HW 1 Solutions

Author:
Fateme Noorzad

StudentID:
810198271

Deep Learning and Applications
Spring 2021

Contents

Chapter 1

Introduction

In this homework, various kinds of neural networks are studied. In the first problem, after the required preprocesses on UTKFaces' data set, fully connected neural networks are implemented to carry out regression and classification tasks. In the second and third problem, a given grayscale image goes through a convolutional neural network with specific features and the goal is to find out the output of this network.

Chapter 2

Question 1

In this problem, the UKTFaces dataset is given, and the goal of it is to perform regression and classification, in order to predict some labels.

To achieve this goal, first of all, we need to read this dataset and perform some preprocessing on it. Therefore, the given problem is divided into two sub-problems.

2.1 Read and Preprocess Dataset

Due to the size of the given dataset, reading it without any tricks is almost an impossible task. The reason behind the stated statement is that it takes a long time to perform the mentioned job, as well as the fact that due to the images' dimensionality, RAM becomes full.

To solve this issue, first of all, 5000 images are selected randomly from the main dataset, and copied to create a subset. Although this task is time-consuming too, it is more efficient than reading and preprocessing the whole dataset.

After copying, these 3000 images and their labels are read. Based on the tasks which we are asked to do, three labels of these images are desired. These labels include age, gender, and race. Therefore, a dictionary is created to store all these three. For instance, Figure 2.1 depicts one of them with its age, gender, and race.

FIGURE 2.1: The sample image from UTKFace dataset

Note that to work with these images easier, the read pixel data are flattened and then saved in a variable. As for the second step, we need to train a normalization and dimension reduction method.

To perform normalization, in the "preprocess" class, various methods are implemented. However, based on what we aim to do, "normalize" method is the best choice. In this method, in addition to transforming the given data, the scaler is also saved for the remaining data in the main dataset to be normalized afterwards. Then, the

normalized data goes through a PCA transformation to change its dimension from 120000 to 128. Again, same as what has been done in the normalizer step, besides transforming the given dataset, the trained PCA is saved for the main UTKFaces dataset's dimension reduction.

Now, since normalization and PCA are in hand, we can read the main data set and employ these preprocesses on them. However, since reading all data at once can cause the mentioned problems, 1000 image at a time is read, preprocessed, and saved as a ".csv" file with its desired features.

After saving all data into 20 files, each containing 1000 data, we need to read the reduced dimension files and save them all together in one variable. Now, what is left to do is to separate the training, validation, and test datasets. After splitting data, 18k of them are set as training dataset while 10% of it is for the validation task, the remaining 2k is set as test dataset. Afterwards, these datasets are transferred into batches of data with size=128.

By doing all the stated procedures, we have batches of data, ready as the input of neural network, which will be discussed in the following section.

2.2 Neural Network

In order to satisfy the given request in the problem, a neural network needs to be implemented. Based on what has been described as the network's features, it has to receive any type of activation function, loss, various number of neurons, and various number of layers. Therefore, all these are among the parameters that initialize the neural network.

As its initialization in the attached code suggests, for providing information to the network such as number of neurons in each layer, number of layers, type of activation function, and loss, a dictionary is pass to the initialization function. Another essential point in training networks which is also mentioned in the described problem, is weights and bias initialization. This matter is also implemented in a separate class and is passed as an input to the initialization function of the network. In addition to the stated inputs, the type of decaying the learning rate for training the weights and biases is a critical input as well.

After all the initialization, as it is known, a neural network consists of two main procedures to be implemented and trained completely. In the below subsections, both of these procedures and their implementation in the attached code are described.

2.2.1 Forward Propagation (Feedforward)

Based on the structure of a fully connected neural network, as depicted in Figure 2.2, all of the inputs of a specific neuron are multiplied to the weights connecting the stated neuron to the next layers' neurons. In fact, neurons in each layer are connected to all the neurons of the previous later, and the weights in the weighted sum defines their activeness or the strength of their connection. These weights can also be seen in Figure 2.2.

Besides, there are also some biases added after the multiplication which are all represented in Figure 2.2. They are the identification of whether each neuron is active or not.

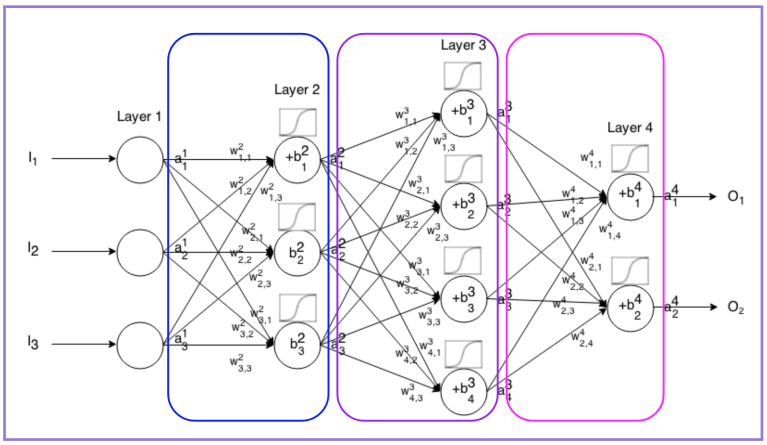

FIGURE 2.2: A fully connected neural network with 4 layers

However, it is clear that the mentioned steps are all linear, which will not provide any additional knowledge for training weights and biases. Therefore, after the mentioned calculations, we are doomed to employ a non-linear function. As depicted in Figure 2.2, one of them is sigmoid function, although nowadays other functions, such as rectified linear unit (a.k.a ReLU) are used instead.

Therefore, if we want to sum all the mentioned steps in mathematical format, we get:

$$\underline{a}^{(i)} = Activation\ Function\{\mathbf{W}\underline{a}^{(i-1)} + \underline{b}\}$$

where

$$\underline{a}^{(i)} = \begin{bmatrix} a_0^{(i)} \\ a_1^{(i)} \\ \vdots \\ a_n^{(i)} \end{bmatrix}, \ \mathbf{W} = \begin{bmatrix} w_{00} & w_{01} & \dots & w_{0n} \\ \vdots & \vdots & \dots & \vdots \\ w_{n0} & w_{n1} & \dots & w_{nn} \end{bmatrix}, \ \underline{b} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$

The above formulation is what has been implemented in "forwProp" function.

2.2.2 Backward Propagation

Learning is a neural network is basically finding the minimum of a certain function called the cost function or loss function. In fact, this function represents a comparision between the desired results and what the network predicts. The difference helps the network to gain insight of how to adjust weights and biases. Therefore, minimizing this function based on various tools provided by optimization techniques provides the best possible performance of a network on all samples.

Therefore, we required a method to perform this calculations, and that method is the back propagation. In fact, in this method, the desire of the last layer to reach the best result is propagated backwards so that all weights and biases in the network be adjusted accordingly. In addition, only the desire of one neuron is not the goal, working the best through all the samples is the altimate aim of the network. As a result the whole samples help in adjusting these parameters in order to get the best possible performance of the network.

In practice, it takes a long time to add up the influence of every single sample to perform optimization algorithms. On the grounds of this fact, batches of data help

solve the gradient value with algorithms such as stochastic gradient descent.

Based on the above descriptions, the network is created.

2.2.3 Regression

For estimating the age of each image, a regression problem needs to be solved. Based on the given information in the problem, the activation function for all layers is leaky ReLU and the loss function is L2.

Leaky ReLU is the same as ReLU, except that for inputs less than zero, its output is a coefficient of the output, where this coefficient is usually set as 0.01. Mathematically speaking:

$$Leaky\ ReLU(x) = max\{x, 0.01 * x\}$$

Therefore, its derivation can be calculated easily:

$$Leaky\ ReLU'(x) = \begin{cases} 1 & x > 0 \\ 0.01 & x < 0 \end{cases}$$

Based on these calculations, the LeakyReLU class is complimented in the attached code to be used as the activation function for the regression task.

In addition, L2 loss can be represented in mathematical format as:

$$L2\ Loss(True\ Value,\ Predicted\ Value) = \frac{1}{2}||True\ Value - Predicted\ Value||^2$$

Therefore, its derivation can be calculated easily:

$$L2\ Loss(True\ Value,\ Predicted\ Value) = ||True\ Value - Predicted\ Value||^2$$

Based on these calculations, the L2 class is complimented in the attached code to be used as the loss function for the regression task.

Then, to perform the regression task, data are read, and are represented in batch sizes as mentioned before. As the next step, network is initialized, and trained with the given parameters and data.

2.2.4 Weight Initialization

Three methods of weight initialization is performed in this manner, and a network of 3 layers while its hidden layer has 64 neurons is implemented and trained. These three methods are "Zero Initialization", "Random Initialization", and "Xaveir Initialization". The results of training, evaluation, and test RMSE can be seen in the below figures.

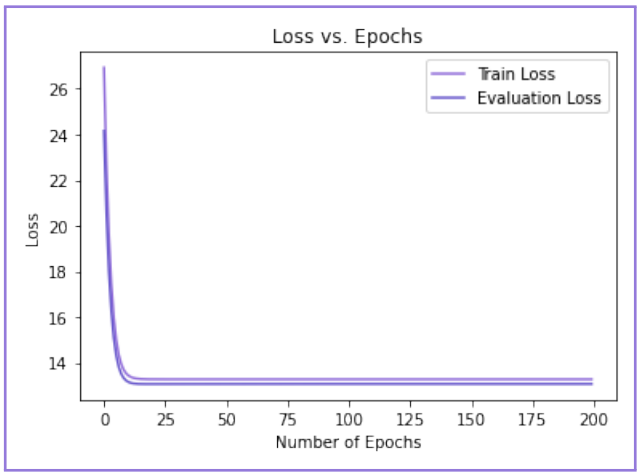

FIGURE 2.3: The loss of training and evaluation in a three layered network with zero weight initialization

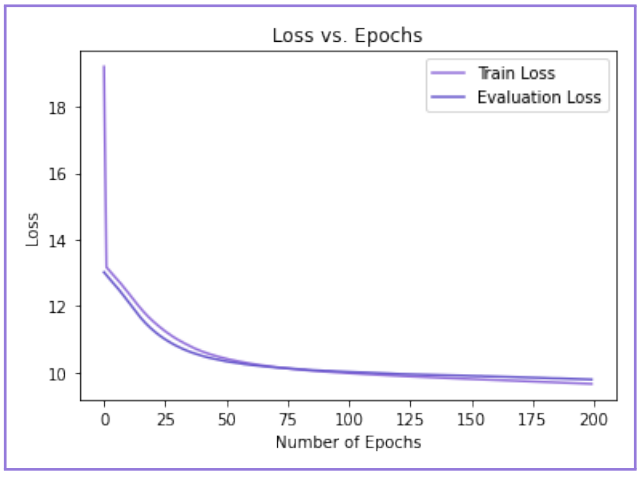

FIGURE 2.4: The loss of training and evaluation in a three layered network with random weight initialization

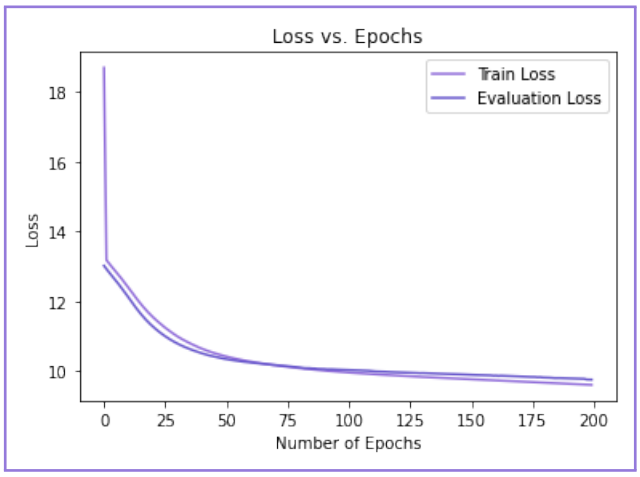

FIGURE 2.5: The loss of training and evaluation in a three layered network with xaveir weight initialization

Various weight initialization	Zero	Random	Xavier
RMSE of Test	19.1553	14.9987	14.9507

FIGURE 2.6: The RMSE of test data of the trained network with various weight initialization methods

The above figures suggest that xavier and random initialization have far better performance than zero initialization and since random initialization is implemented easier this model is selected for the other parts of the problem. (Note that RMSE and loss are almost the same. Due to the 0.5 factor in L2 definition, its value is almost half of RMSEs that can be seen in Figure 2.6)

Note that the number of layers and also the number of neurons in each cell are selected based on various implementation which are mentioned in code. The results of them can be seen in the below plots:

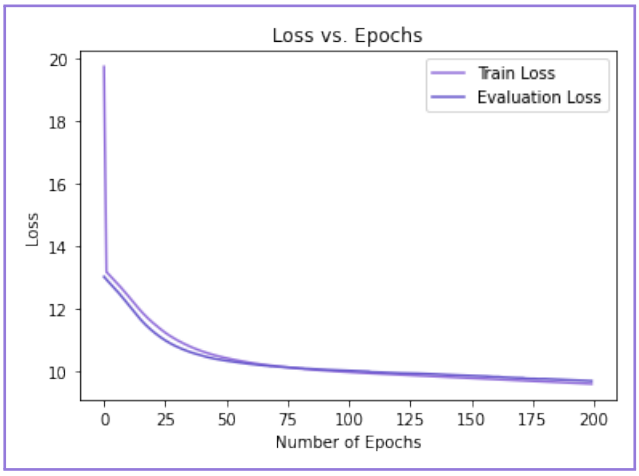

FIGURE 2.7: The loss of training and evaluation in a three layered network with random weight initialization and 64 neurons in hidden layer

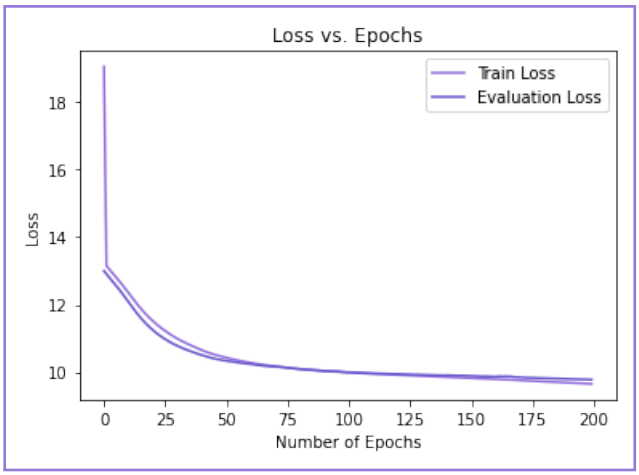

FIGURE 2.8: The loss of training and evaluation in a three layered network with random weight initialization and 32 neurons in hidden layer

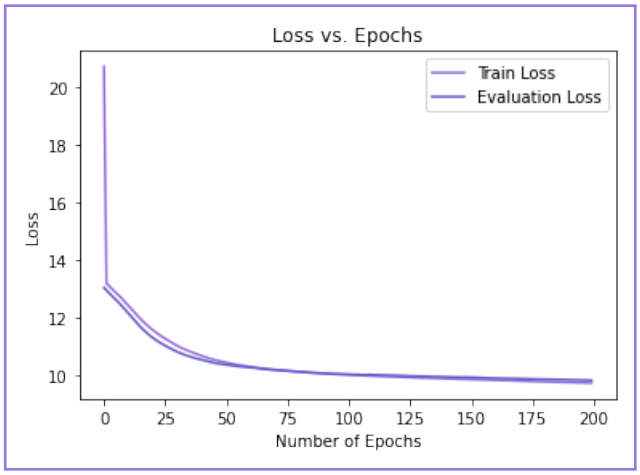

FIGURE 2.9: The loss of training and evaluation in a three layered network with random weight initialization and 16 neurons in hidden layer

Number of Neuron of Hidden Layers' Neurons	64	32	16
RMSE of Test	14.9183	15.0268	15.033

FIGURE 2.10: The RMSE of test data of the trained network with random weight initialization and various neurons in hidden layer

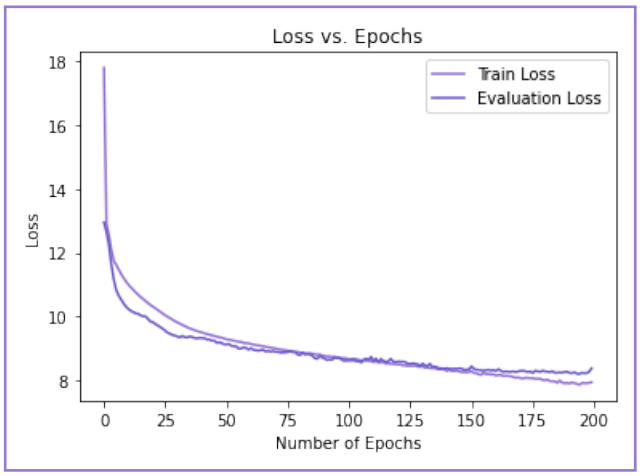

FIGURE 2.11: The loss of training and evaluation in a four layered network with random weight initialization and 64 and 32 neurons in hidden layers respectively

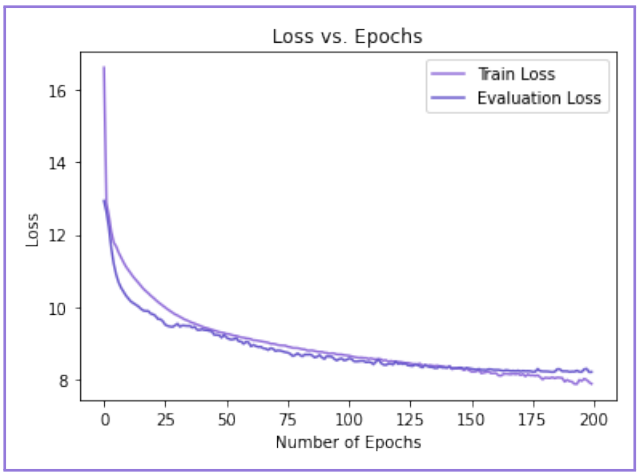

FIGURE 2.12: The loss of training and evaluation in a three layered network with random weight initialization and 64 and 16 neurons in hidden layers respectively

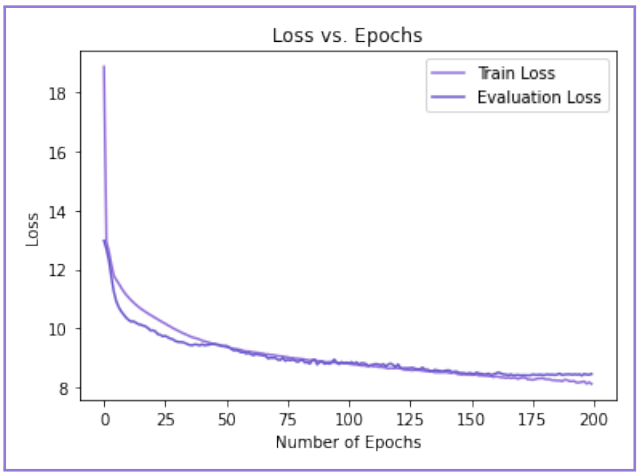

FIGURE 2.13: The loss of training and evaluation in a three layered network with random weight initialization and 32 and 8 neurons in hidden layers respectively

Number of Neuron of Hidden Layers' Neurons	64,32	64,16	32,8
RMSE of Test	13.3533	12.9904	13.3757

FIGURE 2.14: The RMSE of test data of the trained network with random weight initialization and various neurons in hidden layers

The above plots suggest that among the three layered networks, the one with 64 hidden layers is the best one, while among the four layered one, the combination of (64,16) is the best. Although, the comparision between three and four layered networks, suggests that the network with two hidden layers is better, due to the simplicity of model as well as the fact that the difference between the two models is not considerable, and the noise on the loss function is less, a three layered network

while the number of neurons in the hidden layer is 64 and the weight initialization is random is selected.

2.2.5 Learning Rate

In order to change the learning rate, weight is changed with a regularization parameter. (under the name "lambda" in attached code) The result of this matter can be seen in the Figure2.15.

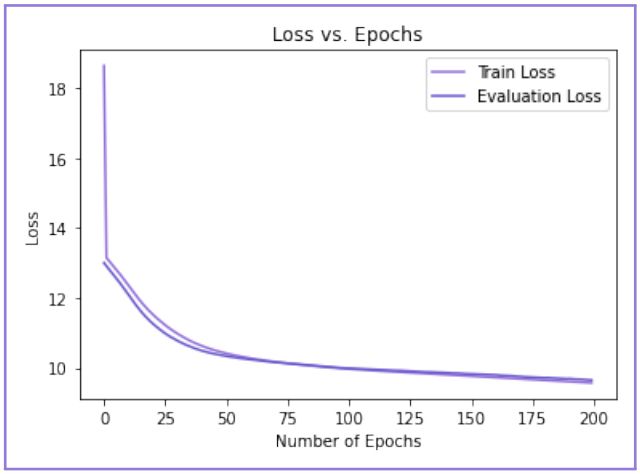

FIGURE 2.15: The loss of training and evaluation in a three layered network with random weight initialization and 64 neurons in hidden layer with changing learning rate

In addition to loss, the value of RMSE on test data is 14.84. Both the above figure and RMSE suggest that the last setting of learning with momentum created a better combination for the training of this neural network.

2.2.6 Overfit

If the number of epochs of training is increased, it can be seen that the loss value on evaluation dataset is increased, which is a representation of overfitting on training dataset. This matter becomes clear if we take a look at the three below figure.

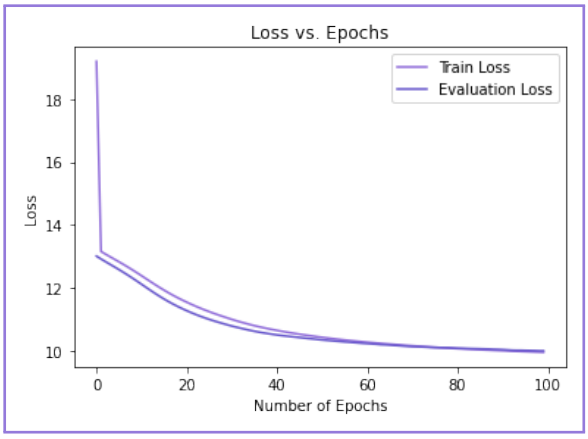

FIGURE 2.16: The loss of training and evaluation in a three layered network with random weight initialization and 64 neurons in hidden layer

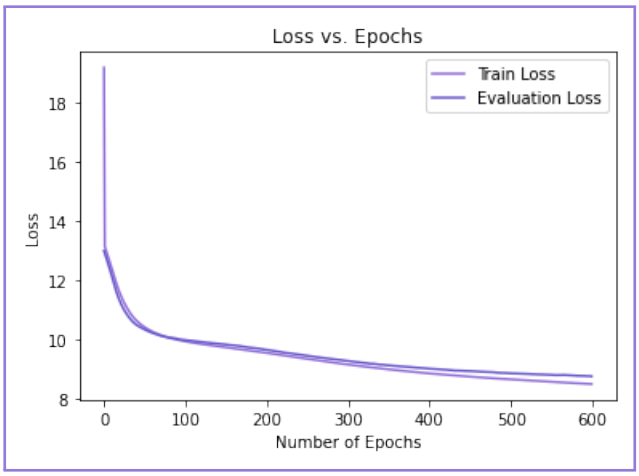

FIGURE 2.17: The loss of training and evaluation in a three layered network with random weight initialization and 64 neurons in hidden layer

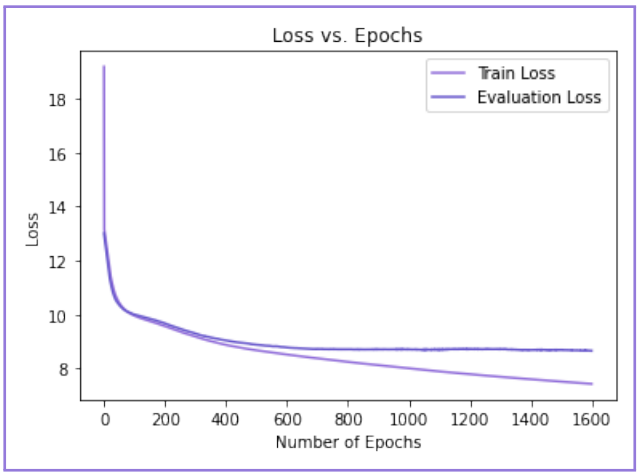

FIGURE 2.18: The loss of training and evaluation in a three layered network with random weight initialization and 64 neurons in hidden layer

2.2.7 Classification

In this part we are asked to classify the images based on the race of people, and employing the neural network form regression part with three main changes. One is that, in classification problem we are required to employ softmax activation function on the last layer, meaning, all the hidden layers are activated with leaky ReLU, while the last layer is activated with softmax. The other difference is that the loss function in this case is negative log likelihood. The last difference is that since we want to classify 5 different classes, the output layer needs to have 5 neurons, while this number was one for the regression task. Mathematically speaking, softmax is represented as below:

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

The above equation suggest that softmax's derivation is one. Therefore, a softmax class is implemented in the attached code.

Now, for the negative log likelihood, in general, we have:

$$NLL(x) = -log(x)$$

However, when softmax is its input:

$$p_i = Softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}},\ L_i = -log(p_i) \rightarrow \frac{\partial L_i}{\partial x_i} = p_i - 1$$

Therefore, the differentiate of the negative log likelihood with respect to the softmax layer is computed as above. Based on this matter, the NLL class is implemented in the attached code.

One point that is essential to be mentioned here is that since we are performing a classification task here, the labels went through a one hot encoder and then used to train the network.

By taking all the mentioned steps, the requested plots are depicted as below:

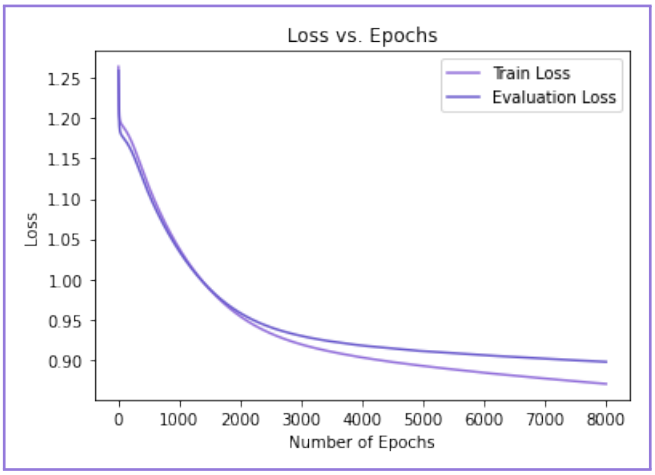

FIGURE 2.19: The loss of training and evaluation in a three layered network with random weight initialization and 64 neurons in hidden layer

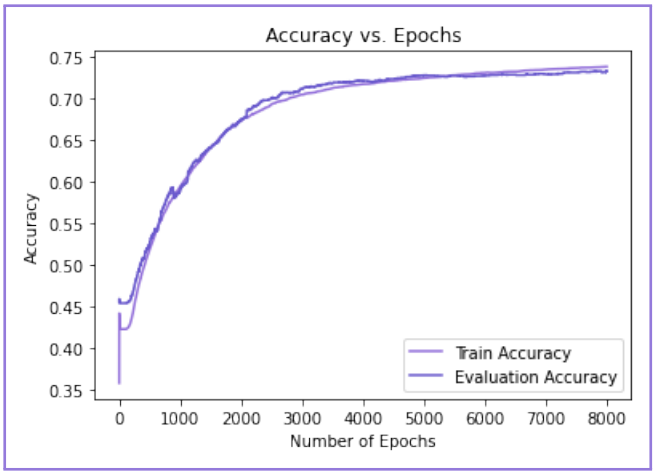

FIGURE 2.20: The accuracy of training and evaluation in a three layered network with random weight initialization and 64 neurons in hidden layer

Besides the above results, the accuracy of this network on test data after 8000 epochs is 72.2.

Chapter 3

Question 2

In this problem, a given picture in grayscale as Table 4.1 goes through a convolutional layer.

2	1	3	3
1	6	2	5
1	4	5	2
6	6	5	3

Table 3.1: Input picture in grayscale

The layer consists of a filter with padding and stride. Afterward, the output of the stated layer goes through a max pooling layer. The goal of this problem is to find out what the output of these two layers is.

To do so, first of all, on Table**??** the stated padding is carried out. Meaning, one row of zero is added at the bottom and top of the table. In addition, one column of zeros are added at the left and right of it. The result can be seen as Table 3.2:

0	0	0	0	0	0
0	2	1	3	3	0
0	1	6	2	5	0
0	1	4	5	2	0
0	6	6	5	3	0
0	0	0	0	0	0

Table 3.2: Added one padding to Table 4.1

The next step is to find the convolution of the above table with the given filters. To do so, each filter is set on every 3*3 squares in the picture with padding=1 as depicted in Table 3.2, and each cell in it is multiplied to the corresponding cell in the given filter. The procedure can be seen in Table 3.3 for the first filter. The same procedure is carried out for the other two filters as well. The results of these filterings are depicted in Table 3.4, Table 3.5, and Table 3.6.

0*0+0*1+0*2 + 0*1+2*0+1*1 + 0*2+1*1+6*0	0*0+0*1+0*2 + 2*1+1*0+3*1 + 1*2+6*1+2*0	0*0+0*1+0*2 + 1*1+3*0+3*1 + 6*2+2*1+5*0	0*0+0*1+0*2 + 3*1+3*0+0*1 + 2*2+5*1+0*0
0*0+2*1+1*2 + 0*1+1*0+6*1 + 0*2+1*1+4*0	2*0+1*1+3*2 + 1*+6*0+2*1 + 1*2+4*1+5*0	1*0+3*1+3*2 + 6*1+2*0+5*1 + 4*2+5*1+2*0	3*0+3*1+0*2 + 2*1+5*0+0*1 + 5*2+2*1+0*0
0*0+1*1+6*2 + 0*1+1*0+4*1 + 0*2+6*1+6*0	1*0+6*1+2*2 + 1*1+4*0+5*1 + 6*2+6*1+5*0	6*0+2*1+5*2 + 4*1+5*0+2*1 + 6*2+5*1+3*0	2*0+5*1+0*2 + 5*1+2*0+0*1 + 5*2+3*1+0*0
0*0+1*1+4*2 + 0*1+6*0+6*1 + 0*2+0*1+0*0	1*0+4*1+5*2 + 6*1+6*0+5*1 + 0*2+0*1+0*0	4*0+5*1+2*2 + 6*1+5*0+3*1 + 0*2+0*1+0*0	5*0+2*1+0*2 + 5*1+3*0+0*1 + 0*2+0*1+0*0

Table 3.3: Result of filtering Table 3.2

2	13	18	12
11	16	33	17
23	34	35	23
15	25	18	7

Table 3.4: Output of convolution layer with filter #1

8	4	14	5
6	17	12	13
13	18	25	9
10	12	11	8

Table 3.5: Output of convolution layer with filter #2

10	24	17	16
22	23	41	17
28	42	35	28
14	31	32	19

Table 3.6: Output of convolution layer with filter #3

After the convolution layer, there exists a max pooling layer with padding=0 and stride=2. If the ouputs of it go through the this stated layer, among each 2*2 square, the cell with the maximum value needs to be chosen. The below tables show the cells with the mentioned feature in each of the above outputs:

2	13	18	12
11	16	33	17
23	34	35	23
15	25	18	7

TABLE 3.7: The output of convolution layer with filter #1 with selected cells

8	4	14	5
6	17	12	13
13	18	25	9
10	12	11	8

TABLE 3.8: The output of convolution layer with filter #2 with selected cells

10	24	17	16
22	23	41	17
28	42	35	28
14	31	32	19

TABLE 3.9: The output of convolution layer with filter #3 with selected cells

Therefore, if we exclude the highlighted cells to create the output of max pooling layer, we get the three tables below:

16	33
34	35

TABLE 3.10: The output of max pooling, and convolution layer with filter #1

17	14
18	25

TABLE 3.11: The output of max pooling, and convolution layer with filter #2

24	41
42	35

TABLE 3.12: The output of max pooling, and convolution layer with filter #3

Chapter 4

Question 3

In this problem, we are given a figure in the grayscale format as the input of a neural network. The NN can be represented with 4 weights as w_{11}, w_{12}, w_{21}, and w_{22} as the Table 4.1 depicted below:

w_{11}	w_{12}
w_{21}	w_{22}

TABLE 4.1: The weights of the CNN

We are asked to find the gradient of the given loss function with respect to all the weights in the given plot. To do so, we need to find the result of each part until we reach the output.

As the first step, we need to perform convolution between the given picture and the stated weights in Table 4.1 with stride=2. The following computations represent this matter:

$$\begin{cases} c_{11} = x_{11}w_{11} + x_{12}w_{12} + x_{21}w_{21} + x_{22}w_{22} \\ c_{12} = x_{13}w_{11} + x_{14}w_{12} + x_{23}w_{21} + x_{24}w_{22} \\ c_{21} = x_{31}w_{11} + x_{32}w_{12} + x_{41}w_{21} + x_{42}w_{22} \\ c_{22} = x_{33}w_{11} + x_{34}w_{12} + x_{43}w_{21} + x_{44}w_{22} \end{cases}$$

As the next step, the above calculations go through an activation function ($g(.)$). Therefore, based on the diagram given, we have:

$$\begin{cases} y_1 = g(c_{11}) \\ y_2 = g(c_{12}) \\ y_3 = g(c_{21}) \\ y_4 = g(c_{22}) \end{cases}$$

Afterward, the above results are weighted to create the output as below:

$$output = w_1y_1 + w_2y_2 + w_3y_3 + w_4y_4$$

Now, based on the above calculations, we can calculate the derivatives which are the goal of this problem.

- w.r.t w_1:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial (output)} \frac{\partial (output)}{\partial w_1} = y_1(output - t)$$

- w.r.t w_2:

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial (output)} \frac{\partial (output)}{\partial w_2} = y_2(output - t)$$

- w.r.t w_3:

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial (output)} \frac{\partial (output)}{\partial w_3} = y_3(output - t)$$

- w.r.t w_4:

$$\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial (output)} \frac{\partial (output)}{\partial w_4} = y_4(output - t)$$

- w.r.t w_{11}:

$$\begin{aligned}\frac{\partial L}{\partial w_{11}} &= \frac{\partial L}{\partial (output)} \Big[\frac{\partial (output)}{\partial y_1} \frac{\partial y_1}{\partial c_{11}} \frac{\partial c_{11}}{\partial w_{11}} + \\ &\qquad \frac{\partial (output)}{\partial y_2} \frac{\partial y_2}{\partial c_{12}} \frac{\partial c_{12}}{\partial w_{11}} + \\ &\qquad \frac{\partial (output)}{\partial y_3} \frac{\partial y_3}{\partial c_{21}} \frac{\partial c_{21}}{\partial w_{11}} + \\ &\qquad \frac{\partial (output)}{\partial y_4} \frac{\partial y_4}{\partial c_{22}} \frac{\partial c_{22}}{\partial w_{11}} \Big] \\ &= (output - t)\big[w_1 g'(c_{11}) x_{11} + \\ &\qquad w_2 g'(c_{12}) x_{13} + \\ &\qquad w_3 g'(c_{21}) x_{31} + \\ &\qquad w_4 g'(c_{22}) x_{33} \big]\end{aligned}$$

- w.r.t w_{12}:

$$\begin{aligned}\frac{\partial L}{\partial w_{12}} &= \frac{\partial L}{\partial (output)} \Big[\frac{\partial (output)}{\partial y_1} \frac{\partial y_1}{\partial c_{11}} \frac{\partial c_{11}}{\partial w_{12}} + \\ &\qquad \frac{\partial (output)}{\partial y_2} \frac{\partial y_2}{\partial c_{12}} \frac{\partial c_{12}}{\partial w_{12}} + \\ &\qquad \frac{\partial (output)}{\partial y_3} \frac{\partial y_3}{\partial c_{21}} \frac{\partial c_{21}}{\partial w_{12}} + \\ &\qquad \frac{\partial (output)}{\partial y_4} \frac{\partial y_4}{\partial c_{22}} \frac{\partial c_{22}}{\partial w_{12}} \Big] \\ &= (output - t)\big[w_1 g'(c_{11}) x_{12} + \\ &\qquad w_2 g'(c_{12}) x_{14} + \\ &\qquad w_3 g'(c_{21}) x_{32} + \\ &\qquad w_4 g'(c_{22}) x_{34} \big]\end{aligned}$$

- w.r.t w_{21}:

$$\begin{aligned}\frac{\partial L}{\partial w_{21}} &= \frac{\partial L}{\partial (output)} \Big[\frac{\partial (output)}{\partial y_1} \frac{\partial y_1}{\partial c_{11}} \frac{\partial c_{11}}{\partial w_{21}} + \\ &\qquad \frac{\partial (output)}{\partial y_2} \frac{\partial y_2}{\partial c_{12}} \frac{\partial c_{12}}{\partial w_{21}} + \\ &\qquad \frac{\partial (output)}{\partial y_3} \frac{\partial y_3}{\partial c_{21}} \frac{\partial c_{21}}{\partial w_{12}} + \\ &\qquad \frac{\partial (output)}{\partial y_4} \frac{\partial y_4}{\partial c_{22}} \frac{\partial c_{22}}{\partial w_{21}} \Big] \\ &= (output - t)\big[w_1 g'(c_{11}) x_{21} + \\ &\qquad w_2 g'(c_{12}) x_{23} + \\ &\qquad w_3 g'(c_{21}) x_{41} + \\ &\qquad w_4 g'(c_{22}) x_{43} \big]\end{aligned}$$

- w.r.t w_{22}:

$$\begin{aligned}\frac{\partial L}{\partial w_{22}} &= \frac{\partial L}{\partial (output)} \Big[\frac{\partial (output)}{\partial y_1} \frac{\partial y_1}{\partial c_{11}} \frac{\partial c_{11}}{\partial w_{22}} + \\ &\qquad \frac{\partial (output)}{\partial y_2} \frac{\partial y_2}{\partial c_{12}} \frac{\partial c_{12}}{\partial w_{22}} + \\ &\qquad \frac{\partial (output)}{\partial y_3} \frac{\partial y_3}{\partial c_{21}} \frac{\partial c_{21}}{\partial w_{22}} + \\ &\qquad \frac{\partial (output)}{\partial y_4} \frac{\partial y_4}{\partial c_{22}} \frac{\partial c_{22}}{\partial w_{22}} \Big] \\ &= (output - t)\big[w_1 g'(c_{11}) x_{22} + \\ &\qquad w_2 g'(c_{12}) x_{24} + \\ &\qquad w_3 g'(c_{21}) x_{42} + \\ &\qquad w_4 g'(c_{22}) x_{44} \big]\end{aligned}$$

Chapter 5

Appendix: How to Run the Codes

All the codes can be found in the codes folder. Note that first of all, DatasetProcess file needs to be run. In this file change the direction of the main dataset, copied images, and csv files' storage. Afterwards, run regression or classification codes, based on what you aim to see. Note that you need to change the location of the csv file in these two codes based on where you saved them.

Chapter 6

References

- https://stackoverflow.com/questions/49237177/copying-multiple-images-in-a-list-from-
- https://www.codegrepper.com/code-examples/python/how+to+read+all+images+from+a+folder+in+python+using+opencv
- https://stackoverflow.com/questions/8290397/how-to-split-an-iterable-in-constant-siz
- https://stackoverflow.com/questions/45685538/whats-the-advantage-of-using-yield-in-i
- https://medium.com/@udaybhaskarpaila/multilayered-neural-network-from-scratch-using-
- https://towardsdatascience.com/math-neural-network-from-scratch-in-python-d6da9f29ce
- https://towardsdatascience.com/an-introduction-to-neural-networks-with-implementatio
- https://www.youtube.com/watch?v=IHZwWFHWa-w&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=3
- https://github.com/mnielsen/neural-networks-and-deep-learning/blob/master/src/network2.py
- http://neuralnetworksanddeeplearning.com/index.html
- https://techburst.io/improving-the-way-we-work-with-learning-rate-5e99554f163b
- https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5
- https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-lik