به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

یادگیری عمیق با کاربرد در بینایی ماشین و پردازش صوت

Deep Neural Networks

تمرین شماره ۱

عرفان میرزایی

Erfan Mirzaei

۸۱۰۱۹۹۲۸۹

فروردین ماه ۱۴۰۰

April 2021

Contents

Table of Figures

Abstract

Implementing algorithms from scratch is an effective way to gain a more profound knowledge and understanding of an algorithm. In this question, we implement a deep neural network from scratch with NumPy. For this purpose, the UTKFace dataset includes more than 23000 face images of people of different ages, races, and genders. In the first question, I train an artificial neural network for age prediction, then in the second part for race prediction, and last for gender prediction.

In the second question, I do calculations for a convolution layer. In the third question, I do a “Back propagation” operation for a simple convolution neural network that includes one convolution layer and one fully connected layer.

Question 1

Problem Definition and Dataset

Implementing algorithms from scratch is an effective way to gain more profound knowledge and understanding of an algorithm. In this question, we implement a deep neural network from scratch with NumPy. For this purpose, the UTKFace dataset includes more than 23000 face images of people of different ages, races, and genders. In this section, the goal is to train a neural network to predict the age of people from their face images. This dataset splits to train, validation, and test data. We devote 0.9 of images for training and the others for the test set and also use 10 percent of training data for the validation set. In this problem, we use a multi-layer neural network as a Regression method for predicting the age of persons 0 to 116 years old.

Implementation

The implemented code for this question is in the “Codes” folder in the “HW1_Q1_DNN.ipynb” file. For all three sections of this question, we have the same procedure, so after loading data, and splitting to train, validation, and test sets (stratified sampling is used), we should normalize data, in other words, transform data to a normal distribution with zero mean and unit variance. However, in this problem, due to the large size of the dataset and RAM limitation, we cannot use packages, and it is implemented manually. Also, we have this problem with applying PCA transform to our training set. Therefore we use “Incremental PCA” instead of PCA from the “Scikit-Learn” library so that we divide our training data into chunks and, at each iteration, partially fit the data using the “partial. fit” method, and the same procedure for transforming the PCA transformation on training data. To make the optimization easier, we standardize the data and roughly divide each data by standard deviation to have a unit standard deviation. For convenience, I saved the normalized data after PCA; further explanation will come in Appendix1 at the end of the report.

In order to be able to change the network architecture and its hyper parameters easily, I implemented a class for multi-layer neural networks (“DNN” class). I got some inspiration from the MLPClassifier of Scikit-Learn for implementing the methods of the class. Also, I use the same notation as Andrew Ng in deep learning Specialization in Coursera. After training the network, in the last part of the procedure, we test our network based on the test dataset, report the results, and analyze them.

Section 1: Age Prediction (Regression)

In this section, due to the definition of the problem, the network has one node at the last layer, and also, because of applying PCA to the input data, it is a 128-dimensional vector. Thus the network should have 128 nodes in the First Layer. The DNN class gets a list that determines the number of nodes at each layer. Thus we have several layers too. We use L2 Loss for this problem and the “Leaky ReLU” activation function for the last and other layers. For training parameters of the network, we use SGD (Stochastic Gradient Descent) with momentum.

The batch size in this problem is set to 128, and we assume the momentum coefficient to be 0.9. We consider the initial value of the learning rate to be 0.001, which is decaying with a constant step size while training. In this problem, I use a network with two hidden layers with 64 and 32 nodes, respectively. Also, use 2045 as a random seed.

Part a) Zero Initialization vs. Random Initialization

In this part, first, we use Zero initialization for the weights and biases of each layer and train the neural network for 50 epochs with the hyper parameters explained in the last part. The cost of the training set and validation set is as follows:

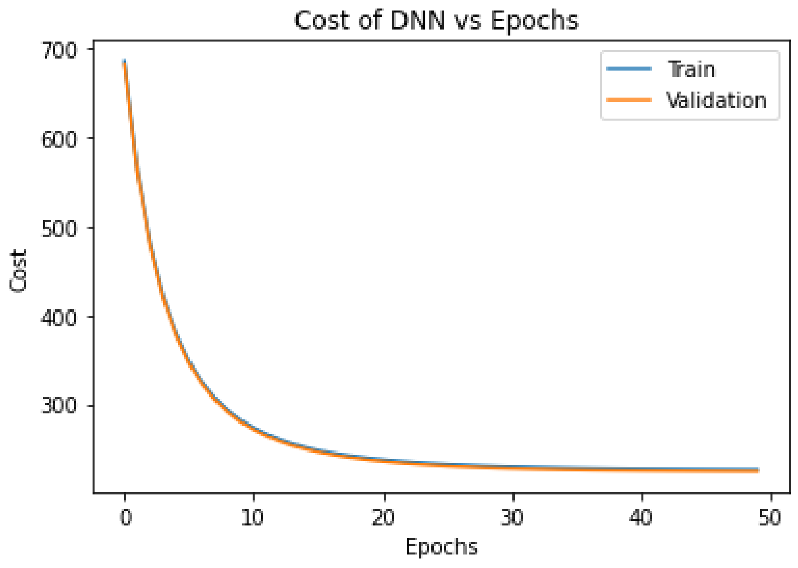

Figure 1- Cost vs. epochs for zero initialization

We can see that there is no over-fitting as the cost is the same for the train and validation set, which means our network performs similarly on unseen data. The RMSE of test data for this network is 21.0.68.

Generally, “Zero Initialization” is not a good approach for initializing parameters since it can use all the capacity of the network due to the symmetry that becomes in updating parameters, and the performance of the network would be the same as a single neuron, in this case, a simple linear regression model.

It is common to use "Random Initialization" for initializing the parameters, and usually use random numbers that come from a standard normal distribution and multiply them to a regulating factor that relates to the architecture of the network. In this assignment, we use 0.1 as a constant regulation factor of random normal numbers due to the network architecture being the same for all questions. The result of training the neural network with "Random Initialization" is as follows:

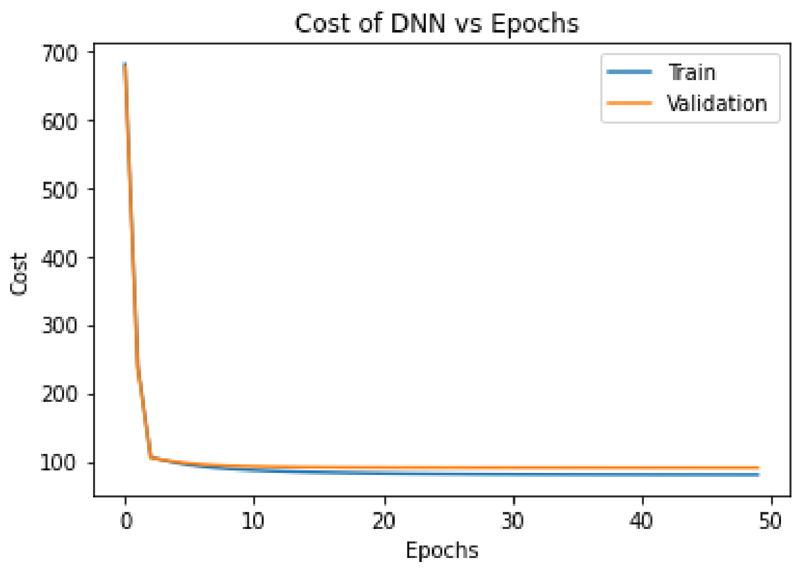

Figure 2-Cost vs. epochs for random initialization

There is a very tiny amount of over-fitting, but it can be neglected. Also, we can see that with random normalization, the amount of cost becomes 1/3, and this shows it is a better initialization. The RMSE of test data for this network is 13.017.

Conclusion

In this part of the first section, we trained a neural network with "random Initialization" and "Zero Initialization" based on the below results. We can conclude that random initialization is a better initialization.

Table 1-Comparison of Random Initialization vs. Zero Initialization

MSE	Zero Initialization	Random Initialization
Cost on Train data	226.168	80.618
Cost on Validation data	224.314	90.575
RMSE for test data	21.018	13.017

Part b) Step decay vs. Constant decay

In the last part, we initialize the learning rate to 0.001and it decays at each epoch with a constant factor. Nevertheless, in this part, we use another method for decay learning through training. We choose the "step decay" method so that it decays the learning rate every t epochs with a constant factor. . The result of training the neural network with "Step decay" is as follows:

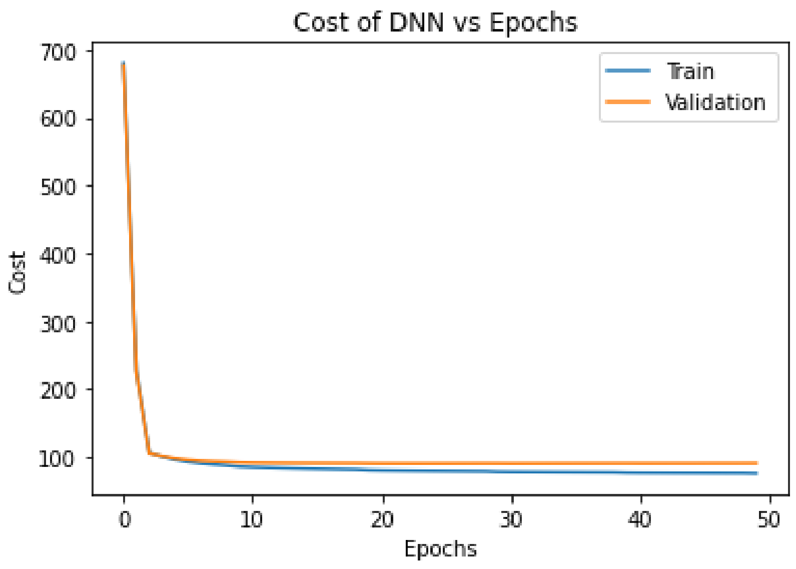

Figure 3-Cost vs. epochs for Step decay

There is a very tiny amount of over-fitting but it can be neglected. The RMSE of test data for this Network is 12.917.

Conclusion

In this part of the first section, we trained a neural network with "Step decay" based on the results below. We can conclude that step decaying for learning is slightly better.

Table 2-Comparison of Step decay vs. Constant decay

MSE	Step decay	Constant decay
Cost on Train data	75.927	80.618
Cost on Validation data	90.735	90.575
RMSE for test data	12,917	13.017

Section 2: Race Prediction (Multi-Class Classification)

In this part, we want to use neural networks for a multi-class classification to predict the race of people from their face images. All the conditions are the same as the last problem, except for the Loss function and activation function and the number of neurons in the output layer. For this problem, we use the "Negative Log-Likelihood" function as a loss function and the "softmax" function for the last layer. Thus we should have five neurons in the last layer because we have five races in our dataset. The race number is an integer from 0 to 4, denoting White, Black, Asian, Indian, and Others (like Hispanic, Latino, and Middle Eastern). Moreover, use one-hot encoding to transform an integer from 0 to 4 to a 5-d vector.
The Negative log-likelihood loss function is as follows:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

Figure 4- Negative log likelihood loss function

Our network for this problem has two hidden layers with 64 and 32 neurons. We use "Random Initialization" and "Step decay," and all the hyper parameters are the same as before. The result of training a neural network is as follows:

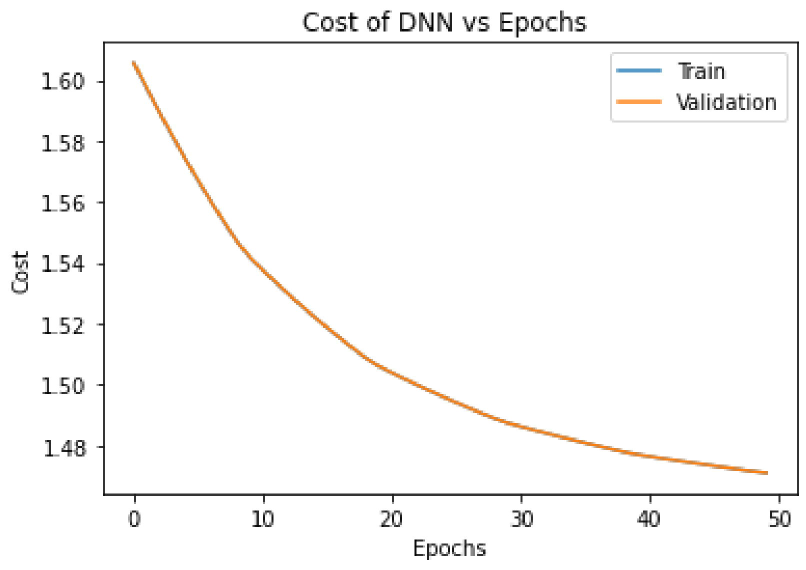

Figure 5- Cost vs. epochs for race prediction

Furthermore, the Accuracy for training and validation set through the training would be as follows:

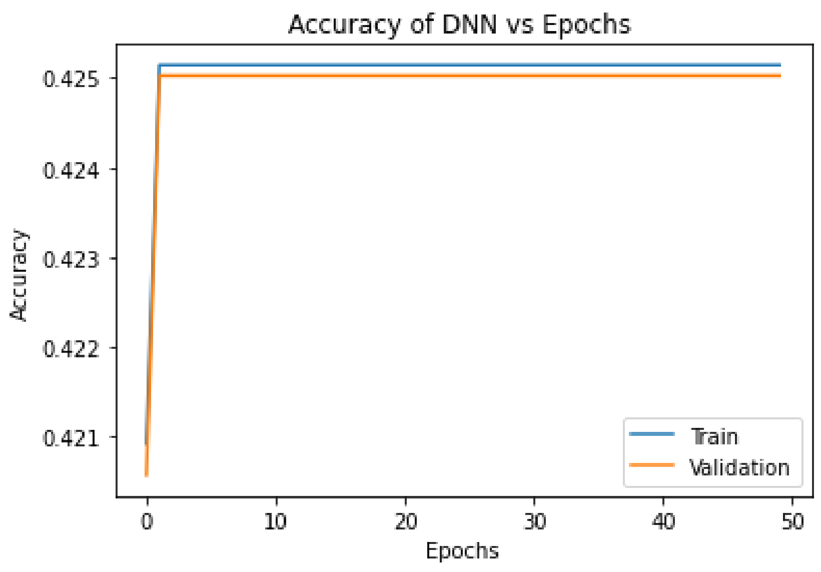

Figure 6-Accuracy vs. epochs for race prediction

We can see from these figures, that there is no effect of over-fitting. Also it can be seen that network hasn't learn too much through the epochs. Furthermore, it seems that the network stuck in one of local optima or a saddle point. The Accuracy of Test set is 0.425. With further analysis, we see that the network predict 0 label for every point in dataset, because our dataset is imbalanced, and about 0.4 for "White" Race.

Table 3-Comparison accuracy of train, validation, and test set for race prediction

	Train	Validation	Test
Accuracy	0.425	0.425	0.425

Section 3: Gender Prediction (Binary Classification)

In this part, we want to use neural networks for binary classification to predict the gender of people from their face images. All the conditions are the same as in the first section except for the loss and activation functions. For this problem, we use the “Logistic Loss” function or “Binary Cross Entropy” as the loss function and also use the “sigmoid” function for the last layer. The gender number is either 0 (male) or 1 (female).

The Logistic Loss” function is as follows:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

$$J(\theta) = \frac{1}{m}[\sum_{i=1}^{m} -y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))]$$

$$m = number\ of\ samples$$

Figure 7-Logistic loss function

Our network for this problem has two hidden layers with 64 and 32 neurons. We use “Random Initialization” and “Step decay,” and all the hyperparameters are the same as before. The result of training a neural network is as follows:

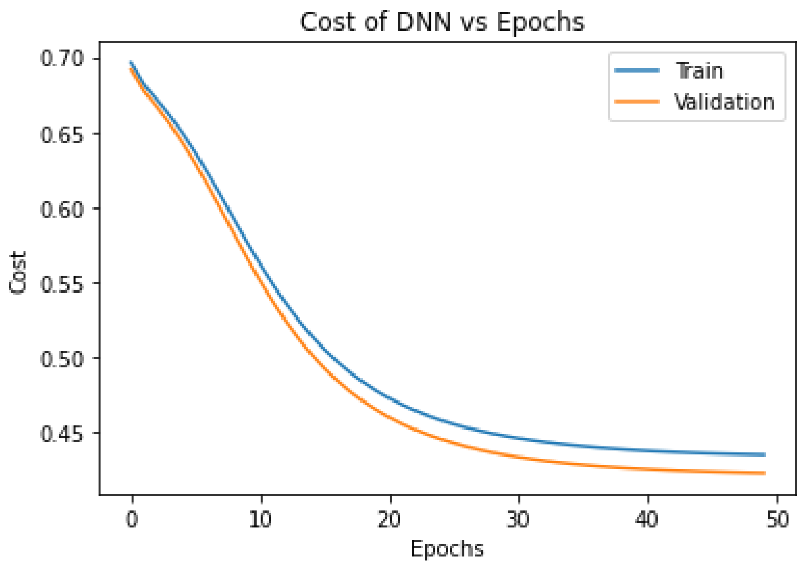

Figure 8-Cost vs. epochs for gender prediction

In addition, the Accuracy for training and validation set through the training would be as follows:

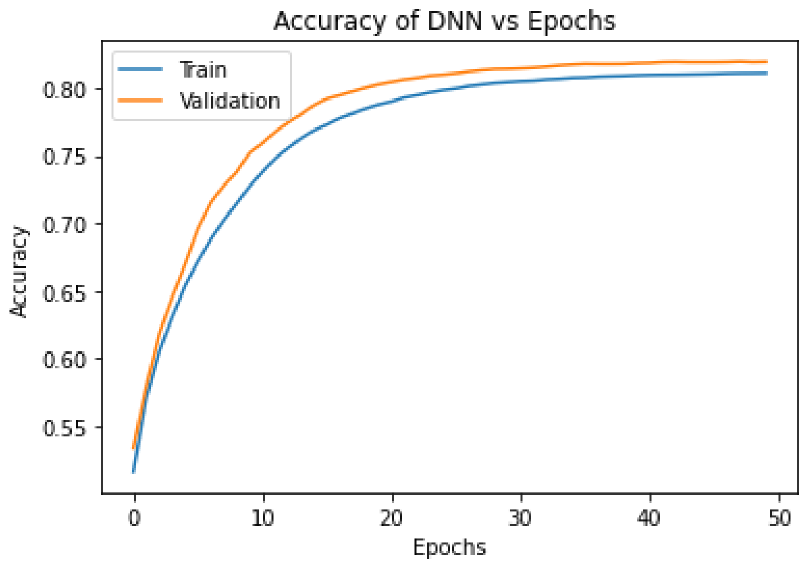

Figure 9-Accuracy vs. epochs for gender prediction

We can see from these figures, there is a very tiny amount of over-fitting but it can be neglected. Also it can be seen that network performs very well on both training and validation dataset. And also the Accuracy of Test set is 0.814.

Table 4-Comparison accuracy of train, validation, and test set for gender prediction

	Train	Validation	Test
Accuracy	0.811	0.819	0.814

Question 2

In this question, we have a 4 * 4 image and should convolve it with three different 3 * 3 filters with stride = 1 and padding = 1.

After adding padding, the image's shape becomes 6 * 6, so that one row of zeros is added to the first and last row of the image, also to the right and left column of the image. The image after adding padding is as follows:

Table 5- Input Image after padding

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

Now convolve the image with 3 filters, and expect to get three 4 * 4 images as outputs, because when we convolve f * f filter with n * n image with padding = p, and stride = s, we'll get the image with $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor * \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$ dimensions. This formula comes from the definition. It's because the filter starts from the left-top of the image after adding padding and gets convolved and then slid to the right in the amount of stride till the end of the first row. Then it slid to the bottom with a stride step and continue just as before. This calculation can be performed manually, or by code. The code of this part is at "HW1_Q2_DNN.ipynb" in the "Codes" Folder. Convolved images are as follows:

Table 2-Convolved Images

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

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

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

Now we should apply max pooling layer to the convolved images from last part. Max pooling layer with f * f filters and stride = s, it's like convolution layer but instead of convolution, gets the maximum element each time. Therefore dimensions of output images should follow the same formula, and thus it would be 2 * 2. The Output images after max pooling would be as follows:

Table 3-Images after max pooling

16	33
34	35

17	14
18	25

24	41
42	35

Question 3

In this question, we want to apply the “Backpropagation” operation to a simple convolution neural network that has one convolution layer without max pooling, and one fully connected layer. For simplicity, we neglect biases in layers.

For this purpose, we should start from the loss function (MSE), therefore gradient of loss function w.r.t[1] output of network would be:

$$\mathrm{dout} = \frac{\partial \mathrm{L}}{\partial \mathrm{out}} = -(\mathrm{t} - \mathrm{out}) = \mathrm{out} - \mathrm{t}$$

We used the same notation of question 1 because we want to calculate the gradient of loss w.r.t different variables, just use the variable name, and dfoo shows the gradient of Loss function w.r.t foo. Then we should propagate this gradient, exactly like what we did in question 1. Before calculating gradients, first, we use the vectorized notation for Y and W, so that:

$$Y = \begin{bmatrix} Y1 \\ Y2 \\ Y3 \\ Y4 \end{bmatrix}, W = [W1 \quad W2 \quad W3 \quad W4], C = \begin{bmatrix} C_{11} \\ C_{12} \\ C_{21} \\ C_{22} \end{bmatrix}, Y = g(C)$$

So we will have:

$$\mathrm{dY} = \frac{\partial \mathrm{L}}{\partial \mathrm{Y}} = \frac{\partial \mathrm{L}}{\partial \mathrm{out}} * \frac{\partial \mathrm{out}}{\partial \mathrm{Y}} = dout * W^T = \begin{bmatrix} (out-t)*W1 \\ (out-t)*W2 \\ (out-t)*W3 \\ (out-t)*W4 \end{bmatrix}$$

$$\mathrm{dW} = \frac{\partial \mathrm{L}}{\partial \mathrm{W}} = \frac{\partial \mathrm{L}}{\partial \mathrm{out}} * \frac{\partial \mathrm{out}}{\partial \mathrm{W}} = \mathrm{dout} * Y^T = [(out-t)*Y1 \quad (out-t)*Y2 \quad (out-t)*Y3 \quad (out-t)*Y4]$$

And for convolution layer:

$$\mathrm{dC} = \frac{\partial \mathrm{L}}{\partial \mathrm{C}} = \frac{\partial \mathrm{L}}{\partial \mathrm{Y}} * \frac{\partial \mathrm{Y}}{\partial \mathrm{C}} = dY * \frac{dg}{dc} = \begin{bmatrix} (out-t)*W1*\acute{g}_{11} \\ (out-t)*W2*\acute{g}_{12} \\ (out-t)*W3*\acute{g}_{21} \\ (out-t)*W4*\acute{g}_{22} \end{bmatrix}$$

[1] With respect to

The vector C comes from convolving the 2 * 2 filter with the input image with stride = 2 and zero padding. We denote elements of this filter like this:

f_{11}	f_{12}
f_{21}	f_{22}

Now, we can rewrite the vector C based on the filter elements and input image pixels:

$$C_{11} = x_{11} * f_{11} + x_{12} * f_{12} + x_{21} * f_{21} + x_{22} * f_{22}$$

$$C_{12} = x_{13} * f_{11} + x_{14} * f_{12} + x_{23} * f_{21} + x_{24} * f_{22}$$

$$C_{21} = x_{31} * f_{11} + x_{32} * f_{12} + x_{41} * f_{21} + x_{42} * f_{22}$$

$$C_{22} = x_{33} * f_{11} + x_{34} * f_{12} + x_{43} * f_{21} + x_{44} * f_{22}$$

And now we can obtain the gradient of loss function w.r.t filter elements:

$$df_{11} = \frac{\partial \mathrm{L}}{\partial f_{11}} = \frac{\partial \mathrm{L}}{\partial C_{11}} * \frac{dC_{11}}{\mathrm{d}f_{11}} + \frac{\partial \mathrm{L}}{\partial C_{12}} * \frac{dC_{12}}{\mathrm{d}f_{11}} + \frac{\partial \mathrm{L}}{\partial C_{21}} * \frac{dC_{21}}{\mathrm{d}f_{11}} + \frac{\partial \mathrm{L}}{\partial C_{22}} * \frac{dC_{22}}{\mathrm{d}f_{11}} =$$
$$(out - t) * \{ W1 * \acute{g}_{11} * x_{11} + W2 * \acute{g}_{12} * x_{13} + W3 * \acute{g}_{21} * x_{31} + W4 * \acute{g}_{22} * x_{33}\}$$

$$df_{12} = \frac{\partial \mathrm{L}}{\partial f_{12}} = \frac{\partial \mathrm{L}}{\partial C_{11}} * \frac{dC_{11}}{\mathrm{d}f_{12}} + \frac{\partial \mathrm{L}}{\partial C_{12}} * \frac{dC_{12}}{\mathrm{d}f_{12}} + \frac{\partial \mathrm{L}}{\partial C_{21}} * \frac{dC_{21}}{\mathrm{d}f_{12}} + \frac{\partial \mathrm{L}}{\partial C_{22}} * \frac{dC_{22}}{\mathrm{d}f_{12}} =$$
$$(out - t) * \{ W1 * \acute{g}_{11} * x_{12} + W2 * \acute{g}_{12} * x_{14} + W3 * \acute{g}_{21} * x_{32} + W4 * \acute{g}_{22} * x_{34}\}$$

$$df_{21} = \frac{\partial \mathrm{L}}{\partial f_{21}} = \frac{\partial \mathrm{L}}{\partial C_{11}} * \frac{dC_{11}}{\mathrm{d}f_{21}} + \frac{\partial \mathrm{L}}{\partial C_{12}} * \frac{dC_{12}}{\mathrm{d}f_{21}} + \frac{\partial \mathrm{L}}{\partial C_{21}} * \frac{dC_{21}}{\mathrm{d}f_{21}} + \frac{\partial \mathrm{L}}{\partial C_{22}} * \frac{dC_{22}}{\mathrm{d}f_{21}} =$$
$$(out - t) * \{ W1 * \acute{g}_{11} * x_{21} + W2 * \acute{g}_{12} * x_{23} + W3 * \acute{g}_{21} * x_{41} + W4 * \acute{g}_{22} * x_{43}\}$$

$$df_{22} = \frac{\partial \mathrm{L}}{\partial f_{22}} = \frac{\partial \mathrm{L}}{\partial C_{11}} * \frac{dC_{11}}{\mathrm{d}f_{22}} + \frac{\partial \mathrm{L}}{\partial C_{12}} * \frac{dC_{12}}{\mathrm{d}f_{22}} + \frac{\partial \mathrm{L}}{\partial C_{21}} * \frac{dC_{21}}{\mathrm{d}f_{22}} + \frac{\partial \mathrm{L}}{\partial C_{22}} * \frac{dC_{22}}{\mathrm{d}f_{22}} =$$
$$(out - t) * \{ W1 * \acute{g}_{11} * x_{22} + W2 * \acute{g}_{12} * x_{24} + W3 * \acute{g}_{21} * x_{42} + W4 * \acute{g}_{22} * x_{44}\}$$

Appendix 1: Execution Procedure

Because the dataset is one google drive, we use colab notebook for this problem. For loading data, we should go to site of the dataset. Then add the shortcut to your google drive directory. Finally you should just run “Load Data” Section in the notebook.

References:

[1] A. Ng, Deep Learning Specialization. The USA, 2020.

[2] S. Lau, “Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning,” 2017. https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1

[3] https://cs231n.github.io/