FALL 2022

CSE 4088

INTRODUCTION TO MACHINE LEARNING

Homework - 3

Prepared by

İsmail ÖKSÜZ - 150119516

MARMARA UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

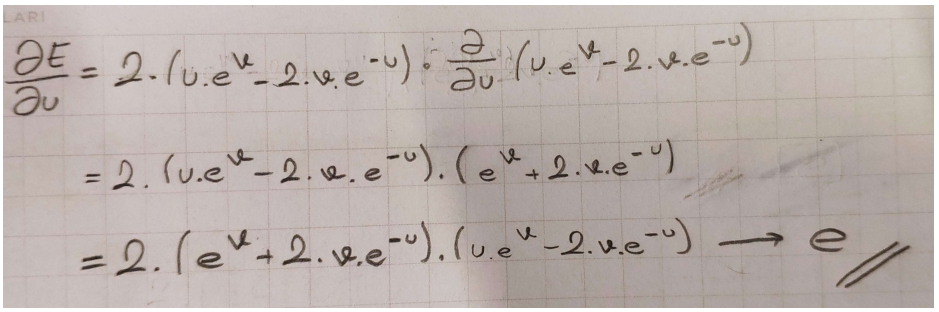
TABLE OF CONTENTS

Part 1 - Gradient Descent

1.1. Questions:

Consider the nonlinear error surface $E(u, v) = (ue^v - 2ve^{-u})^2$. We start at the point $(u, v) = (1, 1)$ and minimize this error using gradient descent in the uv space. Use $\eta = 0.1$ (learning rate, not step size).

1.1.1. What is the partial derivative of E(u, v) with respect to u?



Answer **[e]**

1.1.2. How many iterations (among the given choices) does it take for the error E(u, v) to fall below 10^{-14} for the first time? In your programs, make sure to use double precision to get the needed accuracy.

```
Question 1.2's answer: 10
exc@exc-NBLK-WAX9X:~/Desktop$
```

```python
#Finds the option for q5
def question1_2(ans):
    options = [1,3,5,10,17]
    dif = abs(ans-options[0])
    res = options[0]
    for i in range(len(options)):
        if abs(ans-options[i])<dif:
            dif = abs(ans-options[i])
            res = options[i]
    return res

```

Answer **[d]**

1.1.3. After running enough iterations such that the error has just dropped below 10^{-14}, what are the closest values (in Euclidean distance) among the following choices to the final (u, v) you got in Problem 1.2?

```
exc@exc-NBLK-WAX9X:~/Desktop$  cd /home/exc/D
esktop ; /usr/bin/env /bin/python3 /home/exc/
.vscode/extensions/ms-python.python-2022.18.2
/pythonFiles/lib/python/debugpy/adapter/../..
/debugpy/launcher 43553 -- /home/exc/Desktop/
hw3.py
Question 1.3's answer: (0.045),(0.024)
exc@exc-NBLK-WAX9X:~/Desktop$
```

```python
#Finds the option for q1.3
def question1_3(ans):
    rounded = round3float(ans)
    options = [
        [1.000, 1.000],
        [0.713, 0.045],
        [0.016, 0.112],
        [-0.083, 0.029],
        [0.045, 0.024]
    ]
    res = options[0]
    for i in range(5):
        if rounded==options[i]:
            res = options[i]
    return ("("+str(res[0])+"),("+str(res[1])+")")
```

Answer **[e]**

1.1.4. Now, we will compare the performance of “coordinate descent.” In each iteration, we have two steps along the 2 coordinates. Step 1 is to move only along the u coordinate to reduce the error (assume first-order approximation holds like in gradient descent), and step 2 is to reevaluate and move only along the v coordinate to reduce the error (again, assume first-order approximation holds). Use the same learning rate of η = 0.1 as we did in gradient descent. What will the error E(u, v) be closest to after 15 full iterations (30 steps)?

```
Question 1.4's answer: 0.1
exc@exc-NBLK-WAX9X:~/Desktop$
```

```python
#Finds the option for q1.4
def question1_4(ans):
    options = [10**-1, 10**-7, 10**-14, 10**-17, 10**-20]
    dif = abs(ans - options[0])
    res = options[0]
    for i in range(0,5):
        if (abs(ans - options[i]) < dif):
            dif = abs(ans - options[i])
            res = options[i]
    return res
```

Answer **[a]**

1.2. Description for Part 1

```
vals = np.array([1, 1])

#gets values for q1.2 and q1.3 with calling gradDescent method
coordinates, noiterations= gradDescent(fE, gradient, vals)

#gets values for q1.4 with calling crdDescent method
ansq7 = crdDescent(fE, derivative_fE_du, derivative_fE_dv, 1.0, 1.0)
answerq12 = question1_2(noiterations)
answerq13 = question1_3(coordinates)
answerq14 = question1_4(ansq7)
print("Question 1.2's answer: "+str(answerq12))
print("Question 1.3's answer: "+str(answerq13))
print("Question 1.4's answer: "+str(answerq14))
```

Figure 1. Screenshot of main in part1.py file

By calling gradDescent, answers for q1.2 and q1.3 occur

By calling crdDescent, answer for q1.4 occur

```
#Gradient Descent, answers for q1.2 and q1.3 come from this method
def gradDescent(fE, gradient, arr, tol=1e-14, n=0.1):
    noiterations = 0
    while fE(arr) > tol and noiterations <= 100:
        arr = arr - n*gradient(arr)
        noiterations += 1
    if noiterations > 100:

        #nothing will return
        return
    else:

        #returns no of iterations and coordinates here (answers of the q5 and q6)
        return (arr, noiterations)

#Coordinate Descent, answer for q1.4 come from this method
def crdDescent(fE, derivative_fE_du, derivative_fE_dv, uinit, vinit, tol=1e-14, n=0.1):
    v1 = uinit
    v2 = vinit
    v1arr = [v1]
    v2arr = [v2]
    for i in range(0,15):
        du = derivative_fE_du(v1, v2)
        v1 = v1 - n*du
        v1arr.append(v1)
        v2arr.append(v2)
        dv = derivative_fE_dv(v1, v2)
        v2 = v2 - n*dv
        v1arr.append(v1)
        v2arr.append(v2)
    return fE([v1, v2])
```

Figure 2. Screenshot of gradDescent and crdDescent methods in part1.py file

The gradDescent method takes parameters and finds coordinates and noiterations variables. Coordinates is the variable which is the answer of q1.3 and noiterations is the variable which is the answer of q1.2.

The crdDestent method takes parameters and finds the err variable. The err variable stands for the error after 15 iterations. It is the answer of q1.4.

Besides these, gradDescent method and crdDescent method uses other methods such as gradient, derivative_fE_dv, derivative_fE_du, fE to calculate the values that they return.

```
def fE(arr):
    v1 = arr[0]
    v2 = arr[1]
    expmv1 = np.exp(-v1)
    expv2 = np.exp(v2)
    pdrv = (v1*expv2 - 2*v2*expmv1)**2
    return pdrv

#Returns derivative respect to u
def derivative_fE_du(u, v):
    v1 = u
    v2 = v
    expmv1 = np.exp(-v1)
    expv2 = np.exp(v2)
    prdvroot = (v1*expv2 - 2*v2*expmv1)
    return 2*prdvroot*(expv2 + 2*v2*expmv1)

#Returns derivative respect to v
def derivative_fE_dv(u, v):
    v1 = u
    v2 = v
    expmv1 = np.exp(-v1)
    expv2 = np.exp(v2)
    prdvroot = (v1*expv2 - 2*v2*expmv1)
    return 2*prdvroot*(v1*expv2 - 2*expmv1)

#Calculates gradient
def gradient(arr):
    v1 = arr[0]
    v2 = arr[1]
    expmv1 = np.exp(-v1)
    expv2 = np.exp(v2)
    pdrvroot = (v1*expv2 - 2*v2*expmv1)
    ind1 = 2*pdrvroot*(expv2 + 2*v2*expmv1)
    ind2 = 2*pdrvroot*(v1*expv2 - 2*expmv1)
    arr = np.array([ind1,ind2])
    return arr
```

Figure 3. Screenshot of gradient, derivative_fE_du, derivative_fE_dv and fE methods in part1.py file

The derivative_fE_du returns the value of partial derivative with respect to u and derivative_fE_dv returns the value of partial derivative with respect to v.

```python
#Using for q1.3
def round3float(coordinates):
    v1 = coordinates[0]
    v2 = coordinates[1]
    nv1 = round(v1, 3)
    nv2 = round(v2, 3)
    ncoordinates = [nv1, nv2]
    return ncoordinates
```

Figure 4. Screenshot of the round3float method in part1.py file

This method returns a float number into 3 digits after 0. Using in q1.3 to find option.

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

#Finds the option for q1.2
def question1_2(ans):
    options = [1,3,5,10,17]
    dif = abs(ans-options[0])
    res = options[0]
    for i in range(len(options)):
        if abs(ans-options[i])<dif:
            dif = abs(ans-options[i])
            res = options[i]
    return res

#Finds the option for q1.3
def question1_3(ans):
    rounded = round3float(ans)
    options = [
        [1.000, 1.000],
        [0.713, 0.045],
        [0.016, 0.112],
        [-0.083, 0.029],
        [0.045, 0.024]
    ]
    res = options[0]
    for i in range(5):
        if rounded==options[i]:
            res = options[i]
    return ("("+str(res[0])+","+str(res[1])+")")

#Finds the option for q1.4
def question1_4(ans):
    options = [10**-1, 10**-7, 10**-14, 10**-17, 10**-20]
    dif = abs(ans - options[0])
    res = options[0]
    for i in range(0,5):
        if (abs(ans - options[i]) < dif):
            dif = abs(ans - options[i])
            res = options[i]
    return res
```

Figure 4. Screenshot of the methods to find option in part1.py file

Those methods find the correct answer for each question and print it (in figure 1, it represents).

Part 2 - Logistic Regression

2.1. Description for Part 2

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

#creates dataset
def createDataset(N):
  x1, y1 = 2*np.random.rand()-1, 2*np.random.rand()-1
  x2, y2 = 2*np.random.rand()-1, 2*np.random.rand()-1
  ydif = y2 - y1
  xdif = x2 - x1
  l1 = ydif/xdif
  l2 = y2 - l1*x2
  x = 2*np.random.rand(N, 3)-1
  x[:,0] = 1
  y = np.sign(l2 + l1*x[:,1] - x[:,2])
  dsk = np.where(y==-1)
  yksk = np.where(y==1)
  plt.scatter(x[dsk,1], x[dsk,2])
  plt.scatter(x[yksk,1], x[yksk,2])
  plt.show()
  return x,y

#tried to hold values for epoch and Eout into an array
epoch = []
Eout = []
for i in range(1,100):
  x, y = createDataset(100)
print(np.average(epoch))
print(np.average(Eout))
```

Figure 5. Screenshot of the part2.py file

I can only create a dataset with an input number of N. Into the createDataset method, it generates random points, then with using random points, it generates random lines. The dataset can be created by using these random generated lines.
Via using matplotllib framework, I can visualise the datasets.
Additionally, i tried to hold values of epoch and Eout's into arrays called epoch and Eout. However I could not complete it.
Part2.py file creates a dataset with the given input parameter N. It generates the points and the lines randomly via using numpy.random.rand.

2.2. Some of the examples of the datasets that were created by createDataset method

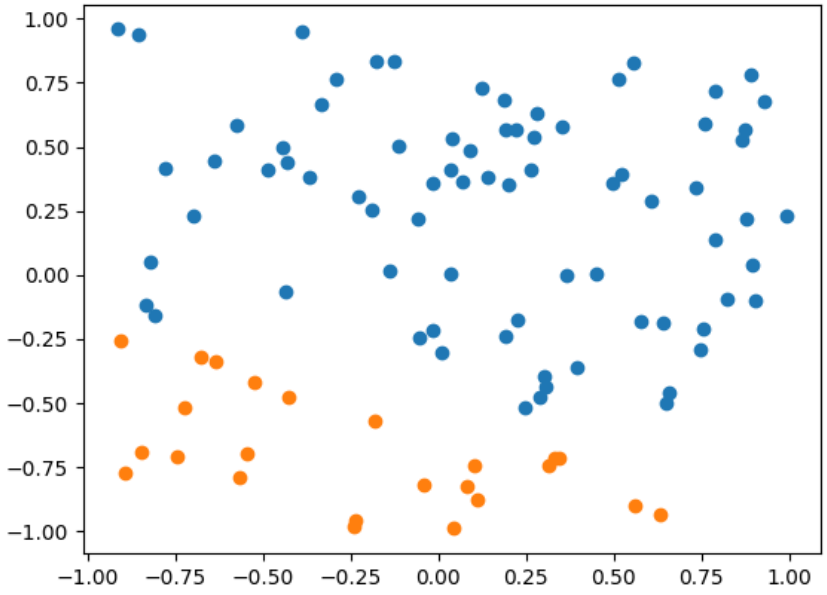


Figure 6. A dataset created by using createDataset method. N=100.

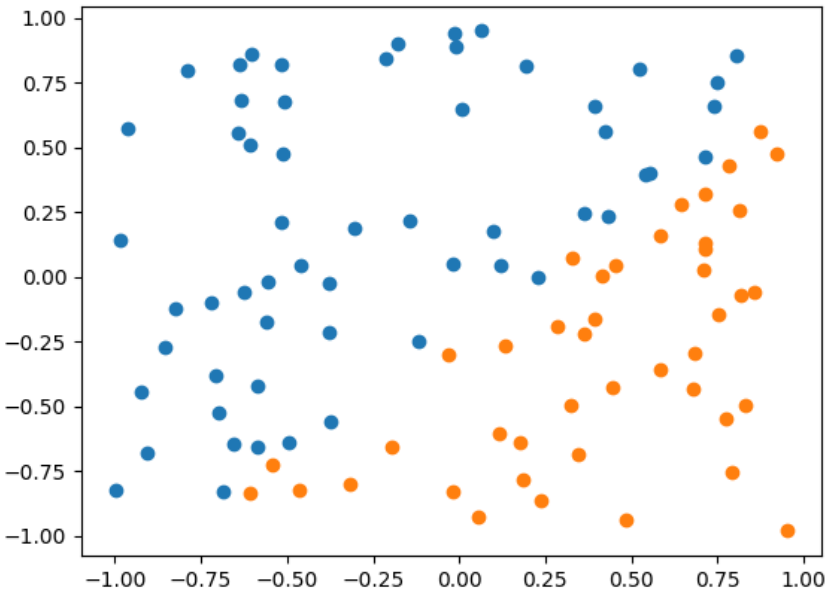


Figure 7. A dataset created by using createDataset method. N=100.

Part 3 - Regularization with weight decay

```python
import numpy as np
import requests

#It requests to download in.dta from website
try:
    with open("in.dta", "x") as input:
        req= requests.get("http://work.caltech.edu/data/in.dta")
        input.write(req.text)
        print("Successfully downloaded.")
except FileExistsError as fee:
    print("It is downloaded before")

#It requests to download out.dta from website
try:
    with open("out.dta", "x") as output:
        req2 = requests.get("http://work.caltech.edu/data/out.dta")
        output.write(req2.text)
        print("Successfully downloaded.")
except FileExistsError as e:
    print("It is downloaded before")

#It loads data
input = np.loadtxt("in.dta")
trainer1, trainer2 = input[:,:2], input[:,2]
output = np.loadtxt("out.dta")
tester1, tester2 = output[:,:2], output[:,2]
```

Figure 8. Screenshot of the part3.py file

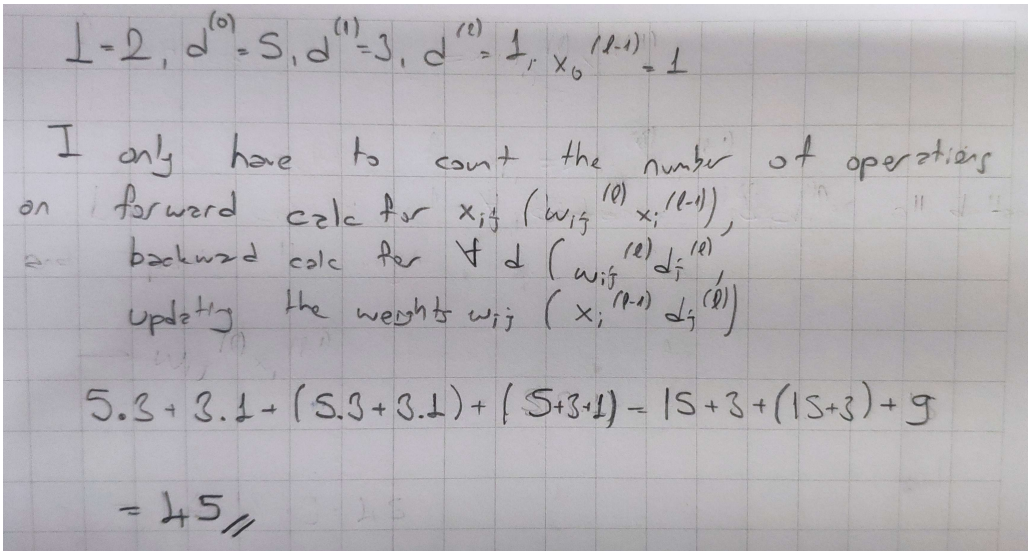
Like part 2, I could not finish that part either. I can only request to download and load data from website using try except blocks.

```
.py
It is downloaded before
It is downloaded before
exc@exc-NBLK-WAX9X:~/Desktop/Fi
s/CSE_7/CSE4088_-_Machine_Learn
```

Figure 9. Screenshot of the output when the part3.py file executes

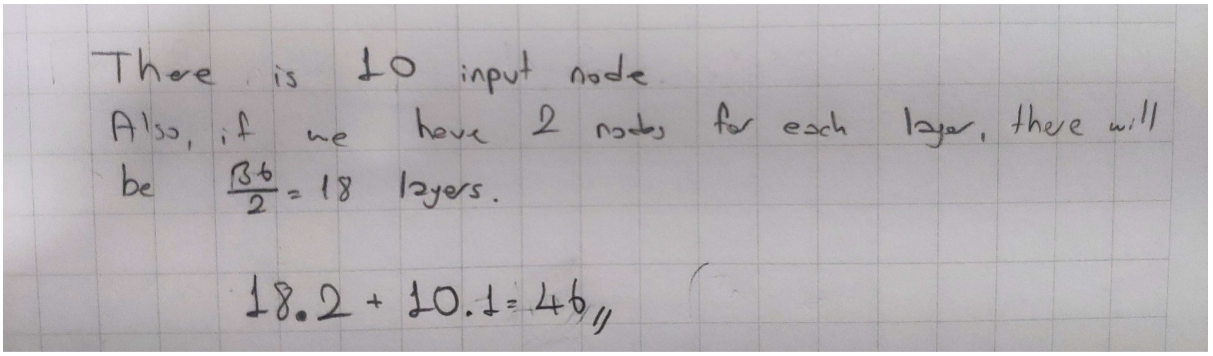
Part 4 - Neural Networks

4.1. Let us call every 'node' in a Neural Network a unit, whether that unit is an input variable or a neuron in one of the layers. Consider a Neural Network that has 10 input units (the constant $x_0^{(0)}$ is counted here as a unit), one output unit, and 36 hidden units (each $x_0^{(l)}$ is also counted as a unit). The hidden units can be arranged in any number of layers $l = 1, \cdots, L-1$, and each layer is fully connected to the layer above it.



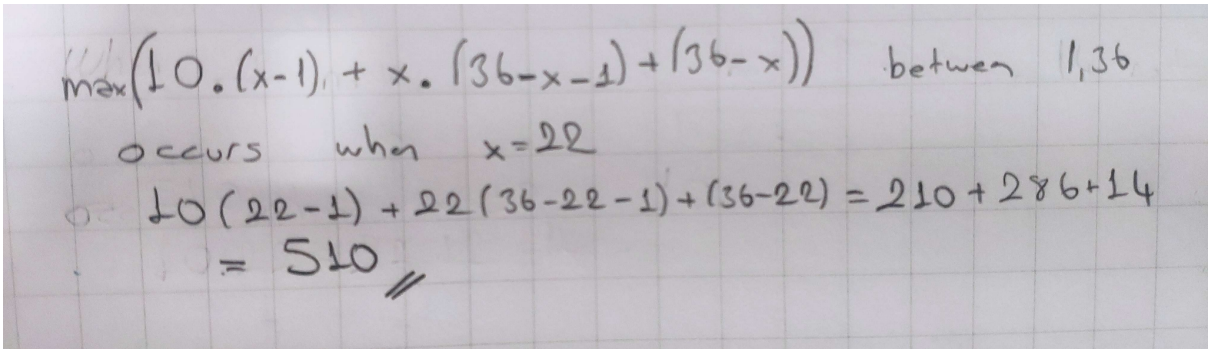
Answer **[d]**

4.2. What is the minimum possible number of weights that such a network can have?



Answer **[a]**

4.3. What is the maximum possible number of weights that such a network can have?



Answer **[e]**