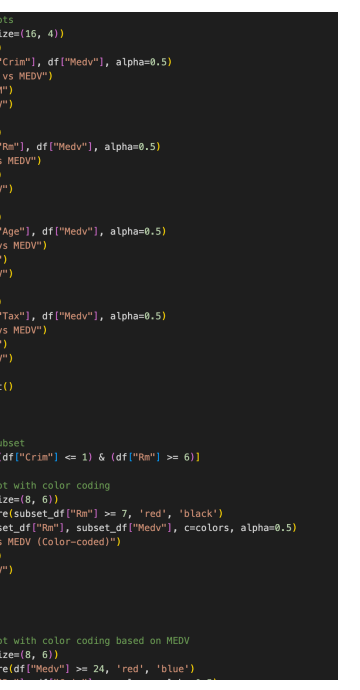
```
    # 3: Scatter plots
    plt.figure(figsize=(16, 4))
    plt.subplot(141)
    plt.scatter(df["Crim"], df["Medv"], alpha=0.5)
    plt.title("CRIM vs MEDV")
    plt.xlabel("CRIM")
    plt.ylabel("MEDV")

    plt.subplot(142)
    plt.scatter(df["Rm"], df["Medv"], alpha=0.5)
    plt.title("RM vs MEDV")
    plt.xlabel("RM")
    plt.ylabel("MEDV")

    plt.subplot(143)
    plt.scatter(df["Age"], df["Medv"], alpha=0.5)
    plt.title("AGE vs MEDV")
    plt.xlabel("AGE")
    plt.ylabel("MEDV")

    plt.subplot(144)
    plt.scatter(df["Tax"], df["Medv"], alpha=0.5)
    plt.title("TAX vs MEDV")
    plt.xlabel("TAX")
    plt.ylabel("MEDV")

    plt.tight_layout()
    #plt.show()


    # 4: Create a subset
    subset_df = df[(df["Crim"] <= 1) & (df["Rm"] >= 6)]

    # 5: Scatter plot with color coding
    plt.figure(figsize=(8, 6))
    colors = np.where(subset_df["Rm"] >= 7, 'red', 'black')
    plt.scatter(subset_df["Rm"], subset_df["Medv"], c=colors, alpha=0.5)
    plt.title("RM vs MEDV (Color-coded)")
    plt.xlabel("RM")
    plt.ylabel("MEDV")

    #plt.show()


    # 6: Scatter plot with color coding based on MEDV
    plt.figure(figsize=(8, 6))
    colors = np.where(df["Medv"] >= 24, 'red', 'blue')
```

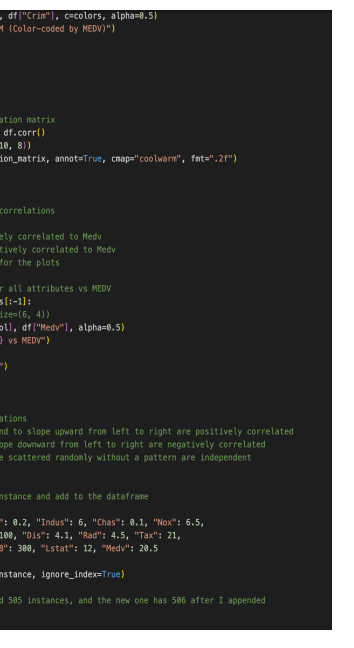


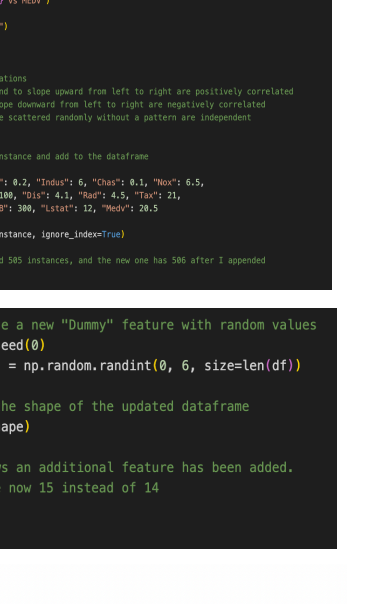
```
plt.scatter(df["Rm"], df["Crim"], c=colors, alpha=0.5)
plt.title("RM vs CRIM (Color-coded by MEDV)")
plt.xlabel("RM")
plt.ylabel("CRIM")

#plt.show()


# 7: Pairwise correlation matrix
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")

#plt.show()

# 8: Explanation of correlations

# Rm is most positively correlated to Medv
# Lstat is most negatively correlated to Medv
# Create a 1x2 grid for the plots

# 9: Scatterplots for all attributes vs MEDV
for col in df.columns[:-1]:
    #plt.figure(figsize=(6, 4))
    plt.scatter(df[col], df["Medv"], alpha=0.5)
    plt.title(f"{col} vs MEDV")
    plt.xlabel(col)
    plt.ylabel("MEDV")

#plt.show()

# 10: Explain correlations
# Datapoints that tend to slope upward from left to right are positively correlated
# Datapoints that slope downward from left to right are negatively correlated
# Datapoints that are scattered randomly without a pattern are independent


# 11: Create a new instance and add to the dataframe
new_instance = {
    "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
}
df = df.append(new_instance, ignore_index=True)

# The original df had 505 instances, and the new one has 506 after I appended
#print(df)
```

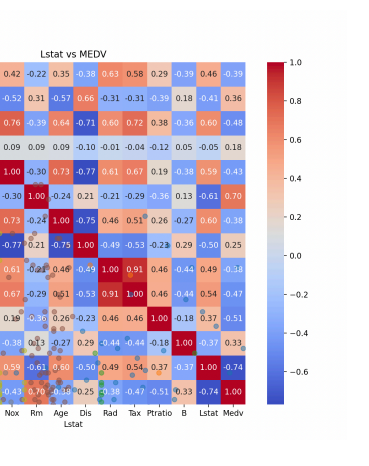


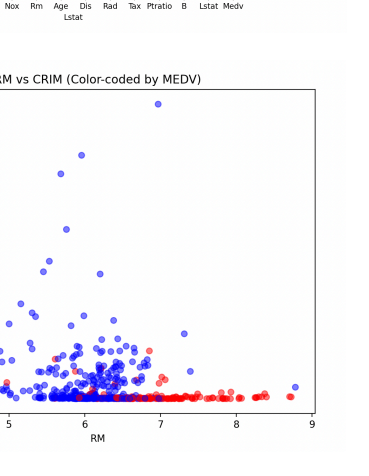
```
    # 12: Create a new "Dummy" feature with random values
    np.random.seed(0)
    df["Dummy"] = np.random.randint(0, 6, size=len(df))

    # Display the shape of the updated dataframe
    print(df.shape)

    # This shows an additional feature has been added.
    # There are now 15 instead of 14
```

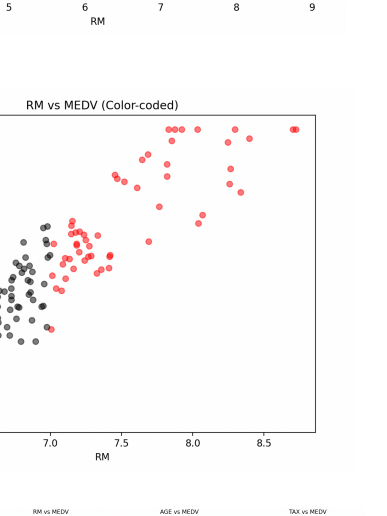


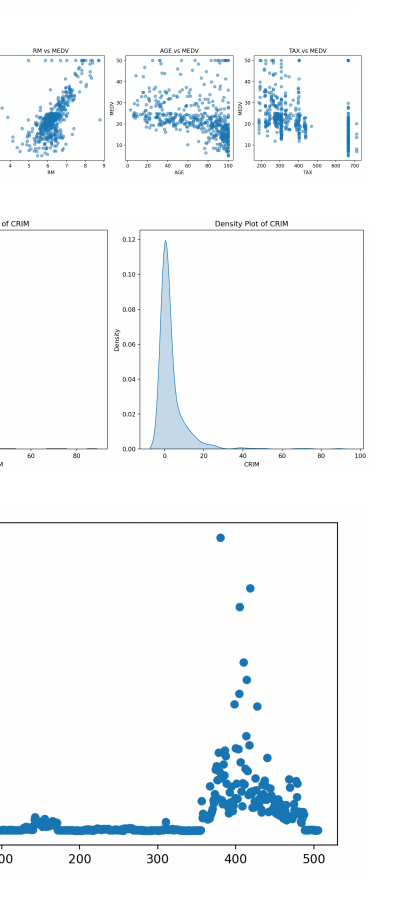




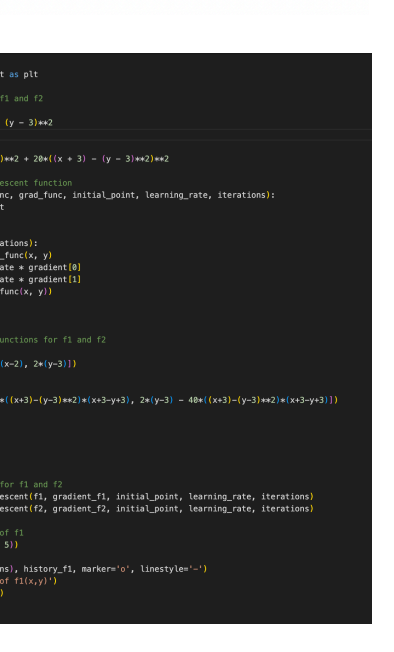


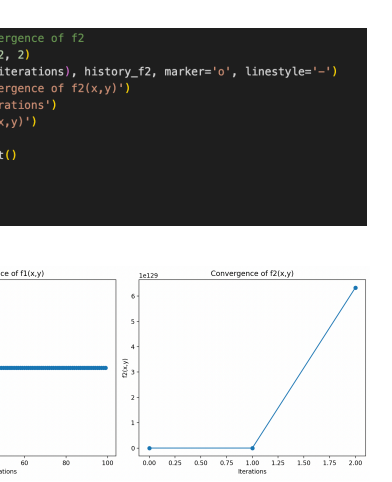






```
import numpy as np
import matplotlib.pyplot as plt

# Define the functions f1 and f2
def f1(x, y):
    return (x - 2)**2 + (y - 3)**2

def f2(x, y):
    return (1 - (y - 3))**2 + 20*((x + 3) - (y - 3)**2)**2

# Define the gradient descent function
def gradient_descent(func, grad_func, initial_point, learning_rate, iterations):
    x, y = initial_point
    history = []

    for _ in range(iterations):
        gradient = grad_func(x, y)
        x -= learning_rate * gradient[0]
        y -= learning_rate * gradient[1]
        history.append(func(x, y))

    return history

# Define the gradient functions for f1 and f2
def gradient_f1(x, y):
    return np.array([2*(x-2), 2*(y-3)])

def gradient_f2(x, y):
    return np.array([40*((x+3)-(y-3)**2)*(x+3-y+3), 2*(y-3) - 40*((x+3)-(y-3)**2)*(x+3-y+3)])

# Define parameters
initial_point = (0, 0)
learning_rate = 0.5
iterations = 100

# Run gradient descent for f1 and f2
history_f1 = gradient_descent(f1, gradient_f1, initial_point, learning_rate, iterations)
history_f2 = gradient_descent(f2, gradient_f2, initial_point, learning_rate, iterations)

# Plot the convergence of f1
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(iterations), history_f1, marker='o', linestyle='-')
plt.title('Convergence of f1(x,y)')
plt.xlabel('Iterations')
plt.ylabel('f1(x,y)')
```



```
# Plot the convergence of f2
plt.subplot(1, 2, 2)
plt.plot(range(iterations), history_f2, marker='o', linestyle='-')
plt.title('Convergence of f2(x,y)')
plt.xlabel('Iterations')
plt.ylabel('f2(x,y)')

plt.tight_layout()
plt.show()
```