Gökcan Kahraman
21702271

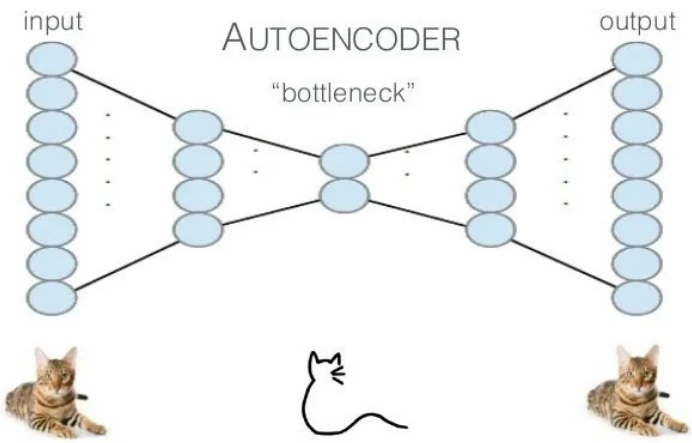
Neural Network Interim Project

Question 1:

In the first question of our Neural Network assignment, we need to build an autoencoder. The concept we need to use in this neural network algorithm is single hidden layer. The reason why we use this is to extract the functions. In order to do this algorithm, we had to convert a colored image to greyscale using the luminosity formula. Then, in order to train this model, the calculation of components such as MSE loss should be included. Then we used Tykhonov regularization to deal with the overfitting problem. Finally, we adjust the sparsity of the parameters using the KL divergence concept.

$$J_{ae} = \frac{1}{2N}\sum_{i=1}^{N}\|d(m) - o(m)\|^2 + \frac{\lambda}{2}\left[\sum_{b=1}^{L_{hid}}\sum_{a=1}^{L_{in}}\left(W_{a,b}^{(1)}\right)^2 + \sum_{c=1}^{L_{out}}\sum_{b=1}^{L_{hid}}\left(W_{b,c}^{(2)}\right)^2\right] + \beta\sum_{b=1}^{L_{hid}} KL(\rho|\hat{\rho}_b)$$

Figure 1: Our aeLost function to minimize in question 1



Gökcan Kahraman
21702271

Figure 2: Simple representation of autoencoder

Part 1.a

In part a of the first question, what we needed to do was to extract the 16x16 RGB image in the data in the appropriate format. Then we had to load the luminosity model given in the project question to convert the image to gray. (Y = 0.2126 * R + 0.7152 * G + 0.0722 * B) Then we need to take the standard derivation of this graph with 3 differences and throw it to std with 3 differences to protect it from saturation. Finally, we had to extract 200 different samples by trimming the data between 0.1 and 0.9.

First, we need to understand how many different samples are in our data in order to extract it properly. When we look at the data, we see that there are 10240 samples. It was already given in the question that it was 16x16 in size. Since our image is a color image, there should also be a "3" in the definition of the data, indicating red, green and blue. We used "h5py" to properly translate the data here and when calling our code in the last part, we used our image as png appropriately.

Gökcan Kahraman
21702271

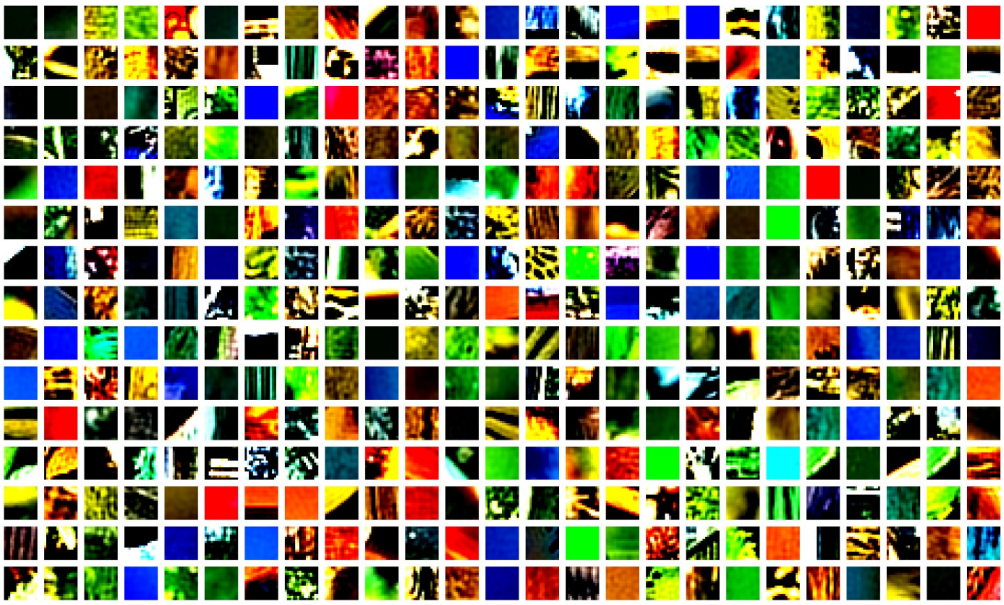


Figure 3: RGB data which sampled on data1.h5

Then, using the luminosity model, we converted our image to gray, just as given in the first question. As I will show in the fourth figure, we needed to convert our image to gray. To do this, we first had to delete the three channels at the end of our image. This way there would be no red, green and blue in the image. Then we applied the standard deviation and std of our data with a difference of 3. Finally, our data was organized into the matrix (10240,256) with the "reshape" command.

Gökcan Kahraman
21702271

Figure 4: Greyscaled data which sampled on data1.h5

As a result, we have extracted our images in gray and color, in accordance with the first question. When we examine our 200 different examples, one of the biggest differences between the images is of course the difference in color scale. The next biggest difference is that gray images have a higher image quality than color images. The reason for this is that we apply a standard deviation with 3 differences in our gray images. Applying standard deviation within images brings the pixels between images closer together. Because of this, images with standard deviation are of higher quality.

Part 1.b

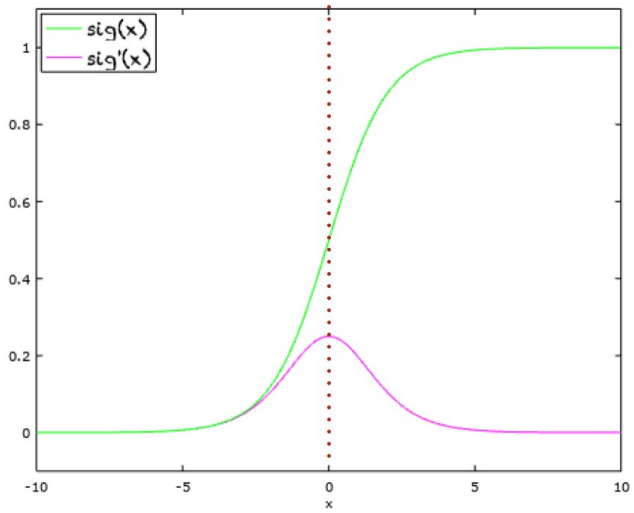
In part b of our question, we needed to calculate the parameters needed for training. We were given specific parts in the question to make these calculations. In the question, we were given the intervals of uniform random numbers [-w0 w0], the value of w0. We were asked to write a function called "aeCost" to calculate this cost function. Thanks to this function, we will find the weights appropriately and find the most appropriate number range in which the elements required

Gökcan Kahraman
21702271

for the parameters should be written. In the question, our Lhid value was given as 64 and our lambda value was given as 5e-4.

First, I had to define the elements in our function. First I wrote the code for my weight function and the parameter matching of my elementr's. Then I wrote the code for linear neurons and hidden layers. In the forward pass part we use the sigmoid function. The reason for this is that the sigmoid activation is logistic and we can extract the weights we want in the way we want. Finally, we use Tikhonov regularization formula because it prevents overfitting. For sparcity tuning we write the equation "KulLeiDiv".

Figure 5:



Plot of σ(x) and its derivate σ'(x)

Domain: (−∞, +∞)
Range: (0, +1)
σ(0) = 0.5

Other properties

$$\sigma(x) = 1 - \sigma(-x)$$

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

Sigmoid Equation

As a result of the work I did on my code, I gave my learning rate value 0.1, batch value 16, rho value 0.05, alpha value 0.85 and epoch value 100. As a result, after running my code, I found the error rate value 1. In fact, under certain conditions, our error rate value can be lower, but due to my explanation in part c, I got the error rate in this way.

Total Epoch	1	2	3	4	10	20	30	40	100
Error Rate	1.75	1.34	1.21	1.15	1.05	1.02	1.01	1	1

Gökcan Kahraman
21702271

Figure 6: Part B error rates

Part 1.c

I had to write solver code in part c. After writing this code, the first layers of conenction weights would appear. We needed to do this differently for all the images and we needed to understand how the view of the images was. First of all, I chose to use the stochastic gradient descent algorithm to write this part. I thought it should be used both because we will use it in question 2 and because it will have a shorter convergence time compared to other algorithms. I set my batch-size to 16 in the code and in order to extract the image I wanted, I mixed all the epochs in the algorithm and examined a single epoch (10240 16) layer in the weights.

Gökcan Kahraman
21702271

Figure 7: Part C result graphic

When we examine these images, as the layer size decreases, we need to extract a feature accordingly so that our error decreases more. However, as a result of this change, the complexity of the images we extract decreases and we get a worse image. If our error function remains constant when our lambda values change, the "w" functions it outputs to us will be the same and the images we extract will be the same.

Part 1.d

In our last part, we were asked to examine the difference of hidden layer features with three different Lhid and lambda values. To examine this, I gave Lhid values of 10, 50 and 100 and lambda values of 0, 10-1 and 10-2.

Gökcan Kahraman
21702271

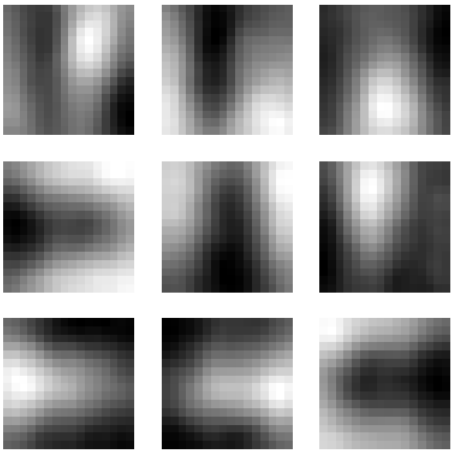
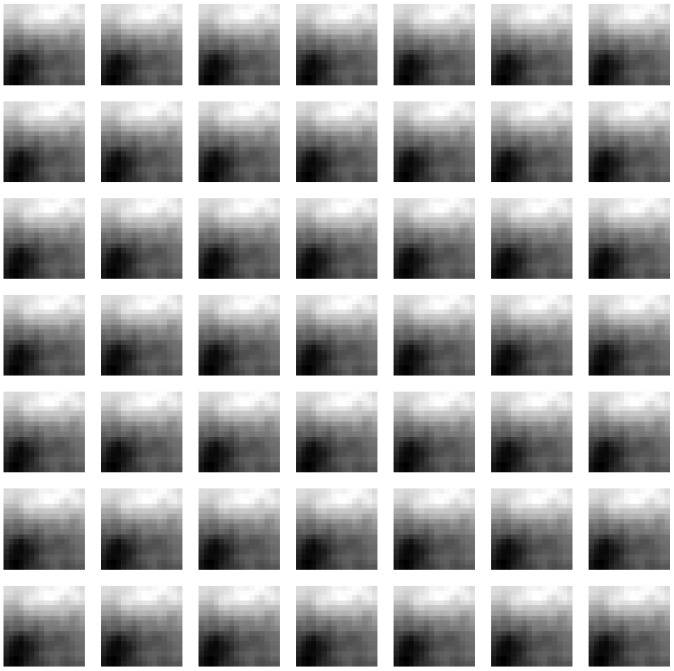
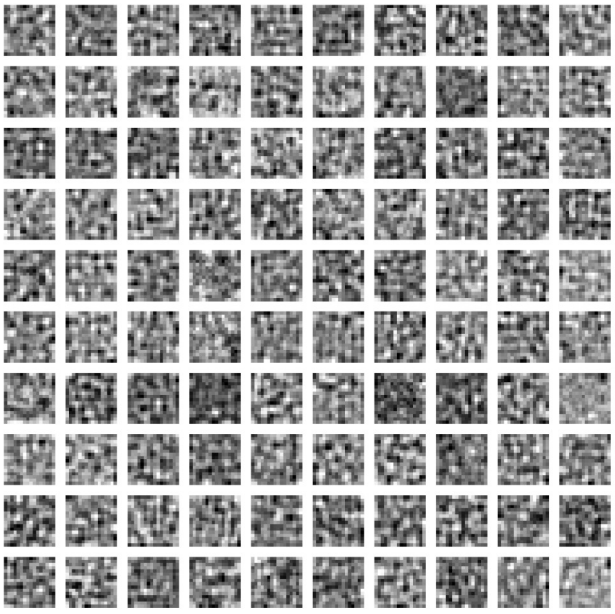


Figure 8: Image with lambda: 0.001 and Lhid: 10 (error is 1.28)



Gökcan Kahraman
21702271

Figure 9: Image with lambda: 0.1 and Lhid: 50 (error is 1.98)



Gökcan Kahraman
21702271

Figure 10: Image with lambda: 0.001 and Lhid: 100 (error is 0.24)

As a result, when we examine the values, when we keep our values low, our image comes out very poor quality. Since it does not work enough, the error rate is also low. When we keep our values in the middle value, it always starts to select our images from the same weight and since it is constantly returning it, our error rate is fixed very early and fixed high. When we set our values high, we get an error rate lower than the median value.

Question 2:

In the second question of our project, we need to write an algorithm on NLP (Natural Language Process), one of the branches of neural networks. What we need to do in the question is to get the input, output, validation and testing files through the data2 file. We need to construct a sentence by generating seeds from 250 words over this database. Then we need to give 3 words from this sentence and choose the ten most likely words of the fourth word. There are certain concepts given to help with the question. First, we need to make the words embedded by embedding the indexed words. Then, after sending our words in the form of logistic hidden units into the hidden layer, we need to estimate the fourth-word using softmax. What we need to do in the "A" part of the question is we need to understand the way our code works based on the different values in D and P. We have a system with 200 samples, a 0.15 learning rate, a 0.85 momentum rate and a maximum of 50 epochs. We also have a Gaussian standard deviation (0.1). Finally, when our code reaches a certain point, it should give a cross-entropy error and go to the next step. In the "B" part of our question, we were asked to perform the algorithm I explained at the beginning and compare the results.

Gökcan Kahraman
21702271

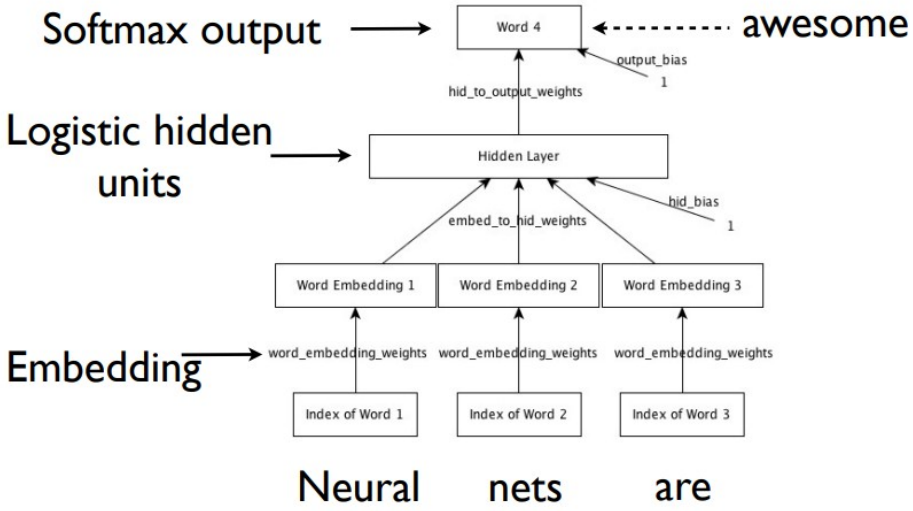


Figure 11: Explanation about question 2

Algorithm

init:

In the function we call "init", we initialize its name. This way we will be able to run other algorithms. In this program, where we load the parameters, we first give the weights and seed values. Seed allows us to generate random sentences from the data. Then I wrote the weight and bias functions in the function that we will initialize. Within the function, the mean was 0, derivation was 0.1. Next, we made our weight-embedded matrix. This matrix is 1x3 and its elements are all the same. Our "b" value in the equation is zero. The biggest reason for this is that the values in our columns stay the same. Next, we name the parameters requested from the trace in the question, softmax, sigmoid tuple, and momentum parameters.

train:

Gökcan Kahraman
21702271

On that function, I wanted to make training function for my algorithm learning how to choose fourth word. Firstly, I defined loss and accuracy lists to get my results in those lists and show what is my error and accuracy on every epoch. Then, I called encoding function for each X and Y variables (I am going to explain function of encoding later). Therefore, I started to build by training loop function. Firstly, my momentum value could interrupt epochs and I needed to preserve my epoch values. Therefore, on training loop, I made momentum zero. Then, I added mini batches interval to get requirements in assignment. After I applied my mini batches, I need to write a algorithm when training is over. Firstly, I needed to choose shuffled data which is in mini batches. Then, I need to update my gradient descent based on results of epoch. Also, I updated my momentum based on gradient descent update rule. Finally, we predicted our result and print our result for each epoch. Finally, I needed to write validation cross entropy code. Based on my algorithm, when my value loss function is smaller than 3.35 and our difference between last five results average is smaller than 0.01, there is warning which training is stopped.

encodeOne:

This function makes encoding operation. The reason we do the Encode operation is to perform the next prediction. The data we find with forward propagation is sometimes not orthogonal. We can't even start prediction and train functions if they are not orthogonal. To do this, we encode and set our data to the appropriate value.

activation:

This function makes softmax and sigmoid operations based on our requirements on question 2.

trainLoss:

For that function, we need to make forward and propagation. Because our code will epoch fifty times and we need to calculate the train loss from the data we obtained according to the previous epoch. With forward propagation, we send the input value to the output, and with

backward propagation, we send the output value to the input value. As in the previous functions, we first define our parameters. Then we will do forward and backward propagation. In forward propagation, we calculate our variables and their derivatives and calculate the loss and gradient. However, in these functions, we must use the "delta" parameter to transfer the data from the output to the input. If we don't use it, we can't do backward propagation.

pred:

For predicting our fourth word, we need to write prediction function. That function simply makes, firstly getting three different variables, then we got each of variables dot product with variables and add their base function. When we make forward pass, we get our label. After that, we have a different function, we applied our operation I gave in first sentence. After get that labels max value, when our forward propagated value is equal to other variable, our prediction is correct based on classification.

Results

2.a

For when our D and P functions are 8 and 64, our results are:
Train Loss: 3.343612
Value Loss: 3.349868
Train Accuracy: 27.154631
Value Accuracy: 26.806452
Test Accuracy: 27.103225806451615
Ended at 38. Epoch

Gökcan Kahraman
21702271

For when our D and P functions are 16 and 128, our results are:

Train Loss: 3.149707

Value Loss: 3.168190

Train Accuracy: 29.528322

Value Accuracy: 29.397849

Test Accuracy: 29.524731182795698

Ended at 48. Epoch

For when our D and P funtions are 32 and 256

Train Loss: 3.127045

Value Loss: 3.138189

Train Accuracy: 29.983624

Value Accuracy: 29.772043

Test Accuracy: 29.7247311827957

Ended at 50. Epoch

As a result of these, when we increase our D and P variables, our train loss and accuracies difference becomes small earlier than small D and P variables. Therefore, the main result I wrote train loss condition for validation cross entropy is this. If I only coded limit interval for train losses, highest D and P function ended earlier than others. Thus, the higher the D and P values, the shorter the time we approach the optimum difference.

2.b

Sentence	**Label**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
what we do	well	do	want	.	have	think	i	?	get	we	you
think about	.	.	?	it	,	the	that	you	this	i	a

Gökcan Kahraman
21702271

that in my work	,	life	.	way	game	own	home	day	,	place	country
i did nt	want	know	have	want	think	do	get	see	even	like	.
will you do	then	it	?	.	that	the	this	to	,	them	what

As a result for part b, this is my sentences and label orders. When we make a comparison, we see that it finds three out of five lines, or even finds it best in one. The software is very good at finding punctuation marks. He is also good at putting verbs in their place. However, whenever sentences are possible, my algorithm may not yield very good results.

Appendices:

Question 1:

```
#Question 1
# That part helps me to importing
import h5py
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sn
import sys
class Q1Answer(object):
# For question 1, we need to create autoencoder. Therefore, on that class, I worked to create autoencoder.
    def spesification(self, inL, hidL):
    # To determine our autoencoder, we need to assign our value spesification.
        Lout = inL
        baseW = np.sqrt(6 / (inL + hidL))
```

```
        W1 = np.random.uniform(-baseW, baseW, size=(inL, hidL))
        b1 = np.random.uniform(-baseW, baseW, size=(1, hidL))
        W2 = np.random.uniform(-baseW, baseW, size=(hidL, Lout))
        b2 = np.random.uniform(-baseW, baseW, size=(1, Lout))
        We = (W1, W2, b1, b2)
        return We

    def dataTraining(self, data, pmeters,totalBatch, eta=0.1, alpha=0.9, epoch=10):
        # As we can understand, this is our one of the main function which trains our data.
        listJ = []
        inL = pmeters["inL"]
        hidL = pmeters["hidL"]
        We = self.spesification(inL,  hidL)
        mWe = (0, 0, 0, 0)

        itenum = int(data.shape[0] / totalBatch)

        for i in range(epoch):
            totJ = 0
            beg = 0
            end = totalBatch
            perm = np.random.permutation(data.shape[0])
            data = data[perm]
            mWe = (0, 0, 0, 0)

            for j in range(itenum):
                batchData = data[beg:end]
                J, Jgrad, cache = self.aeCost(We, batchData, pmeters)
                We, mWe = self.uptWeight(Jgrad, cache, We, mWe, eta, alpha)
                totJ = J + totJ
                beg = end
                end = totalBatch + end
```

```
            totJ = totJ/itenum
            print("Our loss is: {:.2f} [Total Epoch: {} Current Epoch: {}]".format(totJ, i+1, epoch))
            listJ.append(totJ)

        print("\n")
        return We, listJ


    def aeCost(self, We, data, pmeters):
        # This definition defines our gradients' first error. Then, I did forward pass thanks to detection of gradients.
        listData = data.shape[0]
        W1, W2, b1, b2 = We
        p = pmeters["p"]
        beta = pmeters["beta"]
        lambd = pmeters["lambd"]
        inL = pmeters["inL"]
        hidL = pmeters["hidL"]

        #That side operates forward pass.
        lin1 = data @ W1 + b1
        hid = 1 / (1 + np.exp(-lin1))
        lin2 = hid @ W2 + b2
        hidd = 1 / (1 + np.exp(-lin2))
        # That part operates sigmoid variables.
        derivHid = hid * (1 - hid)
        derivHidd = hidd * (1 - hidd)

        base_p = hid.mean(axis=0, keepdims=True)

        totalLoss = 0.5/listData * (np.linalg.norm(data - hidd, axis=1) ** 2).sum()
        Tyk = 0.5 * lambd * (np.sum(W1 ** 2) + np.sum(W2 ** 2)) #Tyk equation is to apply regularization and cope overfitting
```

Gökcan Kahraman
21702271

```
        KulLeiDiv = p * np.log(p/base_p) + (1 - p) * np.log((1 - p)/(1 - base_p)) #KulLeiDiv makes
sparcity tuning
        KulLeiDiv = beta * KulLeiDiv.sum()


        backPropJ = totalLoss + Tyk + KulLeiDiv


        derivTotalLoss = -(data - hidd)/listData
        derivTykk = lambd * W2
        derivTyk = lambd * W1
        derivKulLeiDiv = beta * (- p/base_p + (1-p)/(1 - base_p))/listData

        cache = (data, hid, derivHid, derivHidd)
        gradbackPropJ = (derivTotalLoss, derivTykk, derivTyk, derivKulLeiDiv)


        return backPropJ, gradbackPropJ, cache


    def uptWeight(self, Jgrad, cache, We, We_m, LearRate, alpha):

        W1, W2, b1, b2 = We
        derivdW1 = 0
        derivdW2 = 0
        derivdB1 = 0
        derivdB2 = 0

        data, hid, derivHid, derivHidd = cache
        totalLoss, derivTykk, derivTyk, derivKulLei = Jgrad
        delta = totalLoss * derivHidd
        derivdW2 = hid.T @ delta + derivTykk
        derivdB2 = delta.sum(axis=0, keepdims=True)
        delta = derivHid * (delta @ W2.T + derivKulLei)
        derivdW1 = data.T @ delta + derivTyk
        derivdB1 = delta.sum(axis=0, keepdims=True)
```

Gökcan Kahraman
21702271

```
        derivWe = (derivdW1, derivdW2, derivdB1, derivdB2)


# We need to update our weights based on momentum
        W1, W2, b1, b2 = We
        derivdW1, derivdW2, derivdB1, derivdB2 = derivWe
        W1_m, W2_m, B1_m, B2_m = We_m

        W1_m = LearRate * derivdW1 + alpha * W1_m
        W2_m = LearRate * derivdW2 + alpha * W2_m
        B1_m = LearRate * derivdB1 + alpha * B1_m
        B2_m = LearRate * derivdB2 + alpha * B2_m

        W1 -= W1_m
        W2 -= W2_m
        b1 -= B1_m
        b2 -= B2_m

        We = (W1, W2, b1, b2)
        We_m = (W1_m, W2_m, B1_m, B2_m)

        return We, We_m


    def predict(self, data, We):
        W1, W2, b1, b2 = We

        lin1 = data @ W1 + b1
        hid = 1 / (1 + np.exp(-lin1))
        lin2 = hid @ W2 + b2
        hidd = 1 / (1 + np.exp(-lin2))
        return hidd

# This is our last file to call data and run our code.
```

Gökcan Kahraman
21702271

```
dataFile = h5py.File("data1.h5", 'r')


dataList = list(dataFile.keys())
derivT = dataFile.get('data')
derivT = np.array(derivT)


dataZeros = np.zeros((10240,16,16))
for i in range(10240):
    img = derivT[i]
    img = img.transpose((1, 2, 0))
    R, G, B = img[:,:,0], img[:,:,1], img[:,:,2]
    dataZeros[i] = 0.2126 * R + 0.7152 * G + 0.0722 * B


# This part is for normalization.
dataZeros = np.reshape(dataZeros, (10240, 256))  # We transform our data to 2 dimensions which is 10240 and 256.
dataZeros = dataZeros - dataZeros.mean(axis=1, keepdims=True)  # We wanted to differentiate image.
dataStandart = np.std(dataZeros)  # We found standart of data.
dataZeros = np.clip(dataZeros, - 3 * dataStandart, 3 * dataStandart)  # We made our standart interval by increasing and decreasing by 3
dataZeros = (dataZeros - dataZeros.min())/(dataZeros.max() - dataZeros.min()) # We apply normalization
dataZeros = 0.1 + dataZeros * 0.8  # We map our data between 0.1 to 0.9
trainData = dataZeros
dataZeros = np.reshape(dataZeros, (10240, 16, 16))  # We reshape our data to check results better.
dataa = derivT.transpose((0, 2, 3, 1))
dataa = np.clip(dataa,0,1)  # Clipping original data
figure, axis = plt.subplots(15, 25, figsize=(25, 15))
figuree, axiss = plt.subplots(15, 25, figsize=(25, 15))


for i in range(15):
    for j in range(25):
```

Gökcan Kahraman
21702271

```
        k = np.random.randint(0, dataa.shape[0])
        axis[i, j].imshow(dataa[k].astype('float'))
        axis[i, j].axis("off")
        axiss[i, j].imshow(dataZeros[k], cmap='gray')
        axiss[i, j].axis("off")
figuree.savefig("grayful.png")
figure.savefig("colorful.png")

lambd = 0
LearRate = 0.1
totalBatch = 16
p = 0.05
alpha = 0.85
epoch = 100
beta = 0.1
inL = trainData.shape[1]
hidL = 100

pmeters = {"p": p, "beta": beta, "lambd": lambd, "inL": inL, "hidL": hidL}
func = Q1Answer()
w = func.dataTraining(trainData, pmeters, totalBatch, LearRate, alpha, epoch)[0]
W = ((w[0] - w[0].min())/(w[0].max() - w[0].min())).T
W = W.reshape((W.shape[0], 16, 16))
name = "result"
W_Dimension = int(np.sqrt(W.shape[0]))

# Our plot function
figure, axis = plt.subplots(W_Dimension,W_Dimension, figsize=(W_Dimension, W_Dimension),
dpi=160)
k = 0
for i in range(W_Dimension):
    for j in range(W_Dimension):
        axis[i, j].imshow(W[k], cmap='gray')
```

Gökcan Kahraman
21702271

```
        axis[i, j].axis("off")
        k += 1


figure.savefig(name + ".png")
```

Question 2:

```
import h5py
import numpy as np
import seaborn as sn
import sys


class Q2Answer(object):
    def init(self, tot, totalLayer = 3, stanDeriv = 0.01):
        # On that function, I initialize parameters on that function.
        randSeed = 1907  # This random seed helps us to create random sentences from data
        W = []
        b = []

        # On that for loop, I aimed to initialize weights and biases. As it was given on interim project,
        # our mean is 0 and our derivation is 0.01.
        for i in range(totalLayer):

            # On our weight embedded matrix, it is 1x3 and all of its elements are same.
            # Since I made "b" zero, our updated matrix's column elements will reamain same.
            if  i == 0:
                np.random.seed(randSeed)
                col = np.random.normal(0, stanDeriv, size=(int(tot[i]/3), tot[i + 1]))
                W.append(np.vstack((col, col, col)))
                assert W[i].shape == (tot[i], tot[i + 1])
                np.random.seed(randSeed)
                b.append(np.zeros((1, tot[i + 1])))
                continue
```

```
            np.random.seed(randSeed)
            W.append(np.random.normal(0, stanDeriv, size=(tot[i], tot[i + 1])))
            np.random.seed(randSeed)
            b.append(np.random.normal(0, stanDeriv, size=(1, tot[i + 1])))
            assert W[i].shape == (tot[i], tot[i + 1])
            assert b[i].shape == (1, tot[i + 1])

        # This side is for loading class parameters.
        fifa = {"W": [None] * totalLayer, "b": [None] * totalLayer} # FIFA is one of the sports game
which uses momentum a lot.
        tot = tot
        pmeters = {"W": W, "b": b}
        neur = ("sigmoid", "sigmoid", "softmax")

        return fifa,tot,pmeters,neur,randSeed

    def train(self, inp, lab, valInp, valLab, tot, ratLearn = 0.5, epoch = 100, totalBatch = 100, alpha =
0):

        fifa,tot,pmeters,neur,seed = self.init(tot,3,0.01)

        totalLayer = 3
        listTRloss = []
        listVAloss = []
        listTRacc = []
        listVAacc = []

        epochIter = int(inp.shape[0] / totalBatch)

        decodedLab = lab
        inp = self.encodeOne(inp)
        lab = self.encodeOne(lab)
        valInp = self.encodeOne(valInp)
```

```
        encodedLabVal = self.encodeOne(valLab)

        # After that loop, our training will be done.
        for i in range(epoch):
            np.random.seed(seed)
            perm = np.random.permutation(inp.shape[0])



            inp = inp[perm]
            lab = lab[perm]
            decodedLab = decodedLab[perm]

            # To preserve our epoch, we need to make our momentum zero
            for l in range(totalLayer):
                fifa["W"][l] = np.zeros((tot[l], tot[l+1]))
                fifa["b"][l] = np.zeros((1, tot[l+1]))

            # This part for our mini batches interval
            beg = 0
            end = totalBatch
            lossTrain = 0

            # When our mini batch operation is done, this is the code which training is over
            for j in range(epochIter):

                # We choose on that shuffled data which is mini batch.
                smallInp = inp[beg:end]
                smallLab = lab[beg:end]



                SmalldecodedLab = decodedLab[beg:end]

                ls, gradients = self.trainLoss(smallInp, smallLab,pmeters,neur)
```

```
            lossTrain += ls

            # On that loop, gradient descents done for its update
            for k in range(totalLayer):

                if k == 0:
                    fifa["W"][k] = ratLearn * gradients["W"][k] + alpha * fifa["W"][k]
                    derivCol1, derivCol2, derivCol3 = np.array_split(fifa["W"][k], 3, axis=0)
                    derivColAvg = (derivCol1 + derivCol2 + derivCol3)/3
                    pmeters["W"][k] -= np.vstack((derivColAvg, derivColAvg, derivColAvg))
                    assert pmeters["W"][k].shape == (tot[0], tot[1])
                    continue

                fifa["W"][k] = ratLearn * gradients["W"][k] + alpha * fifa["W"][k]
                fifa["b"][k] = ratLearn * gradients["b"][k] + alpha * fifa["b"][k]
                pmeters["W"][k] -= fifa["W"][k]
                pmeters["b"][k] -= fifa["b"][k]

            beg = end
            end += totalBatch

        predictio = self.pred(inp,pmeters,neur)
        assert predictio.shape == decodedLab.shape
        accTrain = (predictio == decodedLab).mean() * 100

        predictio = self.pred(valInp,pmeters,neur)
        assert predictio.shape == valLab.shape
        accValue = (predictio == valLab).mean() * 100
        predictVal = self.pred(valInp,pmeters,neur, False)
        assert encodedLabVal.shape == predictVal.shape
        lossValue = np.sum(- encodedLabVal * np.log(predictVal)) / encodedLabVal.shape[0]  #
This is cross entropy loss
```

```
        print('\r(D, P) = (%d, %d) Train Loss: %f, Value Loss: %f, Train Acc: %f, Value Acc: %f
[Current Epoch: %d Total Epoch: %d].'
        % (tot[1], tot[2], lossTrain/(j+1), lossValue, accTrain, accValue, i + 1, epoch), end='')

        listTRloss.append(lossTrain/epochIter)
        listVAloss.append(lossValue)
        listTRacc.append(accTrain)
        listVAacc.append(accValue)


        if i > 5 :
            convo = listVAloss[-5:]
            convo = sum(convo) / len(convo)

            lmt = 0.01
            # This place is for making a warning of cross entropy difference
            if (convo - lmt) < lossValue < (convo + lmt) and lossValue < 3.35:
                print(" ! Warning ! Training stopped ! Validation Cross Entropy difference with
convergence is around 0.01 "
                      "(Last points' average is +- 0.01).\n")
                return {"listTRloss": listTRloss, "listVAloss": listVAloss,
                        "listTRacc": listTRacc, "listVAacc": listVAacc, "pmeters": pmeters, "neur": neur, }


    return {"listTRloss": listTRloss, "listVAloss": listVAloss,
            "listTRacc": listTRacc, "listVAacc": listVAacc,"pmeters":pmeters,"neur":neur}

def trainLoss(self, inp, lab, pmeters,neur):
    totalLayer = 3
    W = pmeters["W"]
    b = pmeters["b"]
    neur = neur
    listOut = [inp]
    deriv = [1]
```

```
        totalBatch = inp.shape[0]

        # This code makes forward propagation.
        variab = inp @ W[0] + b[0]
        variab1, deriv1 = self.activation(neur[0], variab)
        variab2 = variab1 @ W[1] + b[1]
        variab21, deriv21 = self.activation(neur[1], variab2)
        variab3 = variab21 @ W[2] + b[2]
        variab32, deriv32 = self.activation(neur[2], variab3)
        listOut.append(variab1)
        listOut.append(variab21)
        listOut.append(variab32)
        deriv.append(deriv1)
        deriv.append(deriv21)
        deriv.append(deriv32)

        # This code makes loss calculation
        predictio = listOut[-1]
        ls = np.sum(-lab * np.log(predictio)) / lab.shape[0]  # This code is here for to calculate cross entropy loss.
        dlt = predictio
        dlt[lab == 1] -= 1
        dlt = dlt / totalBatch

        # Lets compute grad functions.
        derivW = []
        derivb = []
        ones = np.ones((1, totalBatch))

        # We added delta to calculate backward propagation.
        for i in reversed(range(totalLayer)):
            derivW.append(listOut[i].T @ dlt)
            derivb.append(ones @ dlt)
```

```
            dlt = deriv[i] * (dlt @ W[i].T)

        gradients = {'W': derivW[::-1], 'b': derivb[::-1]}

        return ls, gradients

    def pred(self, inp,pmeters,neur, classify=True):
        W = pmeters["W"]
        b = pmeters["b"]
        neur = neur
        listOut = [inp]
        totalLayer = 3

        # On that side, our prediction function happens.
        variab = inp @ W[0] + b[0]
        variab1 = self.activation(neur[0], variab)[0]
        variab2 = variab1 @ W[1] + b[1]
        variab21 = self.activation(neur[1], variab2)[0]
        variab3 = variab21 @ W[2] + b[2]
        variab32 = self.activation(neur[2], variab3)[0]
        variabb = (np.argmax(variab32, axis=1) + 1).T
        variabb = np.reshape(variabb, (variabb.shape[0], 1))
        return variabb if classify is True else variab32

    def encodeOne(self, inp, tot=250):
        inpEncode = np.zeros((inp.shape[0], 0))

        for i in range(inp.shape[1]):
            temp = np.zeros((inp.shape[0], tot))
            tempp = np.arange(inp.shape[0])
            temp[tempp, (inp-1)[:, i]] = 1
            inpEncode = np.hstack((inpEncode, temp))
```

```
        return inpEncode


    def activation(self, neur, inp):
        if neur == "sigmoid":
            act = 1 / (1 + np.exp(-inp))
            der = act * (1 - act)
            return act, der
        if neur == "softmax":
            act = np.exp(inp) / np.sum(np.exp(inp), axis= 1, keepdims=True)
            der = None
            return act, der
        return None

h5data = "data2.h5"
h5 = h5py.File(h5data, 'r')
words = h5['words'][()]
trainx = h5['trainx'][()]
traind = h5['traind'][()]
valx = h5['valx'][()]
vald = h5['vald'][()]
testx = h5['testx'][()]
testd = h5['testd'][()]
h5.close()

traind = np.reshape(traind, (traind.shape[0], 1))
vald = np.reshape(vald, (vald.shape[0], 1))
testd = np.reshape(testd, (testd.shape[0], 1))
words = np.reshape(words, (words.shape[0], 1))

ratLearn = 0.15
alpha = 0.85
epoch = 50
totalBatch = 200
```

Gökcan Kahraman
21702271

```
D = 8
P = 64
layerHid = [D, P]

tot = [750]
tot += layerHid
tot.append(250)

totalLayer = len(tot) - 1

network = Q2Answer()
info = network.train(trainx, traind, valx, vald,tot, ratLearn, epoch, totalBatch, alpha)
listTRloss, listVAloss, listTRacc, listVAacc,pmeters,neur = info.values()
print()
classTestPred = network.pred(network.encodeOne(testx),pmeters,neur)
print("\n\nTest accuracy for (D, P) = (8, 64): ", (classTestPred == testd).mean() * 100)

# 16, 128
D = 16
P = 128
layerHid = [D, P]

tot = [750]
tot += layerHid
tot.append(250)

network = Q2Answer()
info = network.train(trainx, traind, valx, vald,tot, ratLearn, epoch, totalBatch, alpha)
listTRloss, listVAloss, listTRacc, listVAacc,pmeters,neur = info.values()
print()
classTestPred = network.pred(network.encodeOne(testx),pmeters,neur)
print("\n\nTest accuracy for (D, P) = (16, 128): ", (classTestPred == testd).mean() * 100)
```

Gökcan Kahraman
21702271

```
# 32, 256
D = 32
P = 256

layerHid = [D, P]

tot = [750]
tot += layerHid
tot.append(250)

network = Q2Answer()
info = network.train(trainx, traind, valx, vald,tot, ratLearn, epoch, totalBatch, alpha)
listTRloss, listVAloss, listTRacc, listVAacc,pmeters,neur = info.values()
print()

classTestPred = network.pred(network.encodeOne(testx),pmeters,neur)
print("\n\nTest accuracy for (D, P) = (32, 256): ", (classTestPred == testd).mean() * 100)

w = 5  # Total Prediction

np.random.seed(1907)
p = np.random.permutation(testx.shape[0])
testx = testx[p][:w]
testd = testd[p][:w]

testx_e = network.encodeOne(testx)
test_pred = network.pred(testx_e,pmeters,neur, False)

n = 10
s = (np.argsort(-test_pred, axis=1) + 1)[:, :n]

for i in range(w):
```

Gökcan Kahraman
21702271

```
    print("\n---------------")
    print("Sentence:", words[testx[i][0] - 1], words[testx[i][1] - 1], words[testx[i][2] - 1])
    print("Label:", words[testd[i] - 1])
    for j in range(n):
        print(str(j + 1) + ". ", words[s[i][j] - 1])
```