به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

درس سیستم‌های هوشمند

تمرین شماره سوم

نام و نام خانوادگی:

اشکان جعفری فشارکی

شماره دانشجویی:

۸۱۰۱۹۷۴۸۳

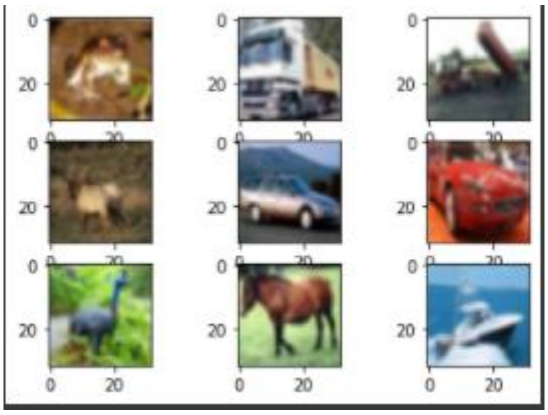
آذر ۱۴۰۰

فهرست سوالات

Question#1

A)

First, we observe the cifar10 data to know more about it!



For this part, I implement some functions which are explained below:

def load_dataset():

In this function we load data and divide them into train and test in 5 to 1 ratio.

def pixel_prep(train, test):

In this function we preprocess our data and rescale them to the range [0,1].

def summary(history):

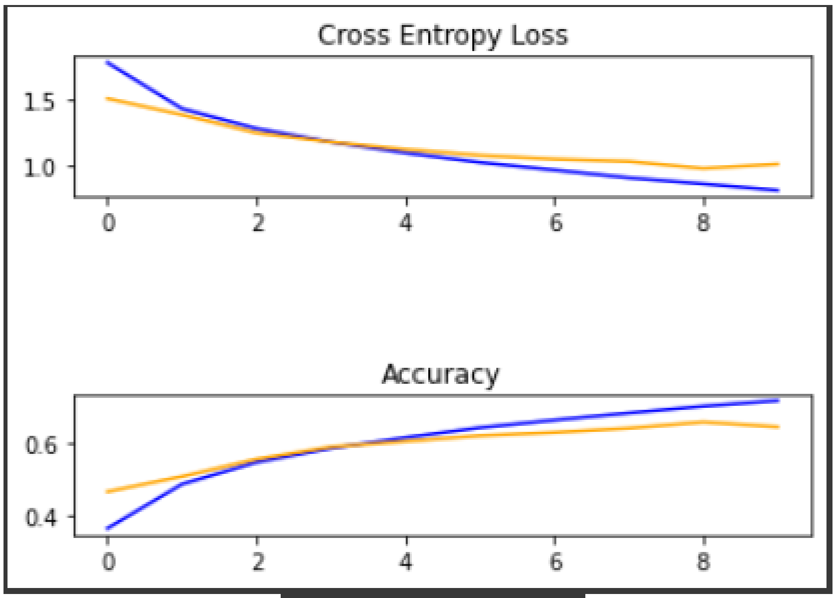
In this function we observe the model's situation by its accuracy and CrossEntropy loss.

def define_model():

In this function, we start to design our CNN by adding layers and MaxPooling and other requirements.

def run_test():

It’s just a function to call above functions and run the CNN.



```
Accuracy = 64.330
```

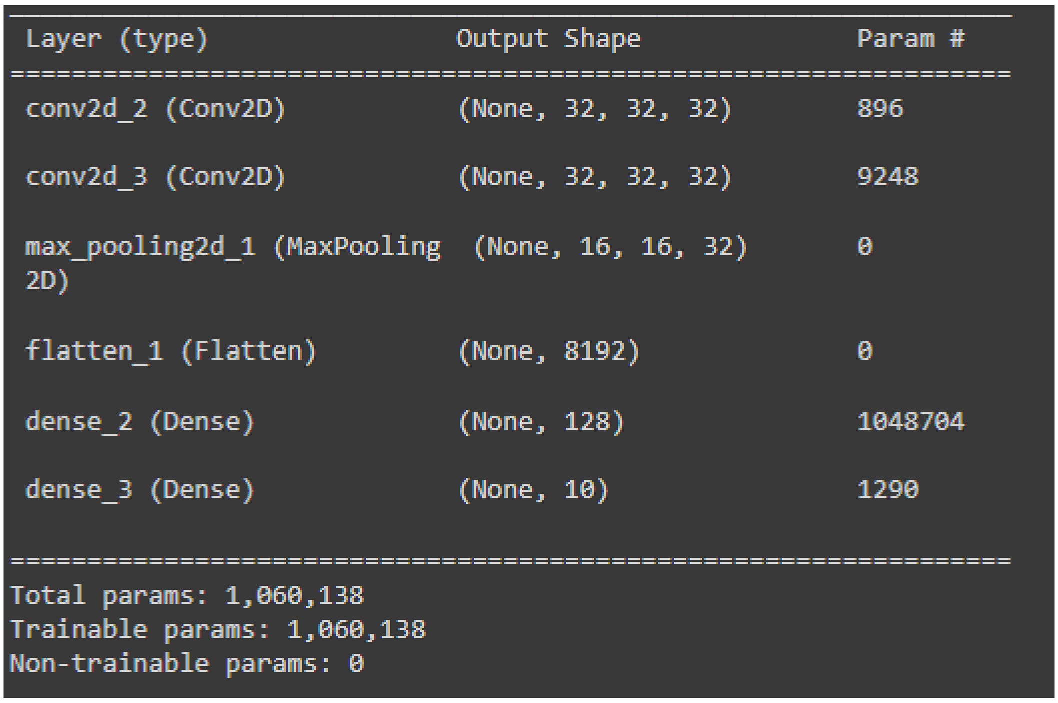
Figure1. Cross Entropy & Accuracy of the 1 Layer CNN for CIFAR10

The preprocess I did before every thing was scaling pixels. As you know pixels color intensity is from 0 to 255 so in order to normalize them, I scaled them to [0,1] range.

The reason that “softmax” should be the activation function is that we want every output to be in [0,1] range and “softmax” as it’s formula (below), normalizes the outputs to their probability distribution.

$$p(y = k|x) = \frac{\exp(s_k)}{\sum_j \exp(s_j)} \quad \text{Softmax function}$$

The parameters and structure of the 1-layer model is as below:



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 32, 32, 32)        896

 conv2d_3 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 flatten_1 (Flatten)         (None, 8192)              0

 dense_2 (Dense)             (None, 128)               1048704

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 1,060,138
Trainable params: 1,060,138
Non-trainable params: 0
```

Figure2. Structure of CNN and Number of parameters

Learning Rate: 0.001

Filters: two 3*3 filters

Maxpooling: 2*2 window

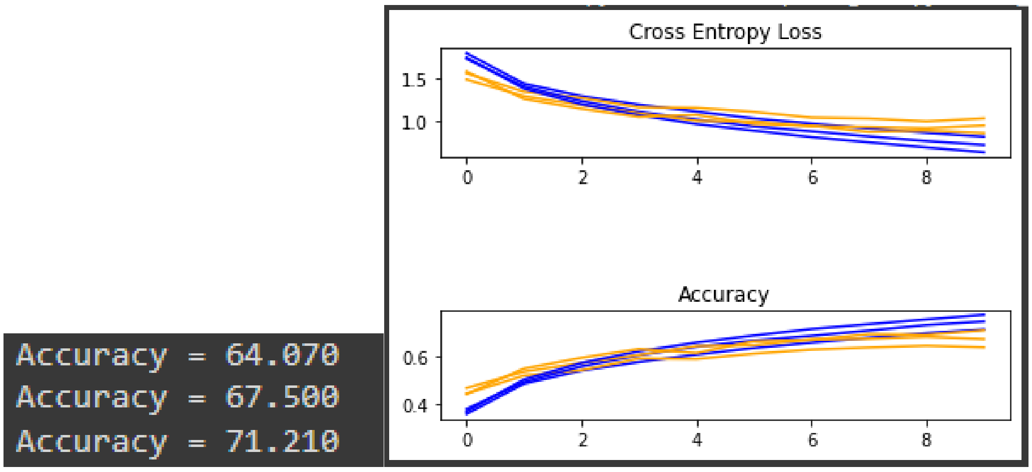
Epoches: 10

Batch-size: 64

B)

We have trained three different models with 1,2 and 3-layer architecture. In all cases, the model was able to learn the training dataset which means showing improvement on the training dataset. This is a good sign and it shows that all three models have sufficient capacity to learn the problem.

The results of the model on the test dataset showed an improvement in accuracy with increase in depth of the model (Figure3). It is possible that this trend would continue if models with four and five layers were evaluated but it should be noticed that the more epoch we train, the more we could get overfitted.

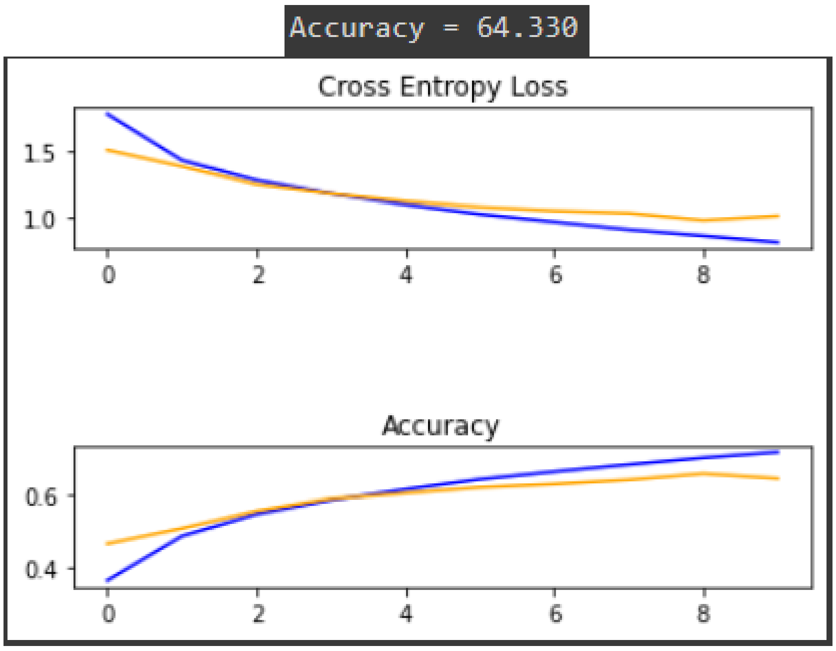


```
Accuracy = 64.070
Accuracy = 67.500
Accuracy = 71.210
```

Figure3. Cross Entropy & Accuracy of the 1,2 and 3-Layer CNN for CIFAR10

C)

In Figure4 and Figure5, we can compare the results of “Tanh” and “ReLU” as activation function in hidden layers and first layer. It should be mentioned that last layer activation function is “softmax”.



```
Accuracy = 64.330
```

Figure4. Cross Entropy & Accuracy of the 1 Layer CNN for CIFAR10 (ReLU)

```
Accuracy = 66.960
```

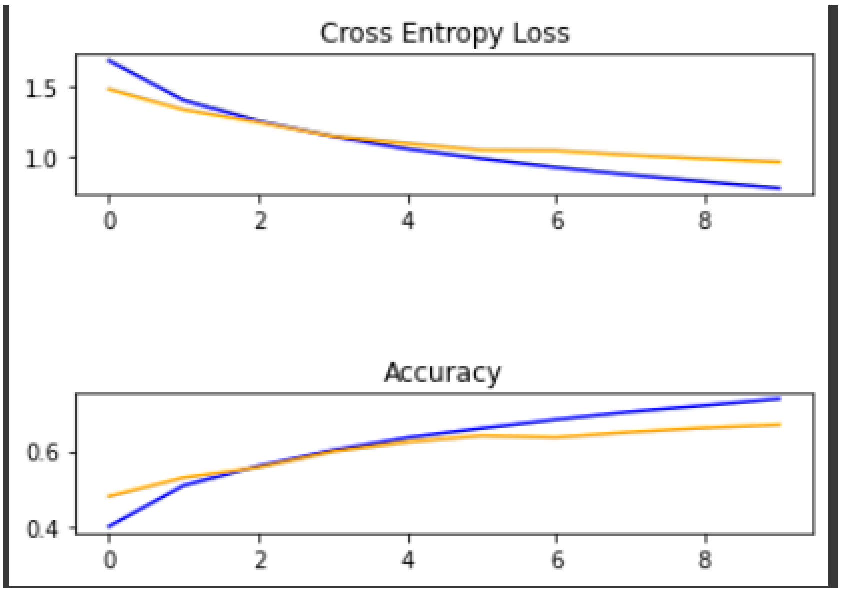


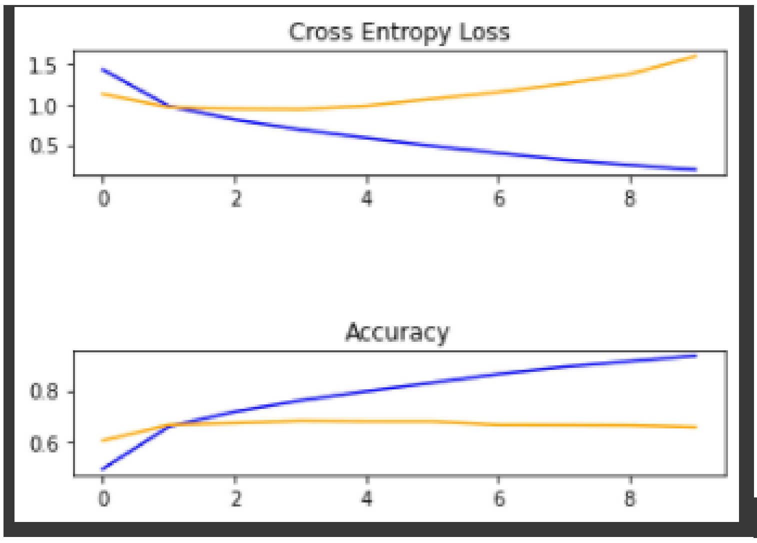
Figure5. Cross Entropy & Accuracy of the 1 Layer CNN for CIFAR10 (Tanh)

As it is shown above “Tanh” give us a better accuracy than “ReLU” in this data but it doesn’t mean that “Tanh” is the better decision because we have other parameters to observe like complexity and time of calculations.

D)

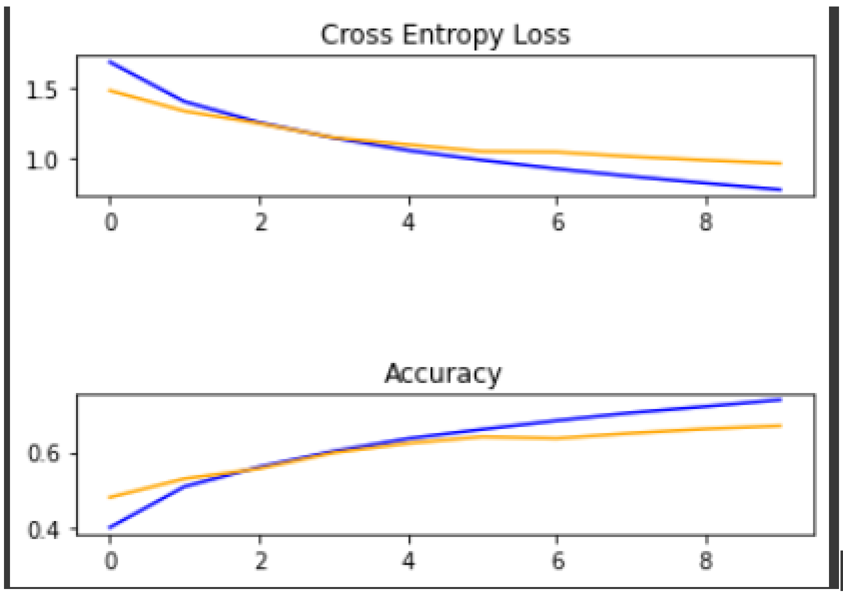
In Figure6 and Figure7, we can compare the result of using “Adam” and “SGD” as optimizer.

It is somehow obvious that SGD perform better than Adam. But we should mention and notice that although Adam has a bit lower accuracy than SGD but it is by far faster than SGD.



```
Accuracy = 65.730
```

Figure5. Cross Entropy & Accuracy of the 1 Layer CNN for CIFAR10 (Adam)



```
Accuracy = 66.960
```

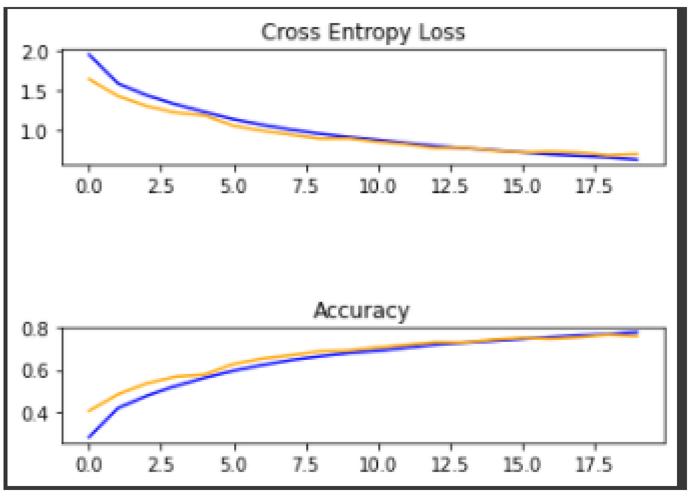
Figure6. Cross Entropy & Accuracy of the 1 Layer CNN for CIFAR10 (SGD)

E)

As we consider and observe above results, they suggest that the model with three layers would be a good baseline model for our investigation. The results also suggest that the model is in need of regularization to address the rapid overfitting of the test dataset.

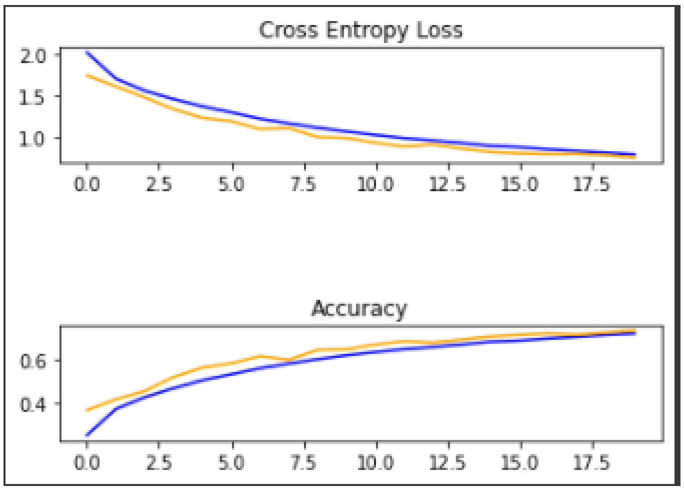
For instance, the results suggest that it may be useful to investigate techniques that slow down the convergence of the model. This may include many techniques. One of them is Dropout regularization.

In this case we use the 3-layer CNN and compare 10%, 20% and 30% dropout. Epochs number will be 20 in order to make results meaningful.



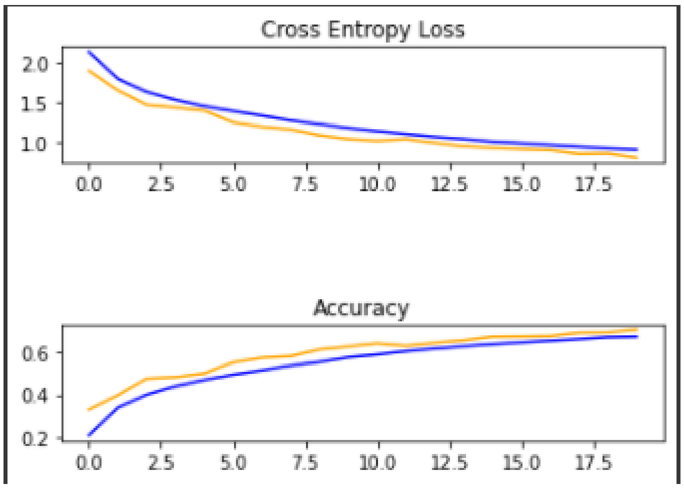
```
Accuracy = 75.740
```

Figure7. Loss & Accuracy of the 3-Layer CNN for CIFAR10 (10% Dropout)



```
Accuracy = 73.830
```

Figure8. Loss & Accuracy of the 3-Layer CNN for CIFAR10 (20% Dropout)



```
Accuracy = 70.630
```

Figure9. Loss & Accuracy of the 3-Layer CNN for CIFAR10 (30% Dropout)

It could be concluded that in 10% dropout we have the highest accuracy as we have more detailed nodes (in other words we have more detailed features).

Question 2

A)

For simplicity I use MATLAB command window and work space for calculations and saving parameters:

$$\begin{cases} \mathcal{L} = \frac{1}{2}(\hat{y}-y)^2 \\ \eta = 0.1 \end{cases} , \; 483: \begin{cases} a = 3 \\ b = 8 \\ y = 4 \end{cases}$$

$$w_1 = \begin{pmatrix} 1.88 & -0.13 & 0.8 \\ 0.28 & -1.3 & 0.83 \end{pmatrix}, b_1 = \begin{pmatrix} 0 \\ 0.81 \\ 1.3 \end{pmatrix}$$

$$w_2 = \begin{pmatrix} 1.35 & -0.58 & 0.83 \end{pmatrix}, \; b_2 = 0.1$$

$$w_3 = \begin{pmatrix} -0.3 \\ 1.8 \end{pmatrix}, \; b_3 = 0.1$$

$$x = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \; ,$$

$$\begin{cases} \hat{y} = \mathrm{ReLU}\left(\sum_{j=1}^{3} w_2(1,j) z_j + b_2\right) + \sum_{i=1}^{2} w_3(i,1) x_i + b_3 \\ \begin{cases} z_j = \tanh\left(\sum_{i=1}^{2} w_1(i,j)\, x_i + b_1(j)\right) \\ j = 1,2,3 \end{cases} \end{cases}$$

$$\frac{\partial L}{\partial \hat{y}} = \hat{y} - y \quad , \quad \frac{\partial \hat{y}}{\partial w_2(1,j)} = \begin{cases} \emptyset & \sum w_2 z + b_2 > \emptyset \\ z_j & \sum w_2 z + b_2 \leqslant \emptyset \end{cases}$$

$$\frac{\partial \hat{y}}{\partial b_2} = \begin{cases} \emptyset & \sum w_2 z_j + b_2 \leqslant \emptyset \\ 1 & \sum w_2 z_j + b_2 > \emptyset \end{cases}$$

$$\frac{\partial \hat{y}}{\partial w_{3(i,1)}} = x_i \quad , \quad \frac{\partial \hat{y}}{\partial b_3} = \underline{\underline{1}}$$

First Level:

$$j = \begin{array}{c|l} 1 & z_1 = \tanh(3.6) \\ 2 & z_2 = \tanh(-3.35) \\ 3 & z_3 = \tanh(5.39) \end{array}$$

$$\hat{y} = \text{ReLU}\big(\overbrace{\sum w_2(1,j)\, z_j + \emptyset.1}^{11.37}\big)$$

$$+ \sum w_3(i,1)\, x_i + \emptyset.1$$

$$\Rightarrow \hat{y} = \text{ReLU}(11.37) + 4.8 + \emptyset.1$$
$$= 16.27$$

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = 16.27 - 4 = 12.27 \quad , \quad \frac{\partial \hat{y}}{\partial b_2} = 1 , \quad \frac{\partial \hat{y}}{\partial w_2} = z_j$$

$$\frac{\partial z_j}{\partial w_1(i,j)} = x_i (1 - z_j)^2 , \quad \frac{\partial z_j}{\partial b_1(j)} = 1 - z_j^2$$

$$\frac{\partial \hat{y}}{\partial z_j} = \begin{cases} w_2(1,j) & \text{o.w.} \\ \emptyset & \sum_{j=1}^{3} w_2(1,j) z_j + b_2 \leqslant 0 \end{cases}$$

$$\frac{\partial \mathcal{L}}{\partial w_3} = 12.27 \begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad , \quad \left\{ \begin{array}{l} \dfrac{\partial \mathcal{L}}{\partial b_2} = 12.27 \\ \\ \dfrac{\partial \mathcal{L}}{\partial b_3} = 12\;\;7 \end{array} \right.$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = 12.27 \begin{pmatrix} 3.6 \\ -3.35 \\ 5.39 \end{pmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial b_1(j)} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_j} \cdot \frac{\partial z_j}{\partial b_1(j)} \quad , m = 1,2,3$$

$$\frac{\partial L}{\partial b_1(j)} = \begin{pmatrix} -198.11 \\ 72.74 \\ -285 \end{pmatrix}$$

$$\frac{\partial L}{\partial w_1(1,1)} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1(1,1)}$$

$$\Rightarrow \frac{\partial L}{\partial w_1(1,j)} = (12.27)\,(w_{2j})\,(\,''\,) = 2\begin{pmatrix} -198.11 \\ 72.74 \\ -285 \end{pmatrix}$$

$$\Rightarrow \frac{\partial L}{\partial w_1(2,j)} = 3\begin{pmatrix} -198.11 \\ 72.74 \\ -285 \end{pmatrix}$$

$$\longrightarrow w_3 = \begin{pmatrix} -0.3 \\ 1.8 \end{pmatrix} - 0.1 \times 12.27 \times \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

$$\longrightarrow \begin{cases} b_3 = 0.1 - 0.1(12.27) \\ b_2 = 0.1 - 0.1(12.27) \end{cases}$$

$$\longrightarrow W_2 = \begin{pmatrix} 1.35 & -5.8 & 0.83 \end{pmatrix} - 0.1 \times 12.27 \times \begin{pmatrix} 3.6 \\ -3.35 \\ 5.39 \end{pmatrix}^T$$

$$\longrightarrow b_1 = \begin{pmatrix} 0 \\ 0.81 \\ 1.3 \end{pmatrix} - 0.1 \begin{pmatrix} -198.11 \\ 72.74 \\ -285 \end{pmatrix}$$

$$\longrightarrow w_1 = \begin{pmatrix} 1.38 & -0.13 & 0.8 \\ 0.28 & -1.3 & 0.83 \end{pmatrix}$$

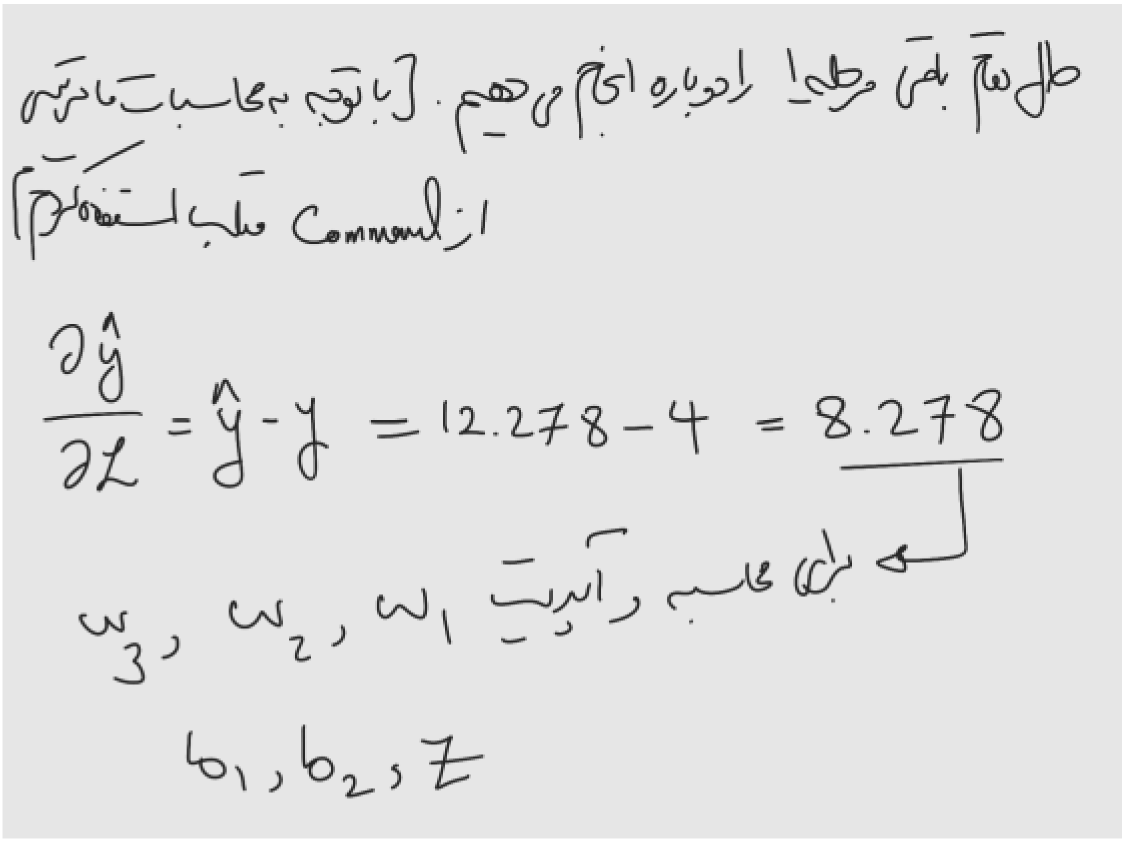
$$- 0.1 \times \underbrace{\begin{pmatrix} 2 \\ 3 \end{pmatrix}}_{2\times1} \underbrace{\begin{pmatrix} -198.11 & 72.74 & -285 \end{pmatrix}}_{1\times3}$$

Next

$$Z = \begin{pmatrix} -273.75 \\ -105.18 \\ 404 \end{pmatrix}$$

$$\hat{y} = ReLU(-1.8\times10^3) \simeq 12.278$$

همانطور که مشاهده می‌شود داریم به 4 نزدیک می‌شویم !



حال سه تا بخش مرحله ۱ را دوباره انجام می‌دهیم. [با توجه به محاسبات ماتریسی از Command متلب استفاده کردم]

$$\frac{\partial \hat{y}}{\partial L} = \hat{y} - y = 12.278 - 4 = \underline{8.278}$$

برای محاسبه گرادینت w_1, w_2, w_3, b_1, b_2, Z

B)

B1)

L1-Loss: This Function is used to minimize the error which is the sum of the all the absolute differences between the true value and the predicted value.

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

L2-Loss: This Function is used to minimize the error which is the sum of the all the squared differences between the true value and the predicted value.

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

Briefly, L2 Loss Function is preferred in most of the cases. But when the outliers are present in the dataset, then the L2 Loss Function does not perform well. The reason behind this bad performance is that if the dataset is having outliers, then because of the consideration of the squared differences, it leads to the much larger error. Hence, L2 Loss Function is not useful here. Prefer L1 Loss Function as it is not affected by the outliers or remove the outliers and then use L2 Loss Function.

Huber-Loss: Huber loss is less sensitive to outliers in data than mean squared error. Definition is as below:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$

The Huber function is less sensitive to small errors than the L1 norm, but becomes linear for large errors. To visualize this, notice that function $|\cdot|$ accentuates points near to the origin as compared to Huber (which would in fact be quadratic in this region). Therefore, the Huber loss is preferred to the L1 in certain cases for which there are both large outliers as well as small (ideally Gaussian) perturbations.

Briefly, as defined above, the Huber loss function is strongly convex in a uniform neighborhood of its minimum; at the boundary of this uniform neighborhood, the Huber loss function has a differentiable extension to an affine function. These properties allow it to combine much of the sensitivity of the mean-unbiased, minimum-variance estimator of the mean (using the quadratic loss function) and the robustness of the median-unbiased estimator (using the absolute value function).

B2)

There could be two possibilities to let this situation happen.

1) Model is implemented well and it is working very well which makes close accuracy in train and validation data.
2) Validation Data have less complexity than Train Data and this situation makes Validation Data accuracy as well as Train Data accuracy.

B3)

Adding momentum to SGD is a method that helps us speed up in proper descanting and also helps us lowering swings. This method is applied by adding a factor of previous step. If we show this factor by α, then the formula is as below:

$$\Delta w_{CURRENT} = -\eta \frac{\partial SSE}{\partial w_{CURRENT}} + \alpha \, \Delta w_{PREVIOUS}$$

"α" is usually about 0.9 and should be in [0,1] range. The second which added to normal SGD makes it to be faster and get converged better. For visualized algorithm you could see figure below:

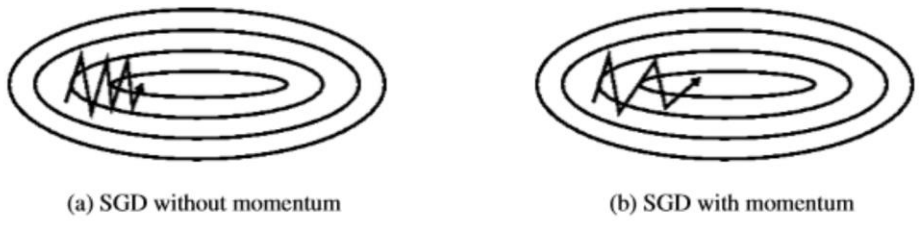


Figure9. Comparing normal SGD with SGD using momentum

C)

Before everything, it should be mentioned that because of random_numbers the model should be fitted several times to give the best result!

C1)

All requirements in this part are done by the functions below:

```
Functions:

def generating(on,num_data):
def sin_data_generating(x_train):
def splitting(x_values, y_values):
def sin(x1, x2):

Commands:

x_values, y_values=generating(on=1,num_data = 10000)
x_train, y_train, x_validation, y_validation, x_test, y_test
 = splitting(x_values, y_values)
```

C2)

In this part, I've just implemented a normalizing function:

```
def normalizing(train,valid,test,range,min_data):
```

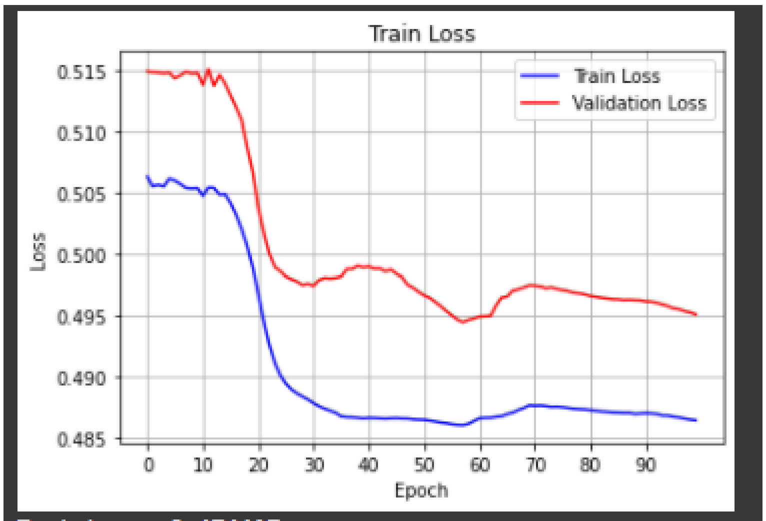
```
Commands:
x_train, x_validation, x_test  = normalizing(x_train, x_vali
dation, x_test, np.max(x_train) - np.min(x_train) , np.min(x
_train))
```

C3)

```
Functions of "class Multi_Layer_Perceptron"
     ReLU
     train
     Forward
     Backward
     cost
     expected

def L2_Loss(y1, y2): Second Norm Function
```

The Loss of train and validation data is as below for sin(x1+x2):



```
Test Loss= 0.474417
```

Figure11. Test Loss and Loss per epoch for Train and Validation Loss

C4)

The Loss of train and validation data is as below for sin(x):

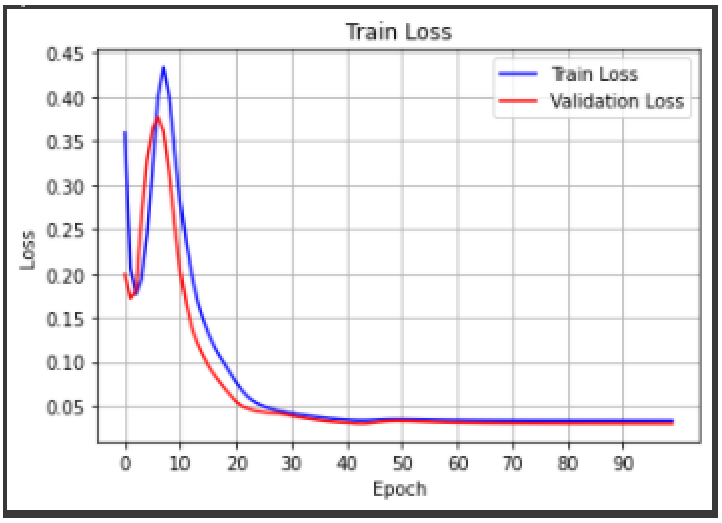
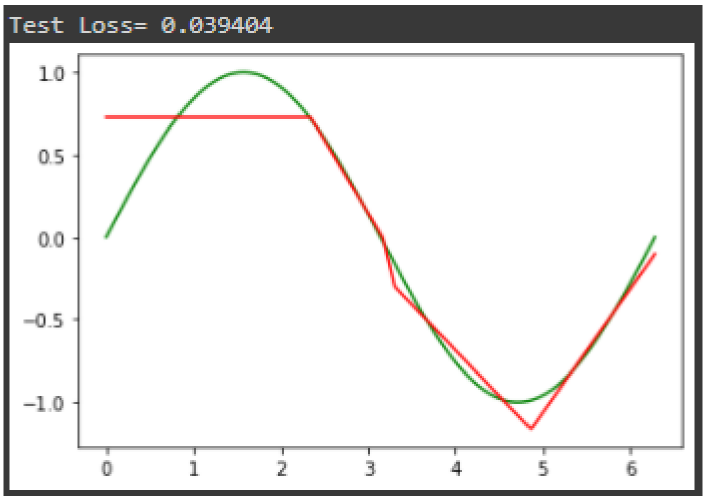


Figure12. Test Loss and Loss per epoch for Train and Validation Loss

The model prediction and real sin (x) is plotted simultaneously as below. **It should be mentioned that because of random_numbers the model should be fitted several times to give the best result!**



```
Test Loss= 0.039404
```

Figure13. Sin via Model predicted Sin

References:

1) Links of Question1!
2) https://towardsdatascience.com/building-neural-network-from-scratch-9c88535bf8e9
3) Andrew NG - Deep learning MIT CA