Assignment 7

Learning rate at 0.0001 at 100.000 runs gives (1 - error rate) of 92% for training and 91% for testing. Multiplication vector w initialized at [0.3, 0]. Parameters have been set after testing to minimize error rate.

Text

Description automatically generated

Graf of training output to y\_train, where y\_train is red and output is green:

Chart, line chart

Description automatically generated

Graf of testing output to y\_test, where y\_test is red and output is green:

A picture containing text, writing implement, pencil

Description automatically generated

Code:

import numpy as np

import matplotlib.pyplot as plt

import math

def func(X: np.ndarray):

    """

    The data generating function.

    Do not modify this function.

    """

    return 0.3 \* X[:, 0] + 0.6 \* X[:, 1] \*\* 2

def noisy\_func(X: np.ndarray, epsilon: float = 0.075):

    """

    Add Gaussian noise to the data generating function.

    Do not modify this function.

    """

    return func(X) + np.random.randn(len(X)) \* epsilon

def get\_data(n\_train: int, n\_test: int):

    """

    Generating training and test data for

    training and testing the neural network.

    Do not modify this function.

    """

    X\_train = np.random.rand(n\_train, 2) \* 2 - 1

    y\_train = noisy\_func(X\_train)

    X\_test = np.random.rand(n\_test, 2) \* 2 - 1

    y\_test = noisy\_func(X\_test)

    return X\_train, y\_train, X\_test, y\_test

def perceptron(w, x, b):

    return np.dot(w, x) + b

def sigmoid(x):

    return 1.0/(1.0 + np.exp(-x))

def sigmoid\_deriv(x):

    return sigmoid(x)\*(1-sigmoid(x))

if \_\_name\_\_ == '\_\_main\_\_':

    np.random.seed(0)

    X\_train, y\_train, X\_test, y\_test = get\_data(n\_train=280, n\_test=120)

    output = np.zeros(len(X\_train)) #outputfra modellen

    output2 = []

    runs = 100000 #Runs

    lr = 0.0001 #Learning-Rate

    # Teori = -2\*Sum( x[d][n]\*(t(d)-o(d))\*sigmoid\_deriv(w.T, x[d]))

    #def gradient\_decent(D, n, runs):

    w = np.array([0.3, 0], dtype='f')#array filled with some small random value #

    for j in range(runs):

        #print("w1:", w)

        for i in range(len(X\_train)):

            w\_d = [0, 0]

            output[i] = perceptron(w.T, X\_train[i], 0) #X\_train[i]\*w.T

            for k in range(len(w\_d)):

                #print("w2:", w, "add:", w[k] - lr\*(-X\_train[i][k]\*(y\_train[i]-output[i])), "k:", k)

                #w\_d[k] -= lr\*(-X\_train[i][k]\*(y\_train[i]-output[i])) #uansett ikke \*2 siden oppgaven spessifiserer E[vektor\_w]= 1/2 \* ...

                w[k] -= lr\*(-X\_train[i][k]\*(y\_train[i]-output[i])\*sigmoid\_deriv(perceptron(w.T, X\_train[i], 0)))

    print("train:", 1-round(np.square(np.subtract(y\_train,output)).mean(), 8), "w:", w, "count\*2:", count, count2)

    plt.plot(y\_train, color='r')

    plt.plot(output, color='g')

    plt.show()

    output2 = []

    for i in range(len(X\_test)):

        output2.append(X\_test[i][0]\*w[0] + X\_test[i][1]\*w[1])

    print("test:", 1-np.square(np.subtract(y\_test,output2)).mean())

    plt.plot(y\_test, color='r')

    plt.plot(output2, color='g')

    plt.show()