ECE 657 ASSIGNMENT 1

Problem 1:
Download and read the technical paper titled On Convergence Proofs for Perceptrons (link) by Albert Novikoff which provides a proof for the convergence of perceptrons in a finite number of steps. After understanding the paper, restate the proof and explain each step in your way

Solution:
Statement: A theorem proposed by Rosenblatt and collaborators states that, "If two classes denoted by +1 and -1 are linearly separable, then a hyperplane separating them can be found by beginning with random weights and followed by repetitively fine-tuning weights using method called 'error correction' within finite iterations. On the other hand, if these classes are not linearly separable, then these steps may repeat infinitely in quest of getting hyperplane."

Setup:
Let's consider **N data points** in Euclidean space and hyperplane or vector distinguishing two classes is denoted by y. Corresponding to these data points, fix dimension set of **input vectors w_1**, …, **w_n** represented. V_0, …, V_n indicates the updated vectors corresponding to **n iterations** (explained in more details in section ahead).

Weight update procedure:
1. Process of finding hyperplane starts with random assignment of weights and thus arbitrary vector (denoted by V0 here).
2. After each iteration, this vector is modified for each incorrectly classified data point.
3. For each iteration, the weight is updated using rule:
    a. If classification is correct, old weight is carried forward. In other words, old weight is assigned as a new weight.
    b. If classification is incorrect, some step size (usually margin or $\boldsymbol{w_{i_n}}$) is added to previous weight in order to get updated weight.
4. If all the data points under analysis are linearly separable, above mentioned procedure will be repeated for finite iterations and at the end, a final hyperplane will be defined, which will classify data points correctly.

Proof:
For each data points $\boldsymbol{w_1}$, …., $\boldsymbol{w_N}$ denoted in Euclidean space, if and only if these points are linearly separable, there exist a hyperplane denoted by y which can satisfactorily separate two classes and is expressed by equation 1 as follows.

$$(\boldsymbol{w}_i \ . \boldsymbol{y}) > \boldsymbol{\theta} > \boldsymbol{0} \qquad \boldsymbol{i} = \boldsymbol{1}, \ldots\ldots, \boldsymbol{N} \qquad \_\_\_\_\text{Equation (1)}$$

To begin with process of achieving a perfect hyperplane, let us consider a vector denoted by v_o as initial random vector separating data points. Further the vector is modified with the help of error correction rule, which states, for correctly classified points the previous weight is carried forward as a new weight, on the other hand for incorrectly classified data points, the new weight vector is summation of old weight vector and weight vector value corresponding to data points. This error correction is iterated over all available data points ranging from 1 to N and thus the resultant error correction is denoted as shown in equation 2.

$$\boldsymbol{v_0} \ \ \textbf{\textit{is arbitrary}}$$

$$\boldsymbol{v_n} = \begin{cases} \boldsymbol{v_{n-1}} & \textbf{\textit{if}} \left(\boldsymbol{w_{i_n}} . \boldsymbol{v_{n-1}}\right) > \boldsymbol{\theta} \\ \boldsymbol{v_{n-1}} + \boldsymbol{w_{i_n}} & \textbf{\textit{if}} \left(\boldsymbol{w_{i_n}} . \boldsymbol{v_{n-1}}\right) \leq \boldsymbol{\theta} \end{cases}$$ _____Equation (2)

dot ($\boldsymbol{w_{i_n}}$, Vn-1)> $\boldsymbol{\theta}$ denote the correctly classified $\boldsymbol{w_{i_n}}$and: dot ($\boldsymbol{w_{i_n}}$, Vn-1) <= $\boldsymbol{\theta}$ indicates incorrect classification. Here correctly classified means that $\boldsymbol{w_{i_n}}$ is on the correct side of the hyperplane defined by v.

Here the hyperplane perpendicular to w is defined by y= {dot ($\boldsymbol{w_{i_n}}$, Vn-1 =$\boldsymbol{\theta}$} which can be represented as y = {dot ($\boldsymbol{w_{i_n}}$, Vn-1) = 0} without loss of generality.

Since there is no re-adjustment of weights for correctly classified data the error correction process can be summarized as shown in equation 3:

$$\boldsymbol{v_n} = \boldsymbol{v_{n-1}} + \boldsymbol{w_{i_n}} \quad \textbf{\textit{and}} \quad \left(\boldsymbol{w_{i_n}} . \boldsymbol{v_{n-1}}\right) \leq \boldsymbol{\theta} \ \textbf{\textit{for each n}}$$ _____Equation (3)

The error correction process is repeated "n" number of times and so, after iterations and error correction considering misclassified point the magnitude of v_n can be denoted in term of number of iterations as follows:

$$\| \boldsymbol{v_n} \|^2 = \boldsymbol{C}n^2$$ ______Equation (4)

Where C is a positive constant if n is sufficiently large.

According to Cauchy-Schwarz inequality the product of sums of squares is always greater than or equal to the square of a sum of products. Thus,

$$\| \boldsymbol{v_n} \|^2 \leq \| \boldsymbol{v_n} \|^2 + (\boldsymbol{2\theta} + \boldsymbol{M})\boldsymbol{n}$$

Where $$\boldsymbol{M} = \textbf{\textit{max}} \ \| \boldsymbol{v_i} \|^2 \qquad \boldsymbol{i} = \boldsymbol{1}, \dots \dots, \boldsymbol{N}$$ ______Equation (5)

$$\| \boldsymbol{v_k} \|^2 - \| \boldsymbol{v_{k-1}} \|^2 = \boldsymbol{2}\left(\boldsymbol{v_{k-1}}, \boldsymbol{w_{i_k}}\right) + \| \boldsymbol{w_{i_k}} \|^2 \leq \boldsymbol{2\theta} + \boldsymbol{M}$$

The inequality follows from the fact that $\boldsymbol{2}\left(\boldsymbol{v_{k-1}}, \boldsymbol{w_{i_k}}\right) < \boldsymbol{0}$ as we had to make an update, meaning w_i was misclassified $\boldsymbol{0} \leq \boldsymbol{2\theta} + \boldsymbol{M} \leq \boldsymbol{1}$.

Problem 2

The three-layer network in Fig. 1 divides the plane with three lines forming a triangle. Calculate the weights that will give a triangle with its vertices at (x, y) coordinates (0, 0), (1, 3), and (3, 1). Remember that a single perceptron can act as a linear separator. Using this idea, create the triangle with the vertices given. Assign a class to be inside the triangle and the other one to be outside the triangle and try to separate them using 3 perceptron's as shown in the figure below.

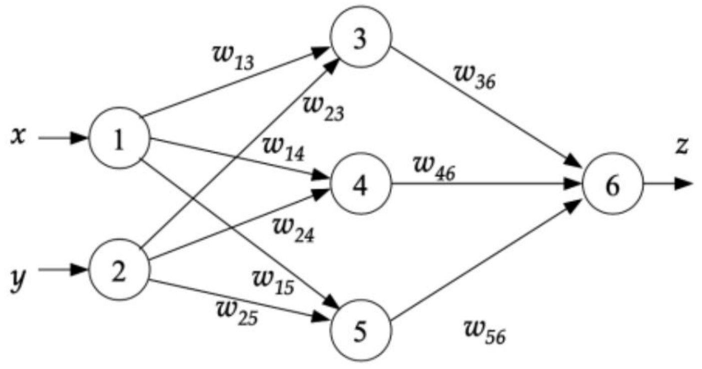


Figure 1: Structure of the three layered network of problem 2

Solution

Hyper plane/vertices are formed by points:

Point 1:	(0, 0) i.e. x=0, y=0
Point 2:	(1, 3) i.e. x=1, y=3
Point 3:	(3, 1) i.e. x=3, y=1

Lines/vertices are given by slope (m) = $\frac{y_2 - y_1}{x_1 - x_2}$

1. For point 1 and point 2
   m = (3 – 0)/ (1 – 0) = 3
   Equation of line: $y - y_1 = m\,(x - x_1)$
   y - 0 = 3 (x - 0)
   3x - y = 0

   Equation of perceptron:
   $w_{13}\,x + w_{23}\,y - b = 0$

   Comparing above equations, we get: $\boldsymbol{w_{13} = 3,\ w_{23} = -1,\ b = 0}$

2. For point 2 and point 3
   m = (1 – 3)/ (3 – 1) = -1
   Equation of line: $y - y_1 = m\,(x - x_1)$
   y - 3 = -1 (x - 1)
   x + y - 4 = 0

   Equation of perceptron:
   $w_{14}\,x + w_{24}\,y - b = 0$

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

Comparing above equations, we get: $\boldsymbol{w_{14}}$**= 1,** $\boldsymbol{w_{24}}$**= 1, b = 4**

3. For point 3 and point 1
   m = (0 – 1)/ (0 – 3) = 1/3
   Equation of line: $y - y_1 = m(x - x_1)$
   y - 1 = 1/3 (x - 3)
   x - 3y = 0

   Equation of perceptron:
   $w_{15}x + w_{25}y - b = 0$

   Comparing above equations, we get: $\boldsymbol{w_{15}}$**= 1,** $\boldsymbol{w_{25}}$**= -3, b = 0**

By taking initial weights and bias as follows
$w_{13} = 3$, $w_{23} = -1$, $w_{14} = 1$, $w_{24} = 1$, $w_{15} = 1$, $w_{25} = -3$

Consider that:

z = -1 (inside the triangle)
z = 1 (outside the triangle)

$$\text{Signum}(x) = \begin{cases} 1 & if\ x \geq 0 \\ -1 & if\ x < 0 \end{cases}$$

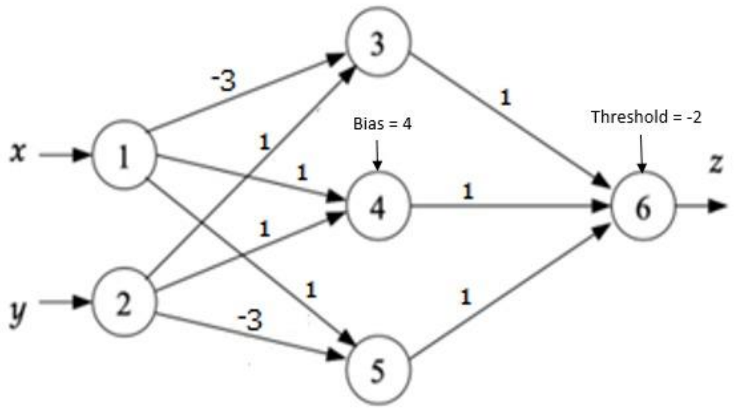
Consider an input (1, 2)

P3 = sgn(X*w_{13} + Y*w_{23}) = sgn (1(3) + (2) (-1)) = 1
P4 = sgn(X*w_{14} + Y*w_{24}) = sgn (1(1) + (2) (1) -4) = -1
P5 = sgn(X*w_{15} + Y*w_{25}) = sgn (1(1) + (2) (-3)) = -1

As we get the output of perceptron as (1, -1, -1), as we are assuming T=-1 as inside of the triangle to obtain this we need to have value of P3 as -1. So, interchanging the polarity of weights of the respective perceptron.

Updated weights:

$\boldsymbol{w_{13} = -3, w_{23} = 1, w_{14} = 1, w_{24} = 1, w_{15} = 1, w_{25} = -3}$



x	2	2	5	-3	5.2	-3.2
y	1	2	4.8	2	-1	-2
T	-1	1	1	1	1	1
z	-1	1	1	1	1	1

T = Expected Output, Z = Final Output

Assuming the weights: $\boldsymbol{w_{36}}$ **=1,** $\boldsymbol{w_{46}}$ **=1,** $\boldsymbol{w_{56}}$ **=1,** $\mathbf{T_b}$ **= -2**

- Consider an input (1, 2)

P3 = sgn(X*w_{13} + Y*w_{23}) = sgn (1(-3) + (2) (1)) = -1
P4 = sgn(X*w_{14} + Y*w_{24}) = sgn (1(1) + (2) (1) -4) = -1
P5 = sgn(X*w_{15} + Y*w_{25}) = sgn (1(1) + (2) (-3)) = -1

Z = sgn ((P3*w_{36}+ P4*w_{46}+ P5*w_{56}) +2) = sgn (((-1) (1) + (-1) (1) + (-1) (1))-(-2)) = -1

As we get the value of z =-1 and as we assume T=-1 for the inputs (1, 2). Hence these points lie inside the triangle.

- Consider an input (2, 2)

P3 = sgn(X*w_{13} + Y*w_{23}) = sgn (2(-3) + (2) (1)) = -1
P4 = sgn(X*w_{14} + Y*w_{24}) = sgn (2(1) + (2) (1) -4) = 1
P5 = sgn(X*w_{15} + Y*w_{25}) = sgn (2(1) + (2) (-3)) = -1

Z = sgn ((P3*w_{36}+ P4*w_{46}+ P5*w_{56}) +2) = sgn (((-1) (1) + (1) (1) + (-1) (1)) - (-2)) = 1

As we get the value of Z=1 and as we assume T=1 for the inputs (2, 2). Hence these points lie outside the triangle.

- Consider an input (5, 4.8)

P3 = sgn(X*w_{13} + Y*w_{23}) = sgn (5(-3) + (4.8) (1)) = -1
P4 = sgn(X*w_{14} + Y*w_{24}) = sgn (5(1) + (4.8) (1) -4) = 1
P5 = sgn(X*w_{15} + Y*w_{25}) = sgn (5(1) + (4.8) (-3)) = -1

Z = sgn ((P3*w_{36}+ P4*w_{46}+ P5*w_{56}) +2) = sgn (((-1) (1) + (1) (1) + (-1) (1)) - (-2)) = 1

As we get the value of Z=1 and as we assume T=1 for the inputs (5, 4.8). Hence these points lie outside the triangle.

- Consider an input (-3, 2)

P3 = sgn(X*w_{13} + Y*w_{23}) = sgn ((-3) (-3) + (2) (1)) = 1
P4 = sgn(X*w_{14} + Y*w_{24}) = sgn ((-3) (1) + (2) (1) -4) = -1
P5 = sgn(X*w_{15} + Y*w_{25}) = sgn ((-3) (1) + (2) (-3)) = -1

Z = sgn ((P3*w_{36}+ P4*w_{46}+ P5*w_{56}) + 2) = sgn (((1) (1) + (-1) (1) + (-1) (1)) - (-2)) = 1

As we get the value of Z=1 and as we assume T=1 for the inputs (-3, 2). Hence these points lie outside the triangle.

- Consider an input (5.2, -1)

P3 = sgn(X*w_{13} + Y*w_{23}) = sgn ((5.2) (-3) + (-1) (1)) = -1
P4 = sgn(X*w_{14} + Y*w_{24}) = sgn ((5.2) (1) + (-1) (1) -4) = 1
P5 = sgn(X*w_{15} + Y*w_{25}) = sgn ((5.2) (1) + (-1) (-3)) = 1

Z = sgn ((P3*w_{36}+ P4*w_{46}+ P5*w_{56}) + 2) = sgn (((-1) (1) + (1) (1) + (1) (1)) - (-2)) = 1

As we get the value of Z=1 and as we assume T=1 for the inputs (5.2, -1). Hence these points lie outside the triangle.

- Consider an input (-3.2, -2)

P3 = sgn(X*w_{13} + Y*w_{23}) = sgn ((-3.2) (-3) + (-2) (1)) = 1
P4 = sgn(X*w_{14} + Y*w_{24}) = sgn ((-3.2) (1) + (-2) (1) -4) = -1
P5 = sgn(X*w_{15} + Y*w_{25}) = sgn ((-3.2) (1) + (-2) (-3)) = 1

Z = sgn ((P3*w_{36}+ P4*w_{46}+ P5*w_{56}) +2) = sgn (((-1) (1) + (1) (1) + (1) (1)) - (-2)) = 1

As we get the value of Z=1 and as we assume T=1 for the inputs (-3.2, -2). Hence these points lie outside the triangle.

Problem 3

The normalized Widrow Hoff learning rule also known as the normalized least mean square learning rule is expressed as: $\Delta \boldsymbol{w}^{(\boldsymbol{k})} = \boldsymbol{\eta}\left(\boldsymbol{t}^{(\boldsymbol{k})} - \boldsymbol{w}^{(\boldsymbol{k})}\boldsymbol{x}^{(\boldsymbol{k})}\right) \frac{\boldsymbol{x}^{(\boldsymbol{k})}}{\left\|\boldsymbol{x}^{(\boldsymbol{k})}\right\|^2}$ Where, $\Delta \boldsymbol{w}^{(\boldsymbol{k})} = \boldsymbol{w}^{(\boldsymbol{k+1})} - \boldsymbol{w}^{(\boldsymbol{k})}$) represents the weight vector change from iteration (k) to iteration (k+1), ||x (k) || is the Euclidean norm of the input vector $\boldsymbol{x}^{(\boldsymbol{k})}$ at iteration (k), $\mathbf{t}^{(\boldsymbol{k})}$is the target at iteration (k) and η is a positive number ranging from 0 to 1: 0 < η < 1. Show that if the same input vector $\boldsymbol{x}^{(\boldsymbol{k})}$is presented at iteration (k + 1), then the weight vector decreases by factor (1 – η) going from iteration (k) to iteration (k+1). That is: $\Delta \boldsymbol{w}^{(\boldsymbol{k+1})} = (\mathbf{1} - \boldsymbol{\eta})\Delta \boldsymbol{w}^{(\boldsymbol{k})}$

Solution:

Given:

$$\Delta w^{(k)} = \eta\left(t^{(k)} - w^{(k)}x^{(k)}\right) \frac{x^{(k)}}{\left\|x^{(k)}\right\|^2} \qquad \text{- (1)}$$

From Equation 1

$$\frac{\Delta w^{(k)}\left\|x^{(k)}\right\|^2}{\eta x^k} = \left(t^{(k)} - w^{(k)}x^{(k)}\right)$$

$$w^{(k)}x^{(k)} = \left(t^{(k)} - \frac{\Delta w^{(k)}\left\|x^{(k)}\right\|^2}{\eta x^k}\right) \qquad \text{- (2)}$$

Given

$$\Delta w^{(k)} = w^{(k+1)} - w^{(k)}$$

$$w^{(k+1)} = \Delta w^{(k)} + w^{(k)} \qquad \text{- (3)}$$

Proof:

Given

$$\Delta w^{(k)} = \eta\left(t^{(k)} - w^{(k)}x^{(k)}\right) \frac{x^{(k)}}{\left\|x^{(k)}\right\|^2}$$

Now at $K + 1^{th}$ iteration.

$$\Delta w^{(k+1)} = \eta\left(t^{(k+1)} - w^{(k+1)}x^{(k+1)}\right) \frac{x^{(k+1)}}{\left\|x^{(k+1)}\right\|^2}$$

Since same input vector $x^{(k)}$ is presented at $K + 1^{th}$ iteration.

$$x^{(k)} = x^{(k)} = x$$

$$\Delta w^{(k+1)} = \eta\left(t^{(k+1)} - w^{(k+1)}x\right) \frac{x}{\|x\|^2}$$

From Equation 3

$$\Delta w^{(k+1)} = \eta\left(t^{(k+1)} - \left(\Delta w^{(k)} + w^{(k)}\right)x\right) \frac{x}{\|x\|^2}$$

$$\Delta w^{(k+1)} = \eta\left(t^{(k+1)} - \Delta w^{(k)} - w^{(k)} x\right) \frac{x}{\|x\|^2}$$

From Equation 2

$$\Delta w^{(k+1)} = \eta\left(t^{(k+1)} - \Delta w^{(k)} x - \left(t^{(k)} - \frac{\Delta w^{(k)}\|x\|^2}{\eta x}\right)\right) \frac{x}{\|x\|^2}$$

Since $t^{(k)}$ represents target at iteration (k)

$$\therefore \quad t^{(k+1)} = t^{(k)}$$

$$\Delta w^{(k+1)} = \eta\left(t^{(k+1)} - \Delta w^{(k)} x - \left(t^{(k+1)} - \frac{\Delta w^{(k)}\|x\|^2}{\eta x}\right)\right) \frac{x}{\|x\|^2}$$

$$\Delta w^{(k+1)} = \eta\left(\frac{\Delta w^{(k)}\|x\|^2}{\eta x} - \Delta w^{(k)}\, x\right) \frac{x}{\|x\|^2}$$

$$\Delta w^{(k+1)} = \eta\left(\frac{\Delta w^{(k)}}{\eta} - \Delta w^{(k)}\, \frac{x^2}{\|x\|^2}\right)$$

$Since \ \frac{x^2}{\|x\|^2} = 1$ [Ref: Appendix 1]

$$\Delta w^{(k+1)} = \eta\left(\frac{\Delta w^{(k)}}{\eta} - \Delta w^{(k)}\right)$$

$$\Delta w^{(k+1)} = \left(\Delta w^{(k)} - \eta \Delta w^{(k)}\right)$$

$$\Delta w^{(k+1)} = (1 - \eta)\Delta w^{(k)}$$

Appendix 1:

Consider a vector x (a, b)
Now

$$\|x\| = \sqrt{a^2 + b^2}$$

$$\|x\|^2 = a^2 + b^2 \qquad \text{- Eq (1)}$$

Also,

$$x^2 = x \,.\, x^T = [a \quad b] * \begin{bmatrix} a \\ b \end{bmatrix}$$

$$x^2 = a^2 + b^2 \qquad \text{- Eq (2)}$$

From equation 1 & 2,

$$\boldsymbol{\frac{x^2}{\|x\|^2} = 1}$$

Problem 4

In this problem, you will implement the backpropagation algorithm and train your first multi-layer perceptron to distinguish between 4 classes. You should implement everything on your own without using existing libraries like Tensor flow, Keras, or PyTorch. You are provided with two files "train data.csv" and "train labels.csv" The dataset contains 24754 samples, each with 784 features divided into 4 classes (0,1,2,3). You should divide this into training, and validation sets (a validation set is used to make sure your network did not over fit). You will then provide your model which will be tested with an unseen test set. Use one input layer, one hidden layer, and one output layer in your implementation. The labels are one-hot encoded. For example, class 0 has a label of [1, 0, 0, 0] and class 2 has a label of [0, 0, 1, 0]. Make sure you use the appropriate activation function in the output layer. You are free to use any number of nodes in the hidden layer. You need to provide one single function that allows us to use the network to predict the test set. This function should output the labels one-hot encoded in a numpy array. Your work will be graded based on the successful implementation of the backpropagation algorithm, the multi-layer perceptron, and the test set accuracy. You may want to thoroughly comment your implementation to allow us to easily understand it.

Solution

In [14]:
```
import numpy as np
from numpy import genfromtxt
np.random.seed(0)
import pandas as pd
from math import exp
from sklearn.model_selection import train_test_split
```

In [15]:
```
# read dataset
X = genfromtxt("train_data.csv", delimiter=',')
y = genfromtxt("train_labels.csv", delimiter=',')
```

In [16]:
```
#checking the balance the dataset
print('Number of samples in each class\n', y.sum(axis=0))
```

```
Number of samples in each class
 [5923. 6742. 5958. 6131.]
```

In [17]:
```
#Splitting the data into training and validation set in a ratio of 8:2
#Created validation set to test the model on unknown data
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.2)
```

In [18]:
```
#checking the balance the dataset
print('Number of samples in each class in y_train', y_train.sum(axis=0))
print('Number of samples in each class in y_val', y_val.sum(axis=0))
```

```
Number of samples in each class in y_train [4760. 5382. 4770. 4891.]
Number of samples in each class in y_val [1163. 1360. 1188. 1240.]
```

In [19]:
```
# finding out the number of samples and features in train dataset
(samples,features) = X_train.shape
```

In [20]:
```
#declaring the number of hidden layer neurons, classes, learning rate and number of

hiddenlyr_nodes = 35
num_classes = 4
learning_rate = 0.0001
num_epoch  = 100
```

$SigmoidFunction = \frac{1}{1+e^x}$

S(x) = sigmoid function e = Euler's number

In [21]:
```
# sigmoid function
def sigmoid_func(x):
    return 1 / (1 + np.exp(-x))
# derivative of sigmoid function
def sigmoid_drv(x):
    return sigmoid_func(x) * (1 - sigmoid_func(x))
```

Sigmoid Fuunction $\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

σ = softmax $\vec{z}$ = input vector e^{z_i} = standard exponential function for input vector K = number of classes in the multi-class classifier e^{z_j} = standard exponential function for output vector e^{z_j} =

standard exponential function for output vector

In [22]:
```
# softmax function
def softmax_func(x):
    val = np.exp(x) / np.exp(x).sum(axis=1, keepdims=True)
    return val
```

In [23]:
```
# Convert predicted data into one hot encoding
def one_hot_enc(x):
    for i in range(0,len(x)):
      x[i,x[i,:].argmax()]=1
    out = (x == 1).astype(float)
    return out
```

In [24]:
```
# predicting the accuracy of the model
def accuracy(y_true, y_pred):
    if not (len(y_true) == len(y_pred)):
        print('Size of predicted and true labels not equal.')
        return 0.0

    corr = 0
    for i in range(0,len(y_true)):
        corr += 1 if (y_true[i] == y_pred[i]).all() else 0

    return corr/len(y_true)

# feed forward function
def fwd(inp_data, wt_hidlyr, bias_hidlyr,wt_outlyr,bias_outlyr):
    net_hidden = np.dot(inp_data, wt_hidlyr) + bias_hidlyr
    act_hidden = sigmoid_func(net_hidden)
    net_output = np.dot(act_hidden, wt_outlyr) + bias_outlyr
    act_output = softmax_func(net_output)
    return act_output, act_hidden, net_hidden

# backpropagation function
def bkd(X_train, y_train, net_hidden, act_hidden, weight_output, act_output):
    cf_netHid = act_output - y_train
    grad_bias_out = cf_netHid
    grad_wt_out = np.dot(act_hidden.T, cf_netHid)
    cf_actHid = np.dot(cf_netHid, weight_output.T)
    grad_wt_hid = np.dot(X_train.T, sigmoid_drv(net_hidden) * cf_actHid)
    grad_bias_hid = cf_actHid * sigmoid_drv(net_hidden)
    return grad_wt_out, grad_bias_out, grad_wt_hid, grad_bias_hid

# Updating Weight
def update_weight(weight, cost):
    if cost.shape == (features, hiddenlyr_nodes) or cost.shape == (hiddenlyr_nodes,
        weight = weight - learning_rate * cost
    elif cost.shape == (samples, hiddenlyr_nodes) or cost.shape == (samples, num_cla
        weight = weight - learning_rate * cost.sum(axis=0)
    return weight
```

Loss Function

$$MSE = \frac{1}{n} + \sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

MSE = mean squared error n = number of data points Y_i = observed values $\hat{Y}_i$ = predicted values

In [25]:
```
def cross_ent(original_label,predict_label):
    mse = np.square(original_label - predict_label)
    val = np.mean(mse)
    return val
```

In [26]:
```
# Weight initialization
weight_hidden = np.random.randn(features, hiddenlyr_nodes)
bias_hidden = np.random.randn(hiddenlyr_nodes)
weight_output = np.random.randn(hiddenlyr_nodes, num_classes)
bias_output = np.random.randn(num_classes)
```

In [27]:
```
error_per_epoch = list()
epoch = 0
while epoch < num_epoch:
  epoch+=1
  #forward propagation
  act_output, act_hidden, net_hidden = fwd(X_train, weight_hidden, bias_hidden, weig
  #backward propagation
  cost_wo, cost_bo, cost_wh, cost_bh = bkd(X_train, y_train, net_hidden, act_hidden,
  #weight updating
  weight_hidden = update_weight(weight_hidden, cost_wh)
  bias_hidden = update_weight(bias_hidden, cost_bh)
  weight_output = update_weight(weight_output, cost_wo)
  bias_output = update_weight(bias_output, cost_bo)

  cal_loss = cross_ent(y_train,act_output)
  error_per_epoch.append(cal_loss)

  y_pred, _, _ = fwd(X_val, weight_hidden, bias_hidden, weight_output, bias_output)
  # One hot encoding the prediction
  y_pred_enc = one_hot_enc(y_pred)
  # calculating the accuracy
  ACC = accuracy(y_val,y_pred_enc)
  if epoch%10==0:
    print('epoch = ',epoch,'    ','Loss function value: ', cal_loss,'accuracy = ',ACC
```

```
epoch =  10     Loss function value:  0.07782015133535355 accuracy =  0.812361139163
8053
epoch =  20     Loss function value:  0.038782371286173976 accuracy =  0.88891133104
42335
epoch =  30     Loss function value:  0.0327373254722133 accuracy =  0.9030498889113
311
epoch =  40     Loss function value:  0.029280660098058603 accuracy =  0.91153302363
15896
epoch =  50     Loss function value:  0.026876144030628303 accuracy =  0.91840032316
7037
epoch =  60     Loss function value:  0.025069403111138894 accuracy =  0.92385376691
57746
epoch =  70     Loss function value:  0.023643780827894978 accuracy =  0.92849929307
21067
epoch =  80     Loss function value:  0.022476938312714093 accuracy =  0.93253888103
41345
epoch =  90     Loss function value:  0.021493507960131226 accuracy =  0.93597253080
18582
epoch =  100     Loss function value:  0.02064508910771253 accuracy =  0.93758836598
66694
```

In [28]:
```
y_pred
```

Out[28]:
```
array([[1.13965178e-01, 6.95177561e-05, 1.00000000e+00, 2.17209706e-01],
       [3.12621456e-05, 1.00000000e+00, 6.67724574e-03, 3.57372858e-03],
       [8.78363481e-05, 1.00000000e+00, 9.06675912e-03, 1.86532339e-02],
       ...,
       [1.00000000e+00, 2.02731193e-06, 2.90643006e-03, 4.86376825e-05],
       [2.91785129e-04, 3.58194542e-02, 1.00000000e+00, 4.25361585e-03],
       [5.55190771e-03, 1.48629551e-05, 1.00000000e+00, 5.97542651e-04]])
```

In [29]:

```
ACC= accuracy(y_val,y_pred_enc)
print('accuracy on the validation set is ', ACC*100)
```

```
accuracy on the validation set is  93.75883659866695
```

In [30]:

```
np.save('Updated Weights/weight_hidden.npy', weight_hidden)
np.save('Updated Weights/bias_hidden.npy', bias_hidden)
np.save('Updated Weights/weight_output.npy', weight_output)
np.save('Updated Weights/bias_output.npy', bias_output)
```