Cyber Labs

ML BOOTCAMP PROJECT REPORT

Winter of Code 5.0

SHUBHAM CHAUHAN

IIT(ISM) Dhanbad

About Me:-

Name : Shubham Chauhan

Place : Meerut, Uttar Pradesh

Branch : Integrated Master of technology in Mathematics and Computing (MnC).

Year : 1st Year

DOB : 21/03/2005

Admission No. : 22je0934

Phone no. : 7830003219

Institute Email id : 22je0934@iitism.ac.in

Email Id : shubhampersonal22222@gmail.com

LinkedIn profile: https://www.linkedin.com/in/shubham-chauhan-773317255/

Git Hub : ShubhamChauhan22222

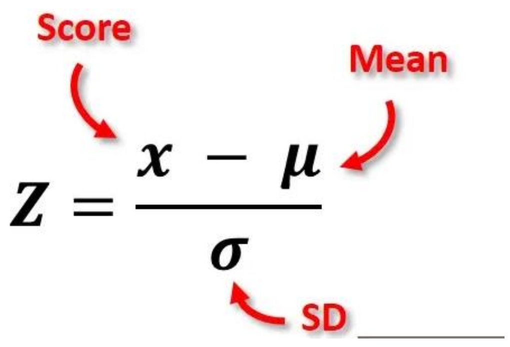
Week 1

Linear regression

- Vectors used in place of loops makes it many times faster and short codes with much more efficiency.
- Trained model without normalization which takes much time due to the low value of learning rate and high no. of iterations.
- **Input data used = linear_train**
- **Label prediction of linear_test_data saved to linear_test_data_pred**
- **Model function =>**

$$f(x^i) = wx^i + b$$

- Normalization used = z- score normalization



$$Z = \frac{x - \mu}{\sigma}$$

- **Cost function used = mean squared error(MSE).**

$$\frac{1}{2m}\sum_{1}^{m}(h(x^{(i)})-y^{(i)})^2$$

- Gradients:

dj_dw = $(1/m)\sum(f(w,b) - y^i)x^i$

dj_db = $(1/m)\sum(f(w,b) - y^i)$

- Gradient descent:

w = w - alpha * dj_dw

b = b - alpha * dj_db

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \|y(i) - \hat{y}(i)\|^2}{N}},$$

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \overline{y})^2}$$

➢ Training logs

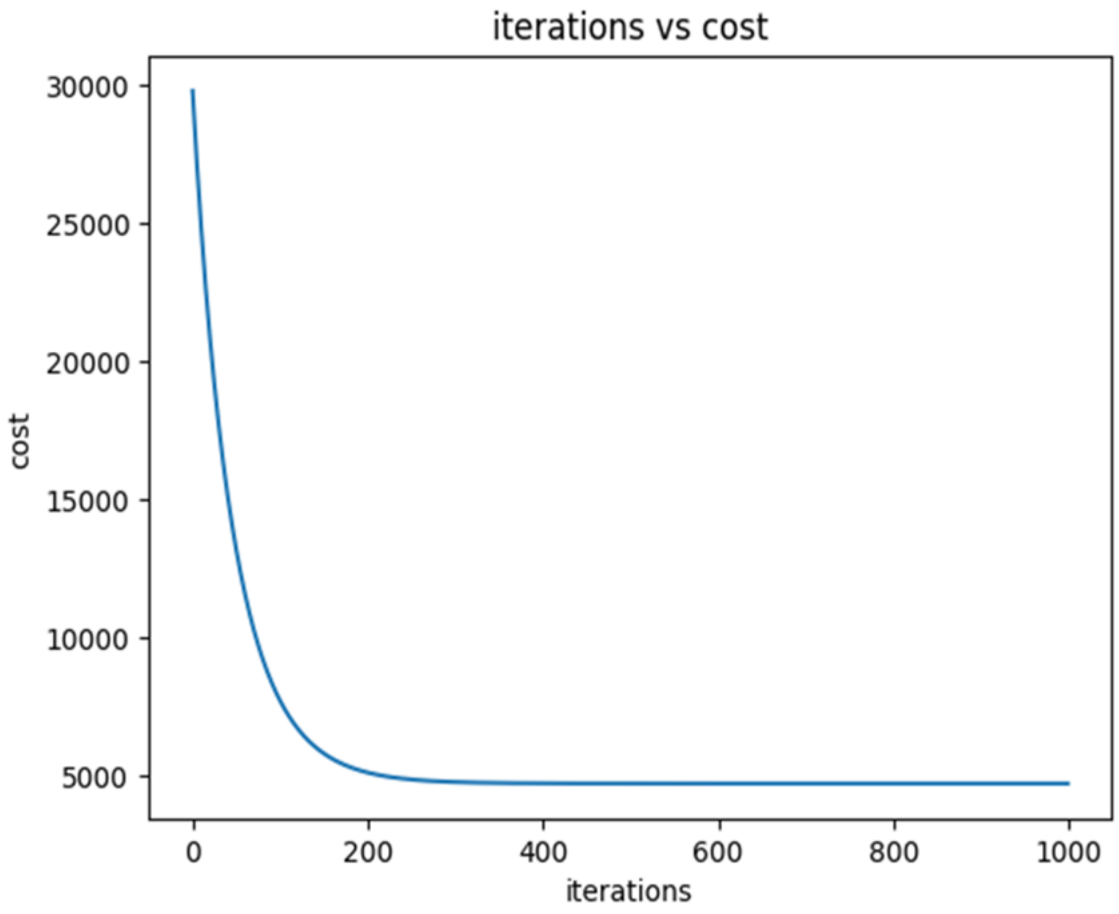
- **TIME taken by model for training data = negligible.**
- **MSE= 4769.768749133481**
- **RMSE = [97.67055594]**
- **R2 Score = [0.84287832]**

-
  ```
  Iteration    0: Cost 29785.03
  Iteration  100: Cost  7772.89
  Iteration  200: Cost  5168.19
  Iteration  300: Cost  4823.44
  Iteration  400: Cost  4777.02
  Iteration  500: Cost  4770.75
  Iteration  600: Cost  4769.90
  Iteration  700: Cost  4769.79
  Iteration  800: Cost  4769.77
  Iteration  900: Cost  4769.77
  ```

➢ Hyperparameters

- **Weights initialization = zeros**
- **Bias initialization = zero**
- **Learning rate(alpha) = 0.01**
- **Iterations = 1000**

➢ Training Visualization



Polynomial regression

The data with a total length of 50,000 is first split into two sets. The train set with 30,000 training examples and the cross validation(CV) set with 20,000 training examples.

Degree 5 polynomial is chosen for training the data because it gives the satisfactory R2 score for both train and CV. And also the cost vs degree curve also gives minima for CV set.

- Model function used

  $$F(x^i)=\sum w_i x_i + b$$

  Where x_i are the elements upto 5 degree polynomial which are total 55 in number.
- Normalization, Cost function, gradient and gradient descent are similar to the linear regression.

➢ Training logs

- **TIME taken by model for training data = 1 minute.**
- For train data set:

```
MSE  = 2216.83
 RMSE = [66.58576333]
 R2 Score = [1.]
```

- **For cross validation set:**

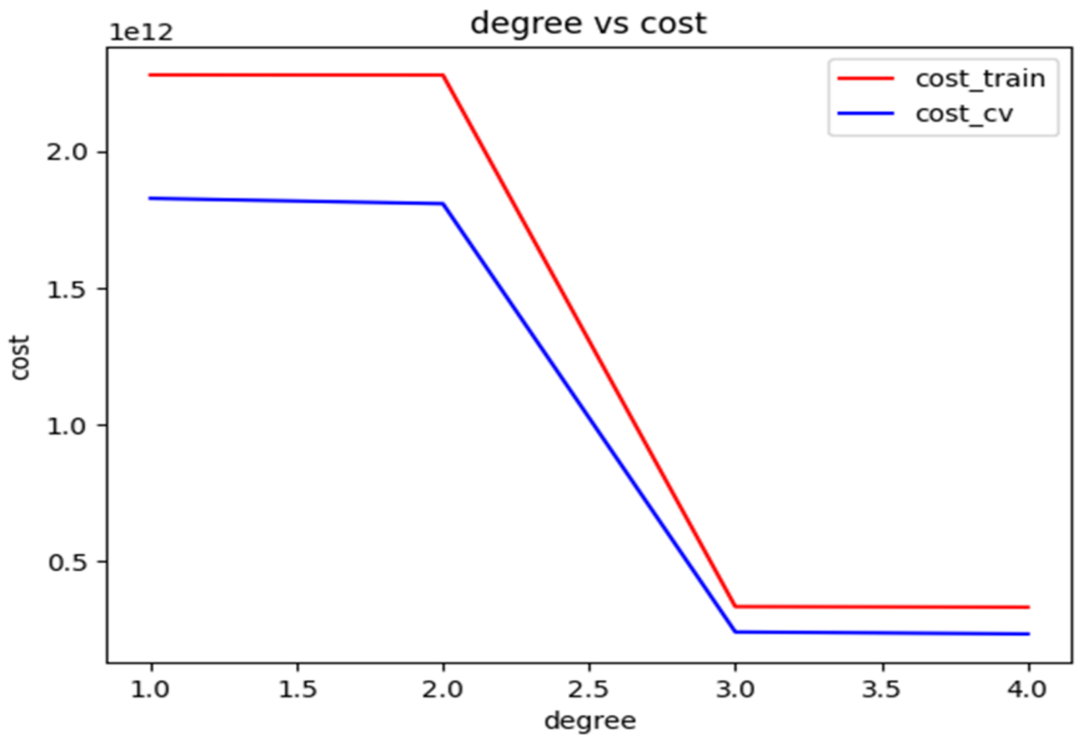
```
RMSE = [36912.55]
R2 Score = [0.99972172]
```

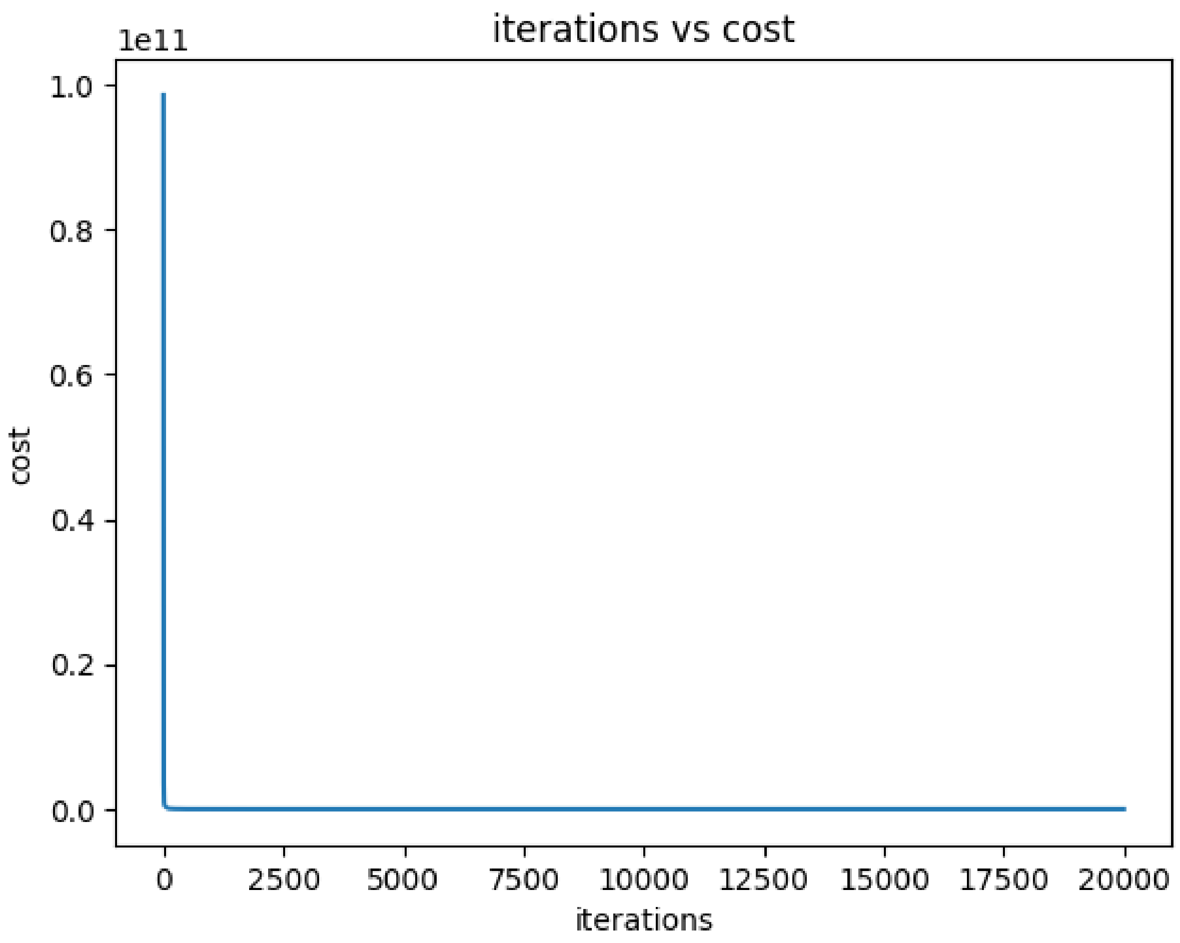
```
• Iteration    0: Cost 98588404131.61
• Iteration 2000: Cost 569054.88
• Iteration 4000: Cost 134569.88
• Iteration 6000: Cost 45830.70
• Iteration 8000: Cost 22879.88
• Iteration 10000: Cost 14162.41
• Iteration 12000: Cost  9501.68
• Iteration 14000: Cost  6544.79
• Iteration 16000: Cost  4548.63
• Iteration 18000: Cost  3172.48
```

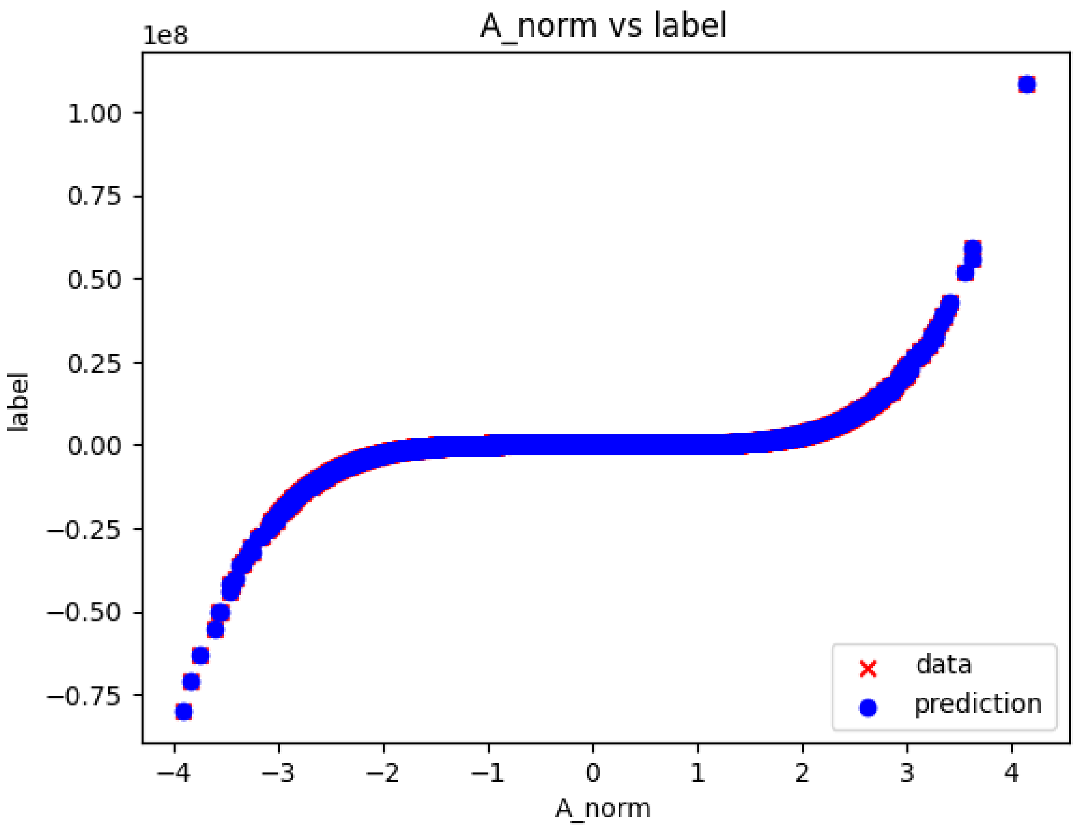
➢ Hyperparameters

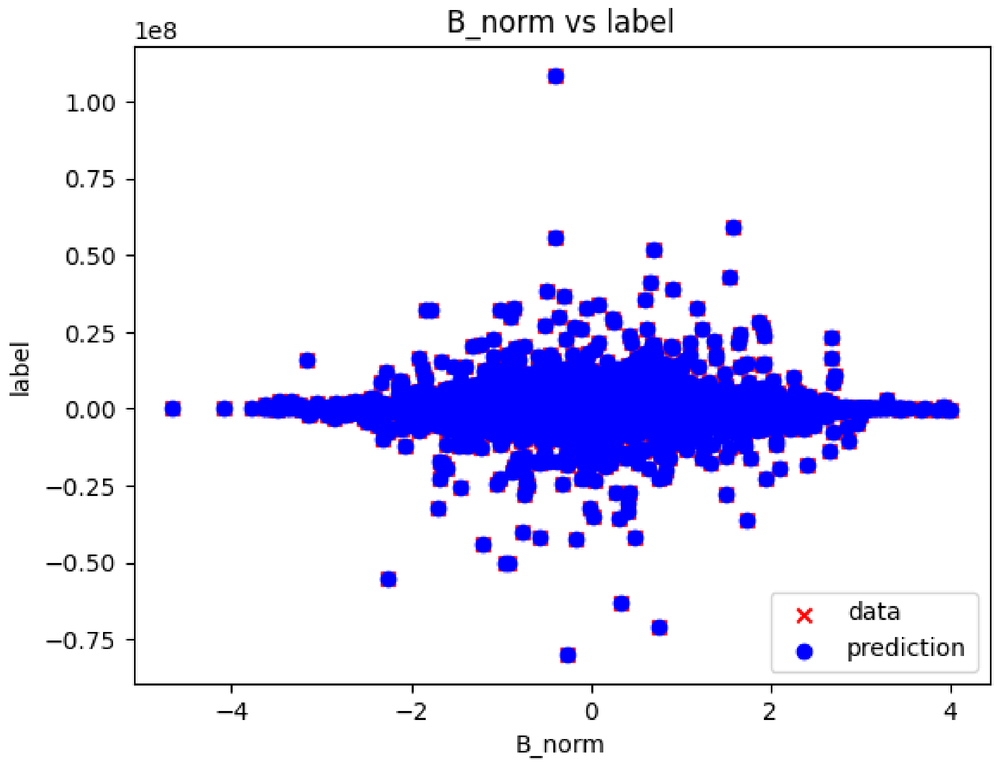
- **Weights initialization = zeros**
- **Bias initialization = zero**
- **Learning rate(alpha) = 0.001**
- **Iterations = 20,000**

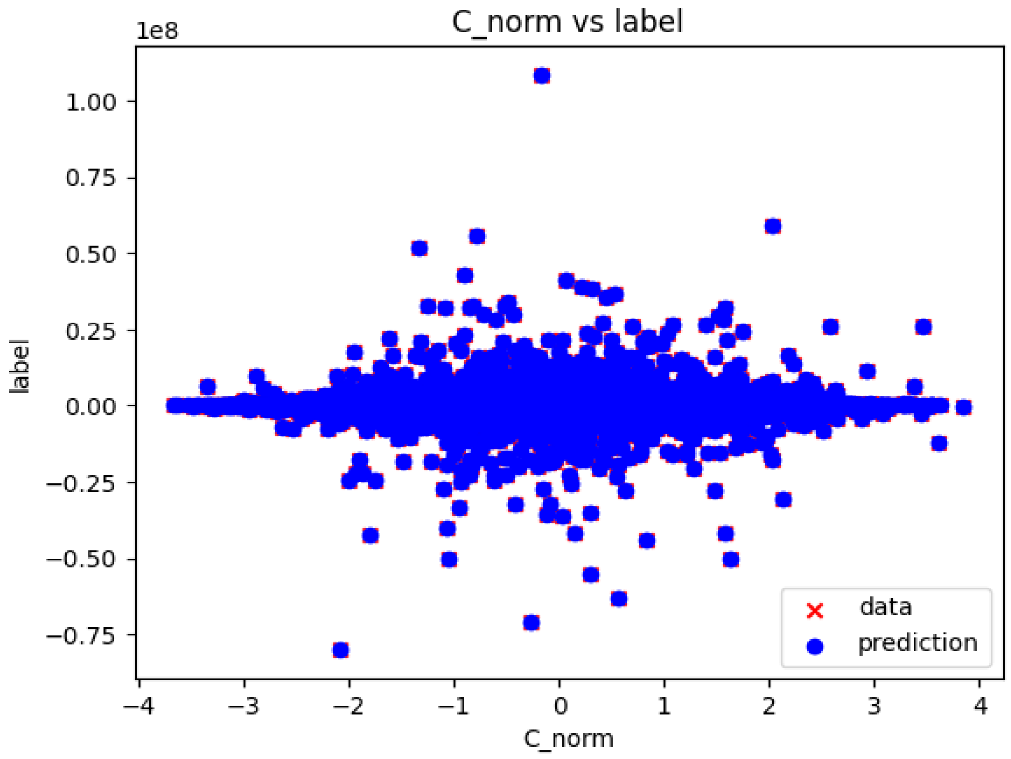
➢ Training Visualization




iterations vs cost
1e11
1.0
0.8
0.6
0.4
0.2
0.0
cost
0
2500
5000
7500
10000
12500
15000
17500
20000
iterations


A_norm vs label
1e8
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
label
−4
−3
−2
−1
0
1
2
3
4
A_norm
data
prediction


B_norm vs label
1e8
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
label
−4
−2
0
2
4
B_norm
data
prediction


C_norm vs label
1e8
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
label
−4
−3
−2
−1
0
1
2
3
4
C_norm
data
prediction

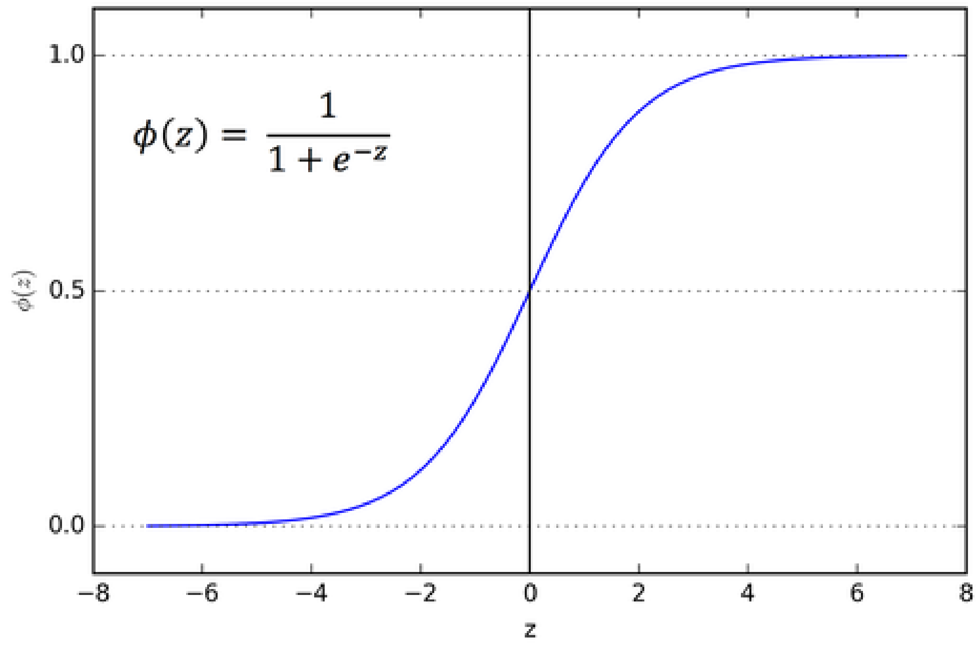
Week 2

Logistic regression

First, I used softmax activation function for classification because it can give multiple probability outputs. But then I get to know that logistic regression must have sigmoid function to transforms its output to return a probability value. Then I tried and made sigmoid logistic regression.

I found out that softmax regression is much more efficient than sigmoid logistic regression on the same training set. Softmax takes less time and gives more accuracy than sigmoid.

Sigmoid function:



I had split the data of 30,000 training examples into train with 20,000 dataset and test with 10,000.

Then I convert my output vector y into one hot encoded Y. One hot encoding gives the particular item a value of 1 and others 0 for a particular training example. One-hot encoding is basically a probability but the difference is that it is an absolute probability because we already know the output of the training example.

I had also added a 1's row at 0 index to remove the need of extra separate bias 'b' in the z-normalized X(input data).

- **Model function** :

  F(X)= sigmoid(X.theta)

- **Cost function:**

  -(1/m) * (sum(Y*log(f(x)) + (1-Y)*log(1-f(x)))

- **Gradient descent:**

  (1/m) * np.dot(X.T, f(x) - Y)

- **Gradient:**

  theta= theta - eta * gradient(theta, X, Y)

➢ Training logs

- Time taken by model for training the data: 1 minute.
- Accuracy of train data = 81.17%.

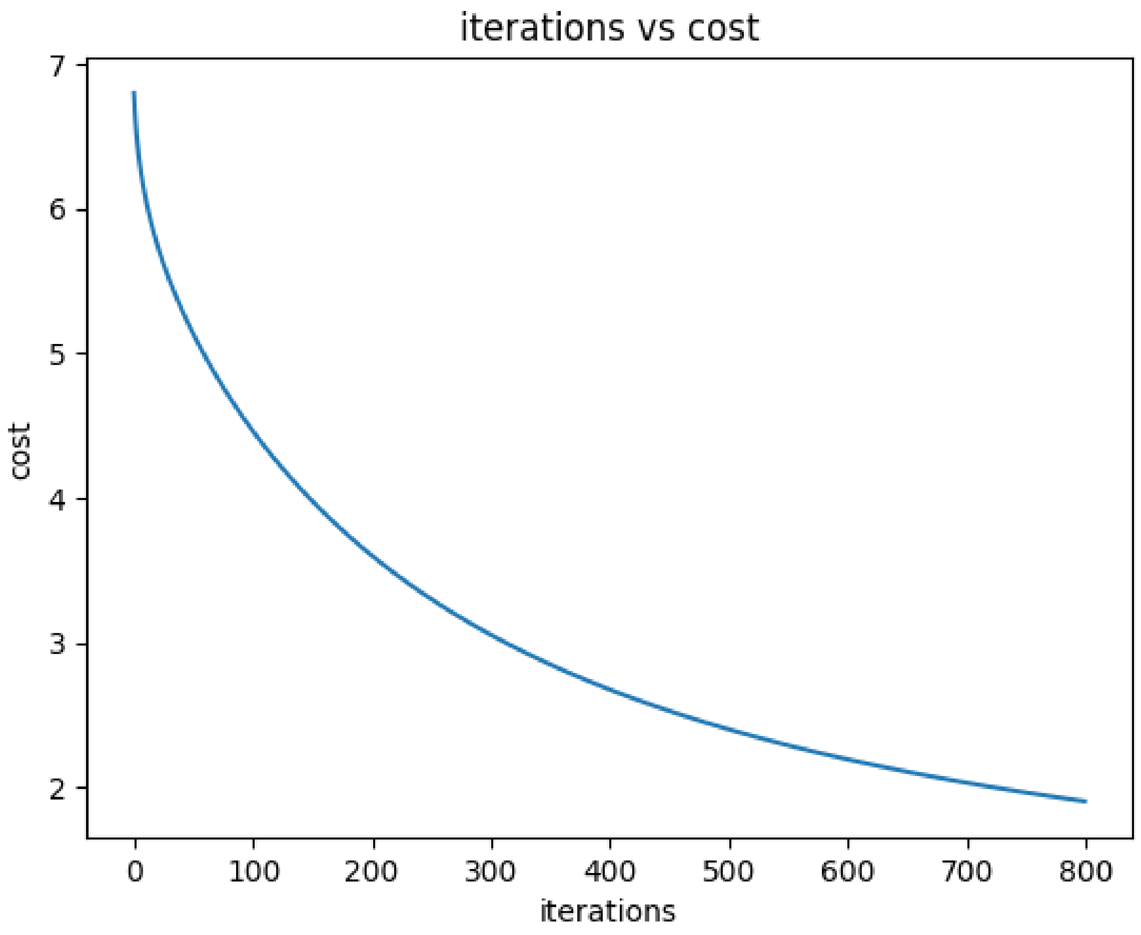
- Accuracy of test data = 80.47%.
- Input data = classification_train.
- Labels predicted of classification_test saved to classification_test_pred.

```
o  Cost after 0 iterations is : 6.795872091517176
o  Cost after 80 iterations is : 4.698562919905196
o  Cost after 160 iterations is : 3.893998688922301
o  Cost after 240 iterations is : 3.3553306716388005
o  Cost after 320 iterations is : 2.9673736315167276
o  Cost after 400 iterations is : 2.6758177848107096
o  Cost after 480 iterations is : 2.449798942727746
o  Cost after 560 iterations is : 2.270198929184516
o  Cost after 640 iterations is : 2.12451974322447
o  Cost after 720 iterations is : 2.0042729877193923
```

➢ Hyperparameters

- **Weights initialization :**
  theta = (np.random.randn(785, 10))*0.01.
- **Learning rate(alpha) = 0.01**
- **Iterations = 8,00**

➢ Training Visualization



Week 3

K-Nearest Neighbour

What is KNN?

KNN basically classifies the new data points based on the similarity with the earlier stored data.

This algorithm basically works in three basic steps:

1. **Calculating Distances:**

Distance functions

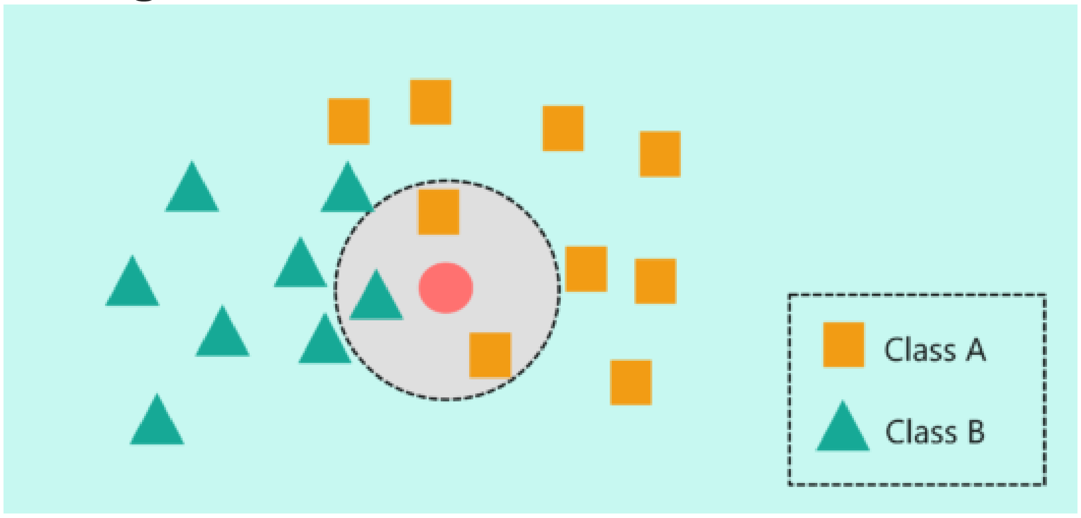
Euclidean $$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan $$\sum_{i=1}^{k}\left|x_i - y_i\right|$$

Minkowski $$\left(\sum_{i=1}^{k}(\left|x_i - y_i\right|)^q\right)^{1/q}$$

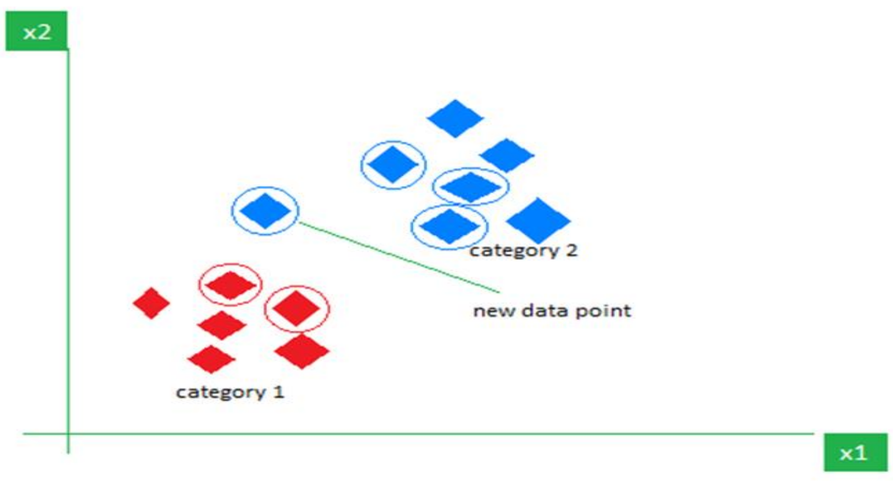
I had used Euclidean distance to calculate the distances of each test data from every given datapoints and store them in matrix.

2. Sorting first K distances:



Sorting the calculated distances of each test data in ascending order and taking out the first k distances.

3. Finding the most frequent category:



The most frequently occurred category will be the category of the particular training example.

I had split the data of 30,000 length into known datapoints of 20,000 and test data points of length 10,000.

Training logs

- Time taken : aprox. 10 minutes.
- Accuracy = aprox. 81.05%
- Input data = classification_train.
- Distance formula used = Euclidean.

Hyperparameter

- K = 60

Week 4

Neural Network

I had split the data of 30,000 training examples into train with 20,000 dataset and test with 10,000.

I had pre processed the data by normalization of input data(X) and then one hot encoding of output data(Y).

I had first made the hardcore 1 hidden layer neural network from scratch.

1 hidden layer neural network

- Initialize Parameters:

```
w1 = np.random.randn(n_1, n)*0.01
 b1 = np.zeros((n_1, 1))
 w2 = np.random.randn(n_2, n_1)*0.01
 b2 = np.zeros((n_2, 1))
```

- Forward propagation:

```
z1 = np.dot(w1, x) + b1
a1 = relu(z1)
z2 = np.dot(w2, a1) + b2
a2 = softmax(z2)
```

- Cost Function:

  -(1/m)*np.sum(y*np.log(a2))

- Backpropagation:

  ```
  dz2 = (a2 - y)
  dw2 = (1/m)*np.dot(dz2, a1.T)
  db2 = (1/m)*np.sum(dz2, axis = 1)


  dz1 = (1/m)*np.dot(w2.T, dz2)*derivative_relu(z1)
  dw1 = (1/m)*np.dot(dz1, x.T)
  db1 = (1/m)*np.sum(dz1, axis = 1)
  ```

- Gradient descent:

  ```
  w1 = w1 - learning_rate*dw1
  b1 = b1 - learning_rate*db1
  w2 = w2 - learning_rate*dw2
  b2 = b2 - learning_rate*db2
  ```

➢ Training logs

- Time taken : aprox. 10 minutes.
- Accuracy of test data = 80%
- Input data = classification_train.

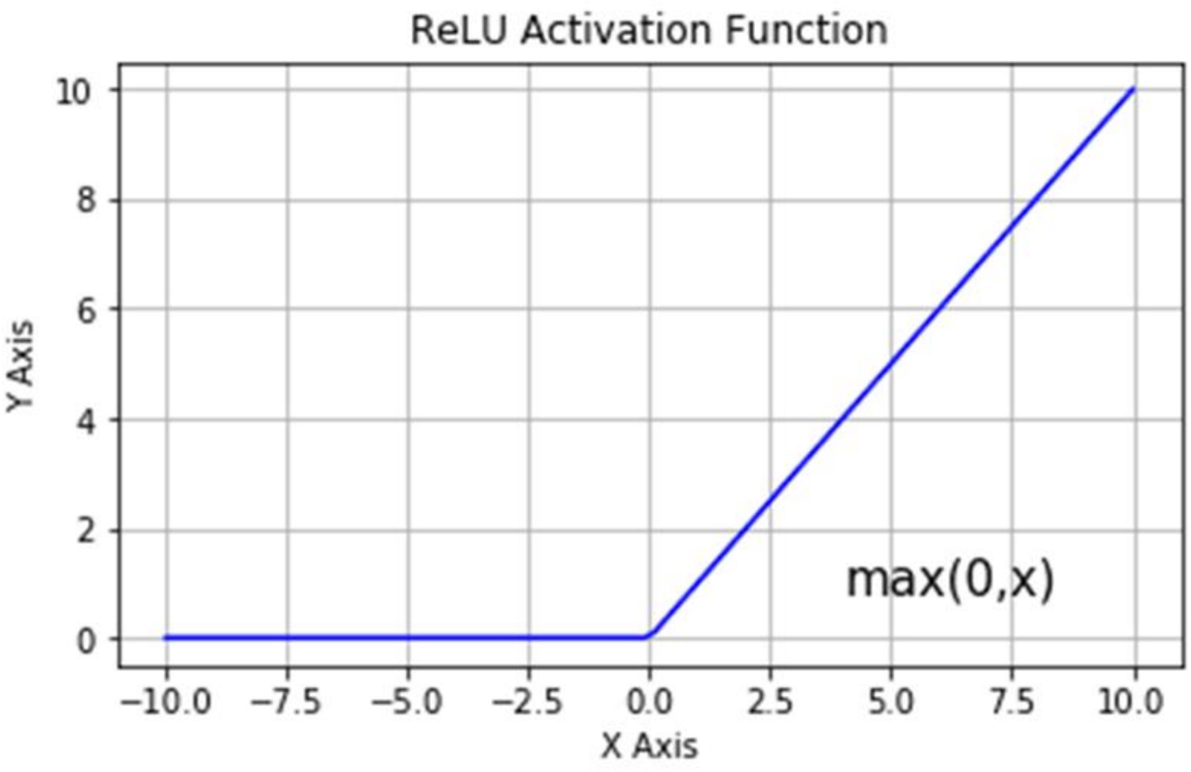
```
o  Cost after 0 iterations is : 2.298846476105506
o  Cost after 100 iterations is : 1.1790628011590636
o  Cost after 200 iterations is : 0.9468607449084548
o  Cost after 300 iterations is : 0.843516559373537
o  Cost after 400 iterations is : 0.7821140110313455
o  Cost after 500 iterations is : 0.740070439701624
o  Cost after 600 iterations is : 0.7087746089107296
o  Cost after 700 iterations is : 0.6841752914983846
o  Cost after 800 iterations is : 0.6640908278022282
o  Cost after 900 iterations is : 0.6472294650566788
```

➢ Hyperparameters

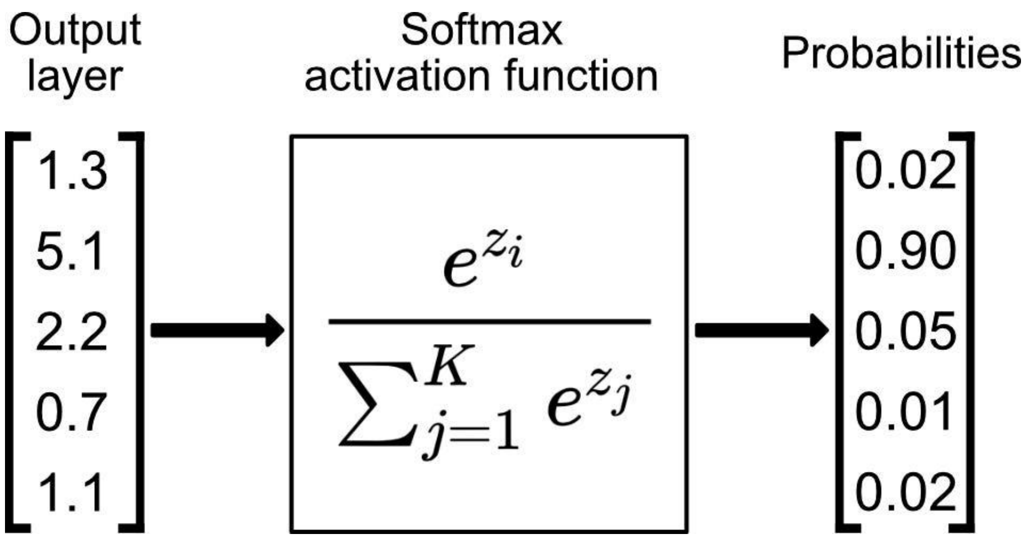
- iterations = 1000
- learning_rate = 0.04
- No. of layers = 2 (1 hidden and 1output layer).
- No. of neurons in each layer:

  Hidden layer = 1000

  Output layer = 10
- Activation functions in each layer:
- Hidden layer = Relu.

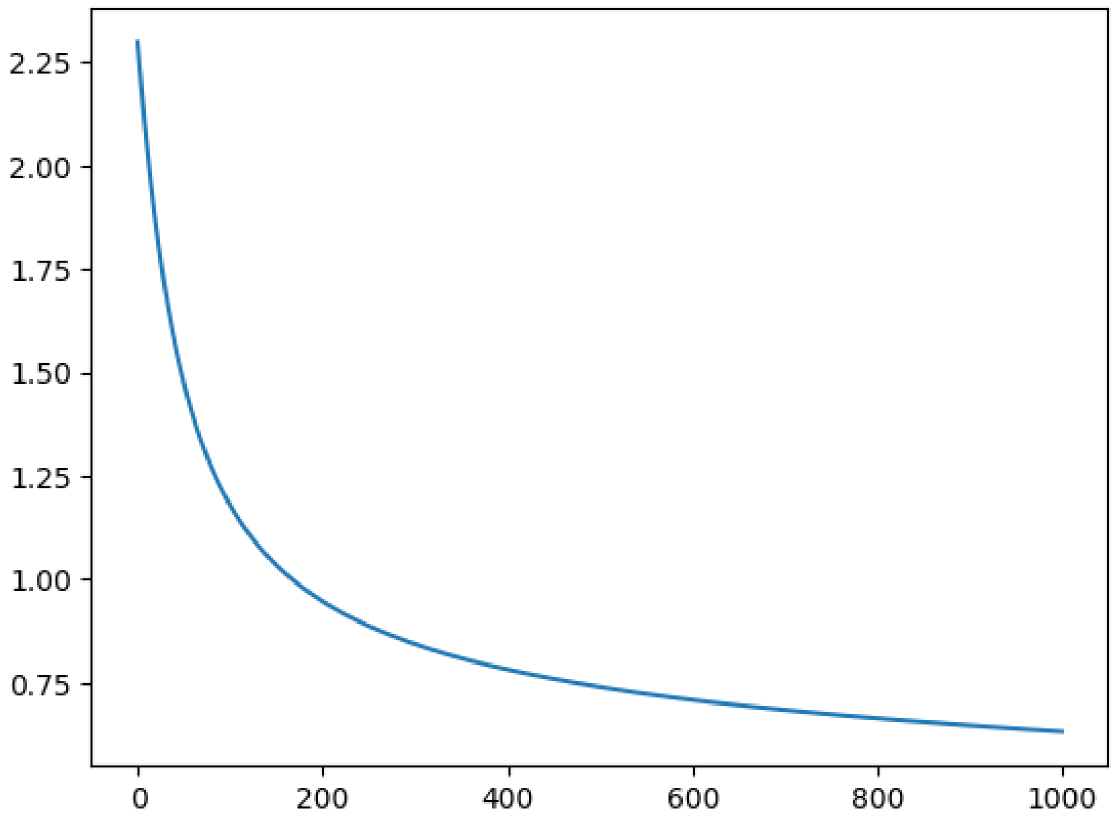


- Output layer = Softmax.



Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

➢ Training Visualization



n- layer neural network

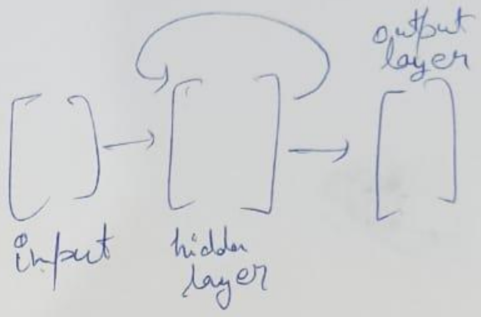
I had made it in such a way, when it is run it takes input of no. of layers and then takes the no. of neurons in each layer.

I can also do the same for activation layer but not done yet.

Cost function:

```
cost = -(1./m) * np.sum(y * np.log(AL))
```

① Initialize parameter



W_l = np. random. randn (n_l, n_{l-1})

b_l = np. zeros $((n_l, 1))$

W_L = np. random

B_L =

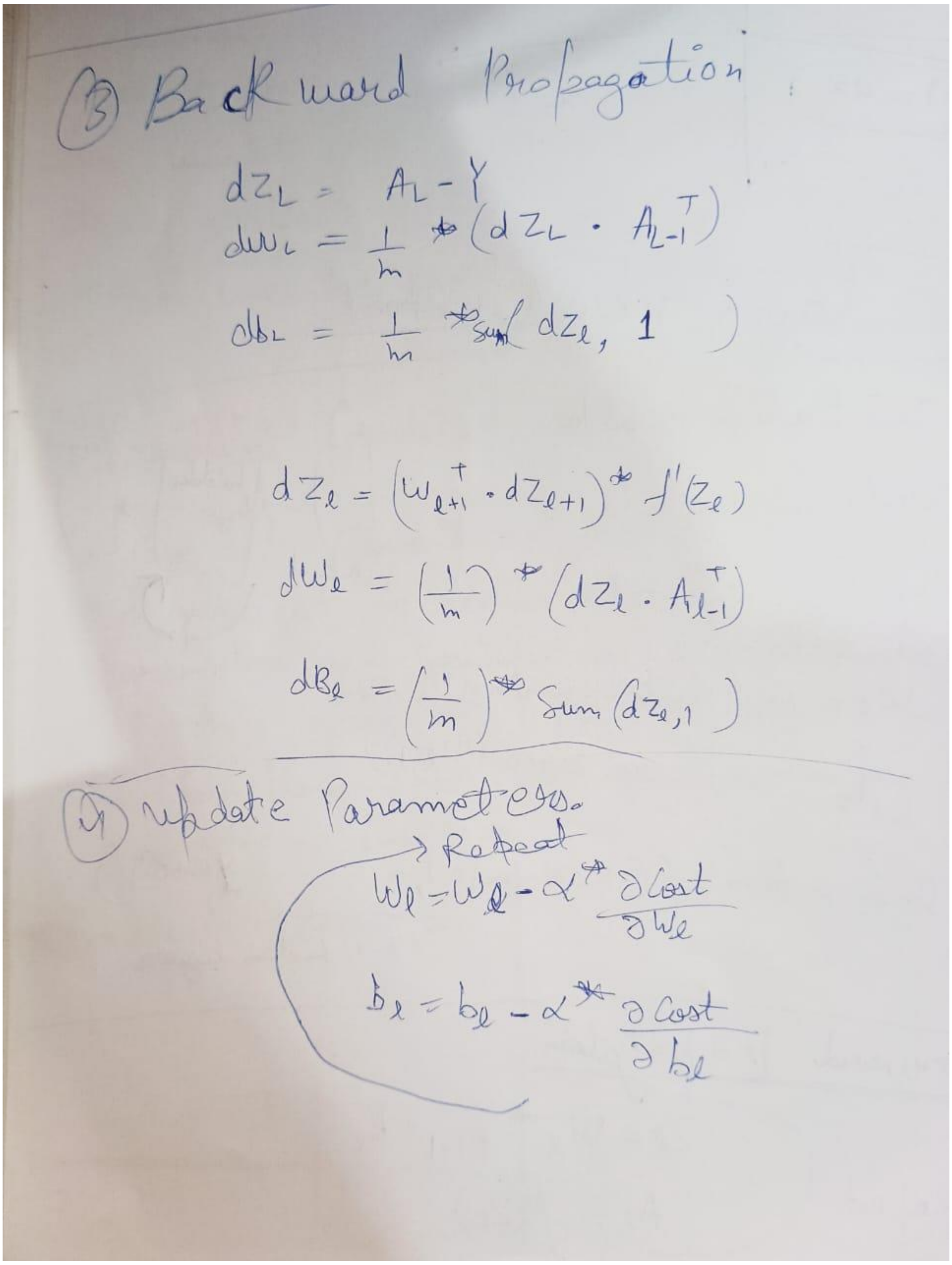
② Forward Propagation

Repeat

$$Z_l = W_l * A_{l-1} + B_l$$
$$A_l = f(Z_l)$$

output layer.

$$\begin{cases} Z_L = W_L * A_{L-1} + B_L \\ A_L = \text{Softmax}(Z_l) \end{cases}$$



③ Backward Propagation

$$dZ_L = A_L - Y$$

$$dw_L = \frac{1}{m} * (dZ_L \cdot A_{L-1}^T)$$

$$db_L = \frac{1}{m} * \text{sum}(dZ_l, 1)$$

$$dZ_l = (W_{l+1}^T \cdot dZ_{l+1}) * f'(Z_l)$$

$$dW_l = \left(\frac{1}{m}\right) * (dZ_l \cdot A_{l-1}^T)$$

$$dB_l = \left(\frac{1}{m}\right) * \text{Sum}(dZ_l, 1)$$

④ update Parameters.

→ Repeat

$$W_l = W_l - \alpha * \frac{\partial Cost}{\partial W_l}$$

$$b_l = b_l - \alpha * \frac{\partial Cost}{\partial b_l}$$

➢ Training logs

- Time taken : aprox. 10 minutes.
- Accuracy of train data = 85.675%
- Accuracy of test data = 84.54%
- Input data = classification_train.

```
o iterations = 0 cost = 2.339407978602308
o iterations = 250 cost = 1.1609545384954905
o iterations = 500 cost = 0.7206475402349102
o iterations = 750 cost = 0.6001089944580688
o iterations = 1000 cost = 0.5437828630472133
o iterations = 1250 cost = 0.5077766559458278
o iterations = 1500 cost = 0.4811315284008757
o iterations = 1750 cost = 0.45930910301857814
o iterations = 2000 cost = 0.44011023563936946
o iterations = 2250 cost = 0.42259213673579615
o
```

➢ Hyperparameters

- iterations = 2500
- learning_rate = 0.009
- No. of layers = 4 (3 hidden and 1 output layer).
- No. of neurons in each layer:

  Hidden layer1 = 50,

  Hidden layer2 = 25,

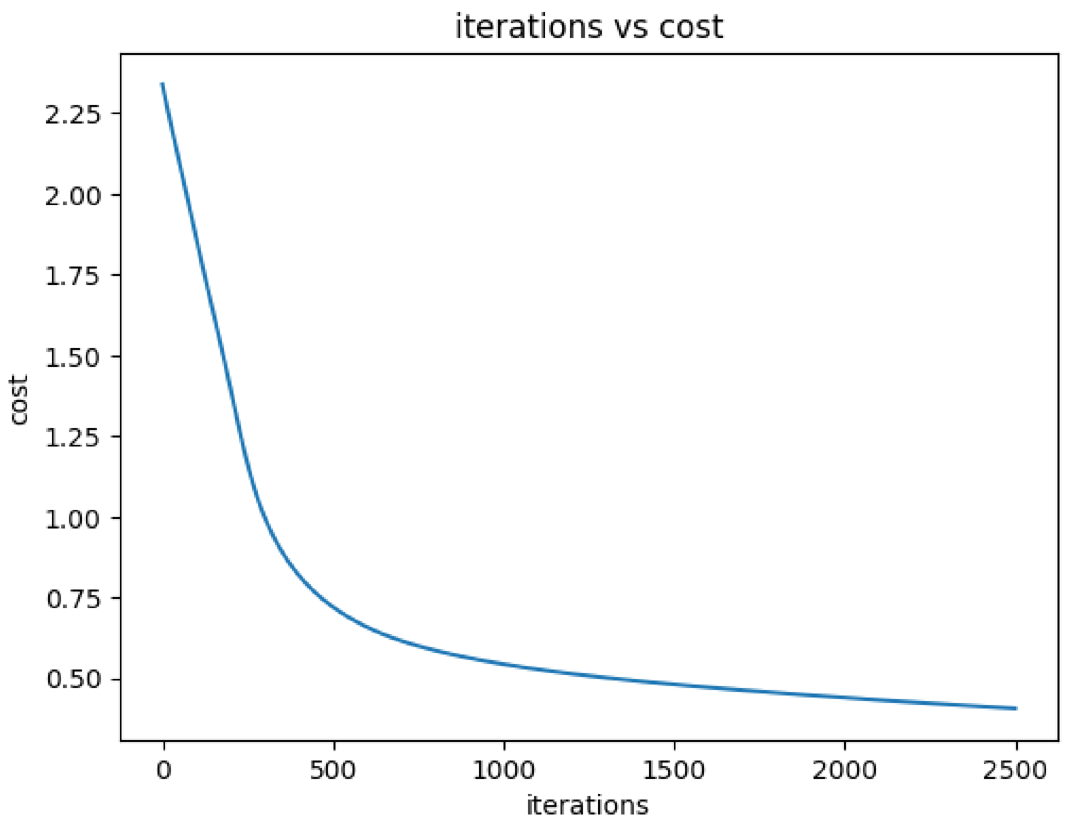
  Hidden layer3 = 15,

  Output layer = 10
- Activation functions in each layer:

  Hidden layers = Relu.

  Output layer = Softmax.

➢ Training Visualization


iterations vs cost
cost
iterations