import numpy as np

import matplotlib .pyplot as plt

import math

from matplotlib.animation import FuncAnimation

from mpl\_toolkits.mplot3d import Axes3D

from IPython.display import HTML

# Data

X = np.array([0.245, 0.247, 0.285, 0.299, 0.327, 0.347, 0.356, 0.36, 0.363, 0.364, 0.398, 0.4, 0.409, 0.421, 0.432, 0.473, 0.509, 0.529, 0.561, 0.569, 0.594, 0.638, 0.656, 0.816, 0.853, 0.938, 1.036, 1.045])

y = np.array([0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1])

# Theta parameters and learning rate

theta0 = 0

theta1 = 0

learning\_rate = 1e-2

iterations = 3000

# Logistic function

def logistic\_function(z):

return 1 / (1 + np.exp(-z))

# Prediction function

def predict(X, theta0, theta1):

z = theta0 + theta1 \* X

gz = logistic\_function(z)

return gz

# Cost function

def cost\_function(X, y\_true, theta0, theta1):

m = len(X)

epsilon = 1e-15

y\_pred = predict(X, theta0, theta1)

cost = -(1 / m) \* np.sum(y\_true \* np.log(y\_pred + epsilon) + (1 - y\_true) \* np.log(1 - y\_pred + epsilon))

return cost

# Gradient Descent

def gradient\_descent(X, y, theta0, theta1, learning\_rate):

m = len(X)

gradient0 = (1 / m) \* np.sum(predict(X, theta0, theta1) - y)

gradient1 = (1 / m) \* np.sum((predict(X, theta0, theta1) - y) \* X)

new\_theta0 = theta0 - learning\_rate \* gradient0

new\_theta1 = theta1 - learning\_rate \* gradient1

return new\_theta0, new\_theta1

theta0, theta1 = gradient\_descent(X, y, theta0, theta1, learning\_rate)

print(theta0)

print(theta1)