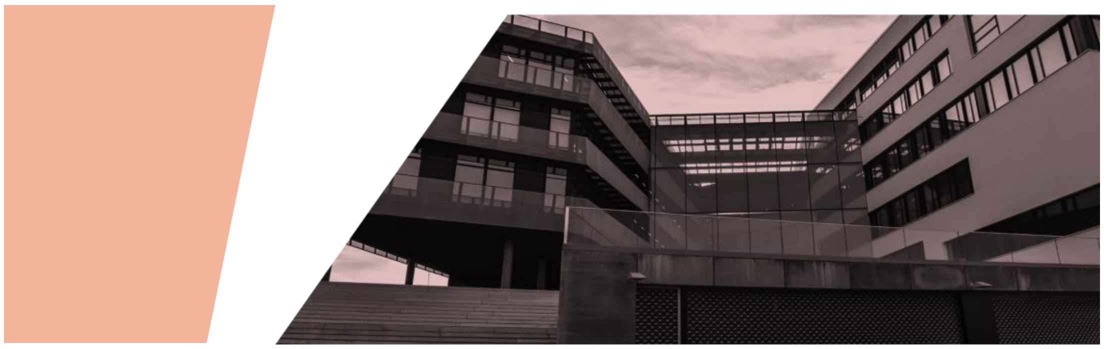
CSL- 450 MACHINE LEARNING

Assignment -1



Submitted by:

Mugdha Satish Kolhe – BT17CSE043

November 28, 2020

TABLE OF CONTENTS

1. Problem Statement

Problem Statement: Write functions that can model the following:

• Configuration parameters can be taken in for feedforward network

  o Number of hidden layers

  o Number of neurons in every layer

  o Activation units to be used in every layer

  o Network connections among neurons (fully connected or a different N/W architecture)

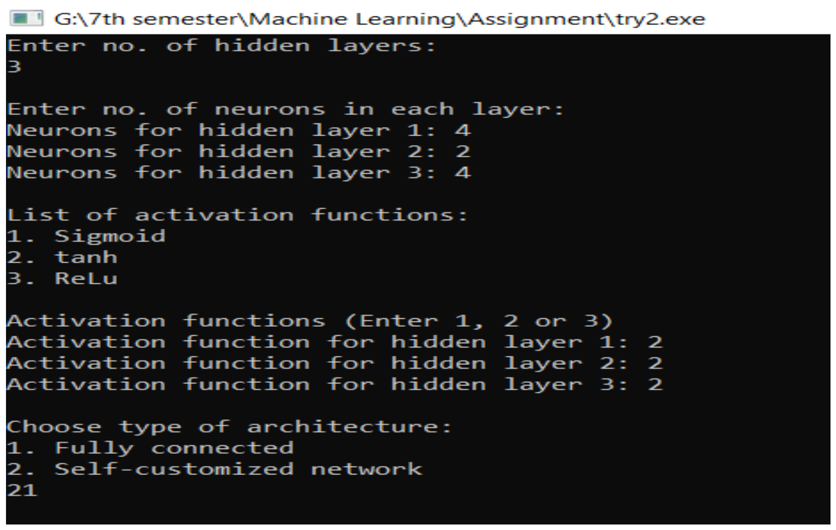
• Write code for backpropagation algorithm that comes up with NN based model for the above mentioned configuration parameters

• An appropriate technique of avoiding overfitting or regularization to be used. Better still, the choice of the overfitting technique can be made configurable

2. Parameters

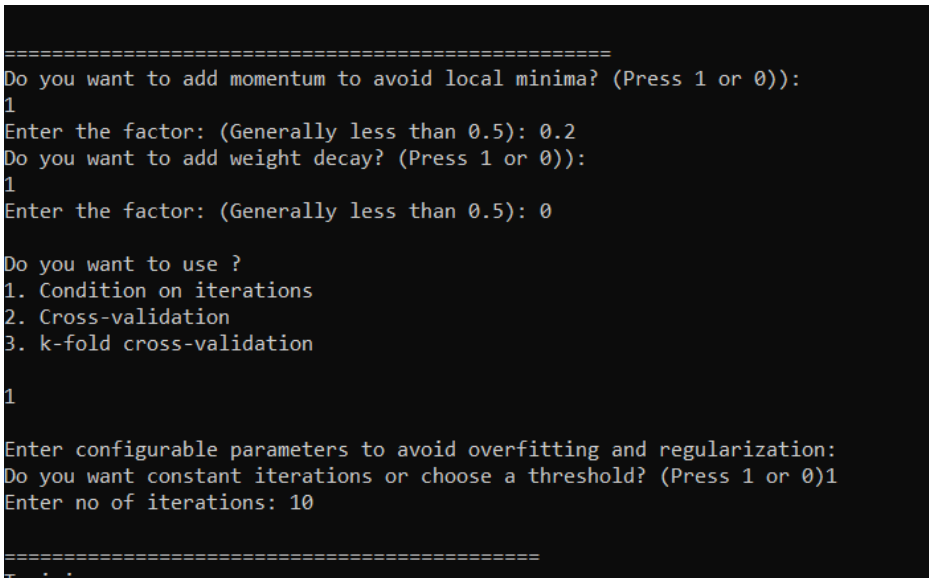
Configurable parameters in code:

User can enter:



```
G:\7th semester\Machine Learning\Assignment\try2.exe
Enter no. of hidden layers:
3

Enter no. of neurons in each layer:
Neurons for hidden layer 1: 4
Neurons for hidden layer 2: 2
Neurons for hidden layer 3: 4

List of activation functions:
1. Sigmoid
2. tanh
3. ReLu

Activation functions (Enter 1, 2 or 3)
Activation function for hidden layer 1: 2
Activation function for hidden layer 2: 2
Activation function for hidden layer 3: 2

Choose type of architecture:
1. Fully connected
2. Self-customized network
21
```

- **No of hidden layers:** User can enter the no of hidden layers in the neural network
- **No of units in each hidden layer:** After entering the no of hidden layers the user wants in the neural network, the user will be prompted to enter no of hidden units in each hidden layer
- **Activation function:** The user has a choose to choose amongst ReLu, Sigmoidal or tanh as activation function for each hidden layer. He will be asked to enter a function for each hidden layer (Press 1 for Sigmoidal, 2 for tanh and 3 for ReLu). The default is set to tanh in case user enters wrong no.
- **Type of architecture:** The user can either choose a fully connected network or can self- design the connections between the neurons. To do so the user will be shown the neurons one by one along with what choices he/she has to connect that neuron with in the next hidden layer. User has to press 1 to have connection between the shown neurons or 0 for no connection.



```
===================================================
Do you want to add momentum to avoid local minima? (Press 1 or 0)):
1
Enter the factor: (Generally less than 0.5): 0.2
Do you want to add weight decay? (Press 1 or 0)):
1
Enter the factor: (Generally less than 0.5): 0

Do you want to use ?
1. Condition on iterations
2. Cross-validation
3. k-fold cross-validation

1

Enter configurable parameters to avoid overfitting and regularization:
Do you want constant iterations or choose a threshold? (Press 1 or 0)1
Enter no of iterations: 10

=============================================
```

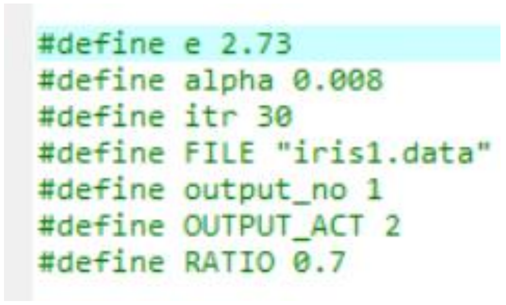
To stop overfitting user can chose parameters like:

- Add **momentum** to converge weights quickly: On pressing 1 (i.e. yes) the suer can enter the threshold value by with the weights will converge.
- User can choose to add **weight decay**: On pressing yes (i.e. 1) user can enter the factor by which each of the weight updates will show down to avoid overfitting.

- User can choose between:
    1. **Condition on no. of iterations:** Enter fixed no of epochs or choose a threshold such that when error reaches a threshold we can stop the training and use those weights for testing.
    2. **Cross-validation:** The training-data will be broken down into two parts by some random no r from 0 to 1 which will be the ratio. According to value of r the data will be sent for training and validation and error will be calculated on validation set. The set of best weights will be maintained. If the difference between training weights currently used give error on validation set and the best weights tested so far on validation sets is greater then a threshold we stop the training**.**
    3. **K-fold Cross-Validation:** We repeat the above process k no of times as entered by the user. Each time splitting training data into different ratios. Then we calculation the no of iterations in each time we perform cross validation and then take the average of them. Then we use that no of iterations to train the entire training set and the used those weights obtained for testing.

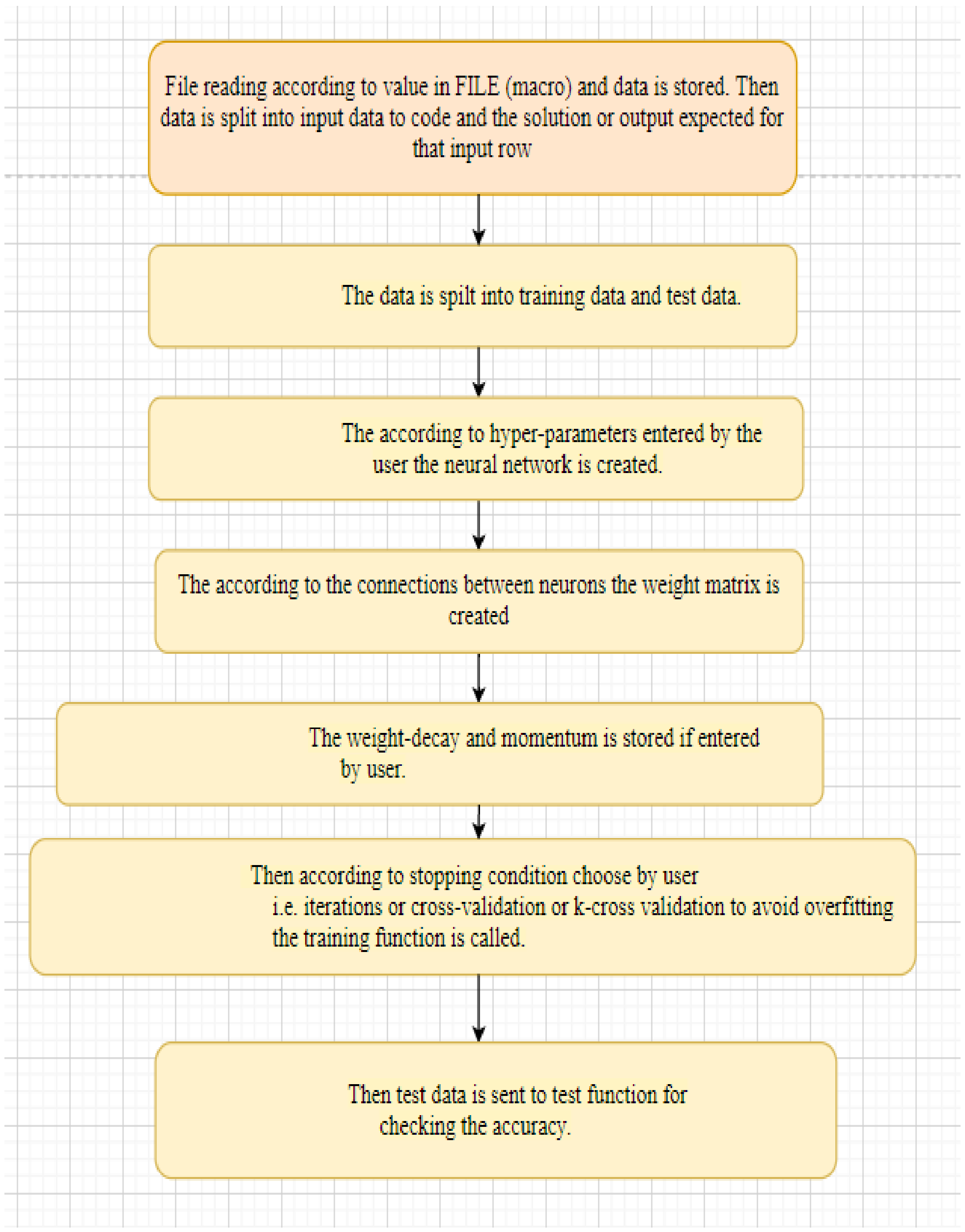
Also, user can change values of parameters such as :

- The learning rate denoted by 'alpha'
- No of outputs in output units in output layer of neural network
- The activation function of output layer

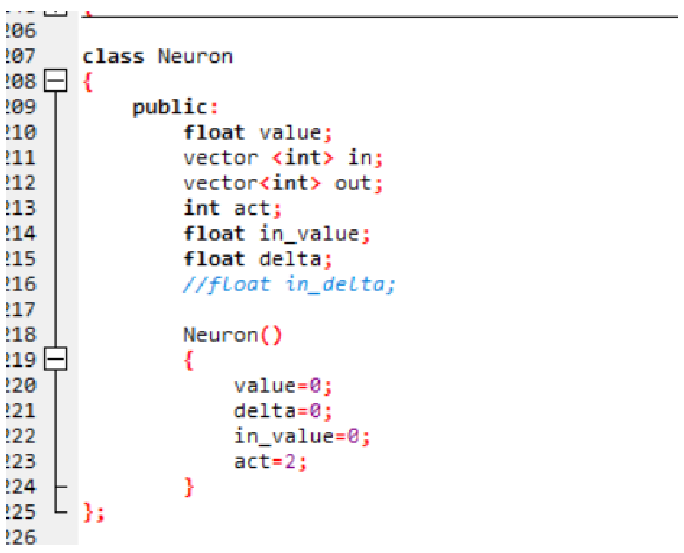


```
#define e 2.73
#define alpha 0.008
#define itr 30
#define FILE "iris1.data"
#define output_no 1
#define OUTPUT_ACT 2
#define RATIO 0.7
```

3. Flow of Code


File reading according to value in FILE (macro) and data is stored. Then data is split into input data to code and the solution or output expected for that input row
The data is spilt into training data and test data.
The according to hyper-parameters entered by the user the neural network is created.
The according to the connections between neurons the weight matrix is created
The weight-decay and momentum is stored if entered by user.
Then according to stopping condition choose by user i.e. iterations or cross-validation or k-cross validation to avoid overfitting the training function is called.
Then test data is sent to test function for checking the accuracy.

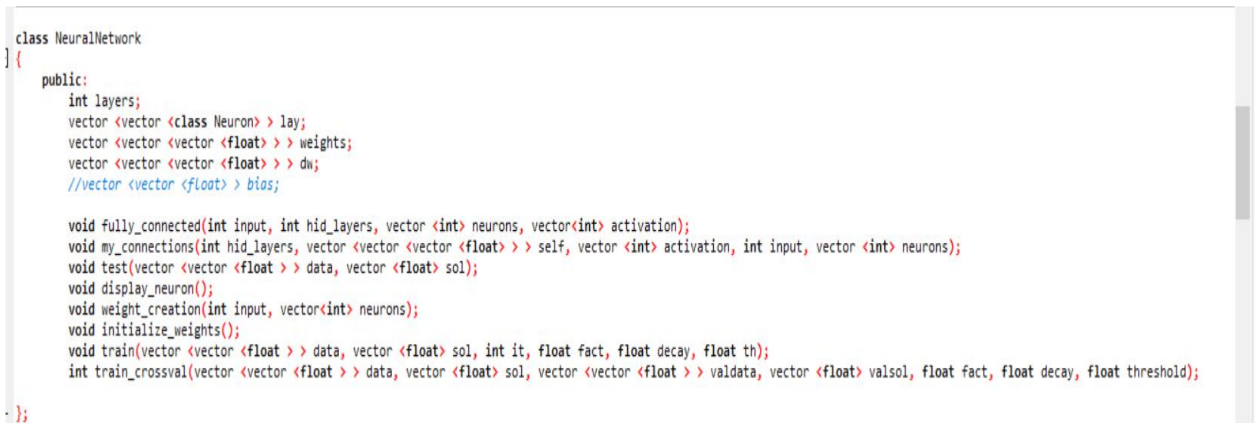
4. Functions



```
class Neuron
{
    public:
        float value;
        vector <int> in;
        vector<int> out;
        int act;
        float in_value;
        float delta;
        //float in_delta;

        Neuron()
        {
            value=0;
            delta=0;
            in_value=0;
            act=2;
        }
};
```

Class Neuron:

- in_value: stores the value incoming to that node that is the summation of weights conncting to that node multiplied by the node in prev layer which connctes that weight
- value: After applying activation function to that neuron on in_value the value obtained is stored in this variable
- act: stores the activation function. 1 for sigmoid 2 for tanh and 3 for Relu
- delta: stores the error at that neuron during back propagation
- in vector: stores the incoming nodes connected to that neuron from previous layer
- out vector: stores the nodes connected by outgoing edges from that neuron to the neuron at next layer



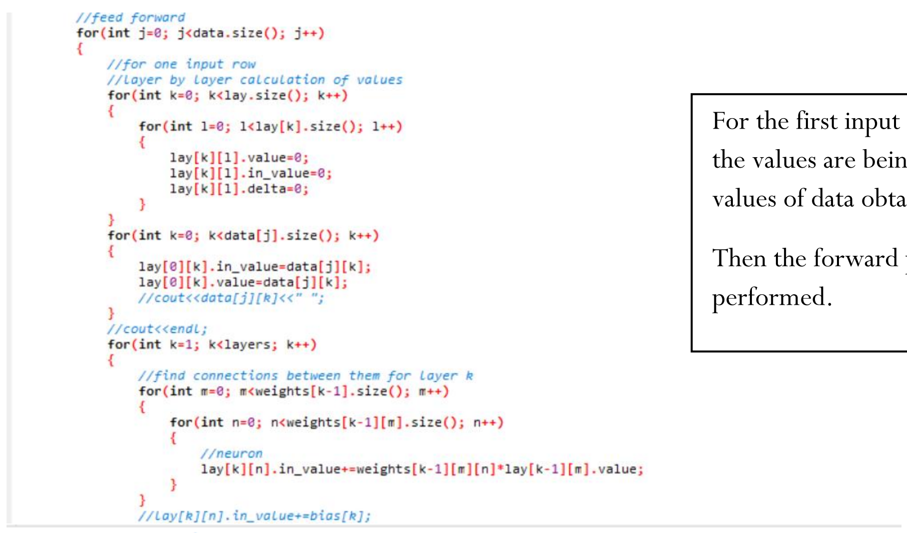
```
class NeuralNetwork
{
    public:
        int layers;
        vector <vector <class Neuron> > lay;
        vector <vector <vector <float> > > weights;
        vector <vector <vector <float> > > dw;
        //vector <vector <float> > bias;

        void fully_connected(int input, int hid_layers, vector <int> neurons, vector<int> activation);
        void my_connections(int hid_layers, vector <vector <vector <float> > > self, vector <int> activation, int input, vector <int> neurons);
        void test(vector <vector <float > > data, vector <float> sol);
        void display_neuron();
        void weight_creation(int input, vector<int> neurons);
        void initialize_weights();
        void train(vector <vector <float > > data, vector <float> sol, int it, float fact, float decay, float th);
        int train_crossval(vector <vector <float > > data, vector <float> sol, vector <vector <float > > valdata, vector <float> valsol, float fact, float decay, float threshold);

};
```

Class NeuralNetwork:

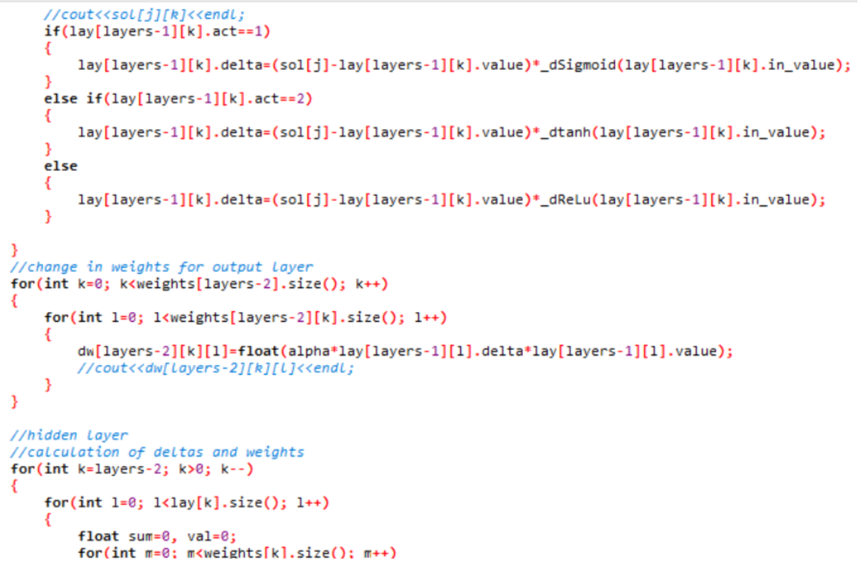
- layers: stores no of layers in the neural network including the input and output layers
- lay: stores the neurons in the neural network
- weights: stores the weight matrix
- dw: stores the change in weights after each iteration used for updating value of weights

Functions of class Neural Network:

- initialize_weights(): This function sets the values of weights to some random values between -0.5 to 0.5
- weights_creation(): According to the connections between neurons i.e. the incoming and outgoing edges the weights matrix is created.
- display_neuron(): displays the value stored in all the neurons in the netweoek
- fully_connected(): Creates the network of neurons and stores the incoming and outgoing edges to each neuron.
  First in the first input layer all the out edges for all neurons are set to connect with all neurons in first hidden layer
  Then for first hidden layer the in edges will be nodes in input layers and for last layer they will be output neuron otherwise for rest other neurons they will be from previous or next hidden layer.
  For the output layer the incoming edges are set to values of last hidden layer
  The case of one hidden layer is handled separately
- my_connections(): Taking input from user of the connections the incoming and outgoing edges are set accordingly similar to previous function however according to user input connections are set
- train(): This function first performs forward propagation and the according to values of error oobtained performs the back propagation.If iterations is given as input it return the weights at that no of iterations:



```
//feed forward
for(int j=0; j<data.size(); j++)
{
    //for one input row
    //layer by layer calculation of values
    for(int k=0; k<lay.size(); k++)
    {
        for(int l=0; l<lay[k].size(); l++)
        {
            lay[k][l].value=0;
            lay[k][l].in_value=0;
            lay[k][l].delta=0;
        }
    }
    for(int k=0; k<data[j].size(); k++)
    {
        lay[0][k].in_value=data[j][k];
        lay[0][k].value=data[j][k];
        //cout<<data[j][k]<<" ";
    }
    //cout<<endl;
    for(int k=1; k<layers; k++)
    {
        //find connections between them for layer k
        for(int m=0; m<weights[k-1].size(); m++)
        {
            for(int n=0; n<weights[k-1][m].size(); n++)
            {
                //neuron
                lay[k][n].in_value+=weights[k-1][m][n]*lay[k-1][m].value;
            }
        }
        //lay[k][n].in_value+=bias[k];
```



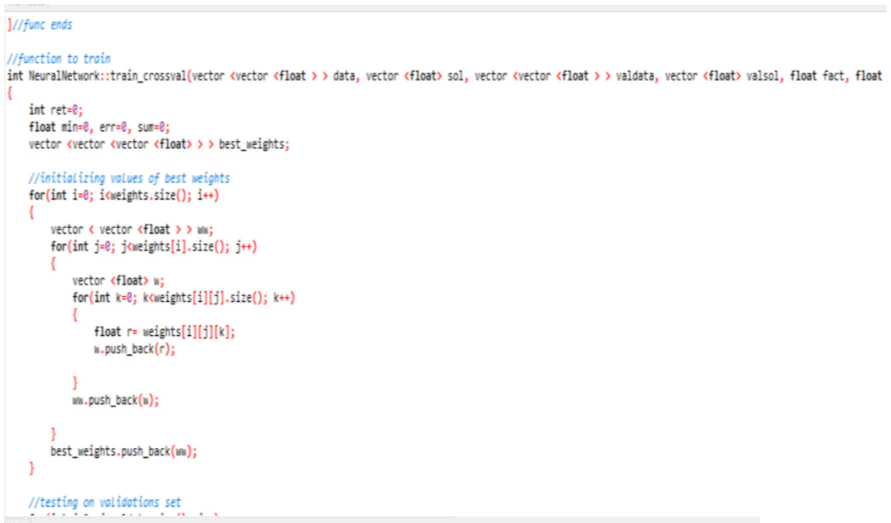
```
        //cout<<sol[j][k]<<endl;
        if(lay[layers-1][k].act==1)
        {
            lay[layers-1][k].delta=(sol[j]-lay[layers-1][k].value)*_dSigmoid(lay[layers-1][k].in_value);
        }
        else if(lay[layers-1][k].act==2)
        {
            lay[layers-1][k].delta=(sol[j]-lay[layers-1][k].value)*_dtanh(lay[layers-1][k].in_value);
        }
        else
        {
            lay[layers-1][k].delta=(sol[j]-lay[layers-1][k].value)*_dReLu(lay[layers-1][k].in_value);
        }

    }
    //change in weights for output layer
    for(int k=0; k<weights[layers-2].size(); k++)
    {
        for(int l=0; l<weights[layers-2][k].size(); l++)
        {
            dw[layers-2][k][l]=float(alpha*lay[layers-1][l].delta*lay[layers-1][l].value);
            //cout<<dw[layers-2][k][l]<<endl;
        }
    }

    //hidden layer
    //calculation of deltas and weights
    for(int k=layers-2; k>0; k--)
    {
        for(int l=0; l<lay[k].size(); l++)
        {
            float sum=0, val=0;
            for(int m=0; m<weights[k].size(); m++)
```

For the first input layer of neural network the values are being set equal to the input values of data obtained from file reading

Then the forward propagation is being performed.

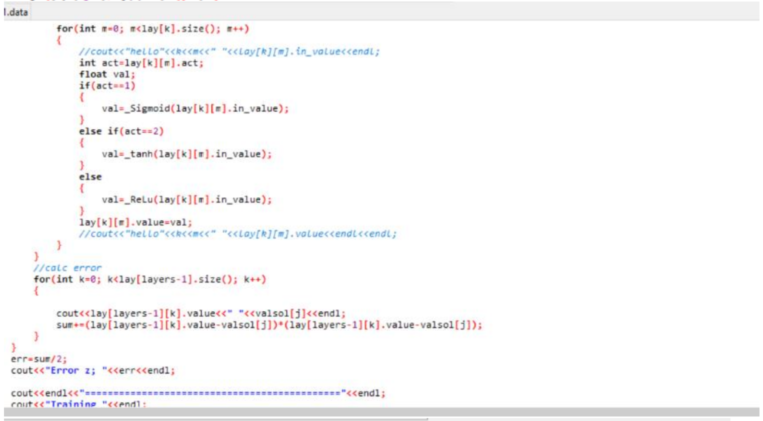
Backpropogation: The delta is being calculated for the output layer nodes and hidden layer neurons by backpropogating the error. The values of weights are changed according to the dw(i, j) values which are alpha*delta_j*derivative of activation function (z_in). delta_j is the summation of product of delta(k) and w(j, k). Then weights are updated by adding the dw to the old weights.

- train_crossval(): This function perform cross validation. It stores the best weights and keeps on updating them as training happens and it calculates the error on validation set each time. When the difference between error using best weights and current weights reaches crosses a threshold the value of weights in returned. It also retuens the no. of iterations which took place.



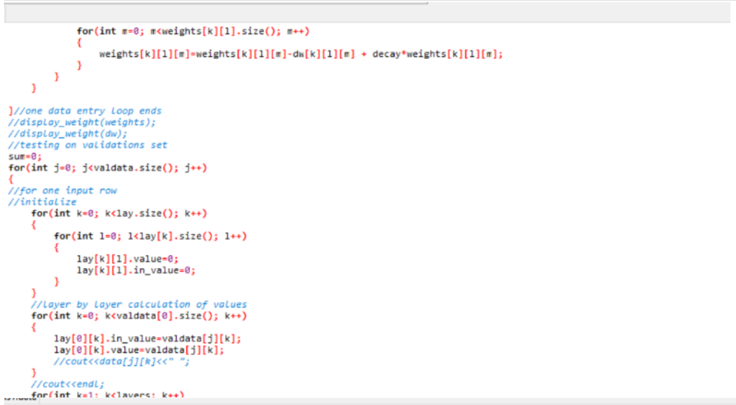
```
}//func ends

//function to train
int NeuralNetwork::train_crossval(vector <vector <float > > data, vector <float> sol, vector <vector <float > > valdata, vector <float> valsol, float fact, float
{
    int ret=0;
    float min=0, err=0, sum=0;
    vector <vector <vector <float> > > best_weights;

    //initializing values of best weights
    for(int i=0; i<weights.size(); i++)
    {
        vector < vector <float > > ww;
        for(int j=0; j<weights[i].size(); j++)
        {
            vector <float> w;
            for(int k=0; k<weights[i][j].size(); k++)
            {
                float r= weights[i][j][k];
                w.push_back(r);

            }
            ww.push_back(w);

        }
        best_weights.push_back(ww);
    }

    //testing on validations set
```

Initializing the values of best weights (global variable)



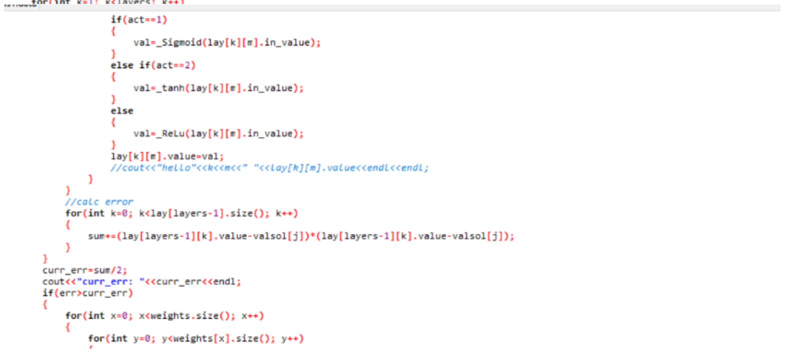
```
            for(int m=0; m<lay[k].size(); m++)
            {
                //cout<<"hello"<<k<<m<<" "<<lay[k][m].in_value<<endl;
                int act=lay[k][m].act;
                float val;
                if(act==1)
                {
                    val=_Sigmoid(lay[k][m].in_value);
                }
                else if(act==2)
                {
                    val=_tanh(lay[k][m].in_value);
                }
                else
                {
                    val=_ReLu(lay[k][m].in_value);
                }
                lay[k][m].value=val;
                //cout<<"hello"<<k<<m<<" "<<lay[k][m].value<<endl<<endl;
            }
        }
        //calc error
        for(int k=0; k<lay[layers-1].size(); k++)
        {

            cout<<lay[layers-1][k].value<<" "<<valsol[j]<<endl;
            sum+=(lay[layers-1][k].value-valsol[j])*(lay[layers-1][k].value-valsol[j]);
        }
    }
    err=sum/2;
    cout<<"Error z; "<<err<<endl;

    cout<<endl<<"============================================="<<endl;
    cout<<"Training "<<endl;
```

Training is being on train_data



```
                for(int m=0; m<weights[k][l].size(); m++)
                {
                    weights[k][l][m]=weights[k][l][m]-dw[k][l][m] + decay*weights[k][l][m];
                }
            }
        }

}//one data entry loop ends
//display_weight(weights);
//display_weight(dw);
//testing on validations set
sum=0;
for(int j=0; j<valdata.size(); j++)
{
//for one input row
//initialize
    for(int k=0; k<lay.size(); k++)
    {
        for(int l=0; l<lay[k].size(); l++)
        {
            lay[k][l].value=0;
            lay[k][l].in_value=0;
        }
    }
    //layer by layer calculation of values
    for(int k=0; k<valdata[0].size(); k++)
    {
        lay[0][k].in_value=valdata[j][k];
        lay[0][k].value=valdata[j][k];
        //cout<<data[j][k]<<" ";
    }
    //cout<<endl;
    for(int k=1; k<layers; k++)
```

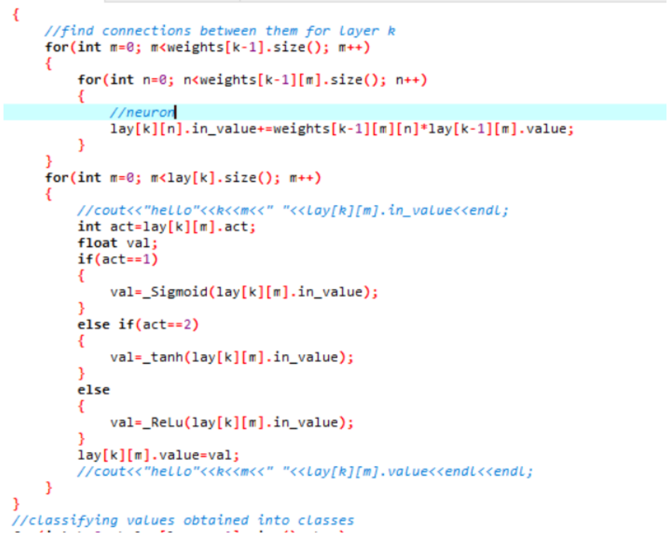
After one iteration the error is calculated on the validation set



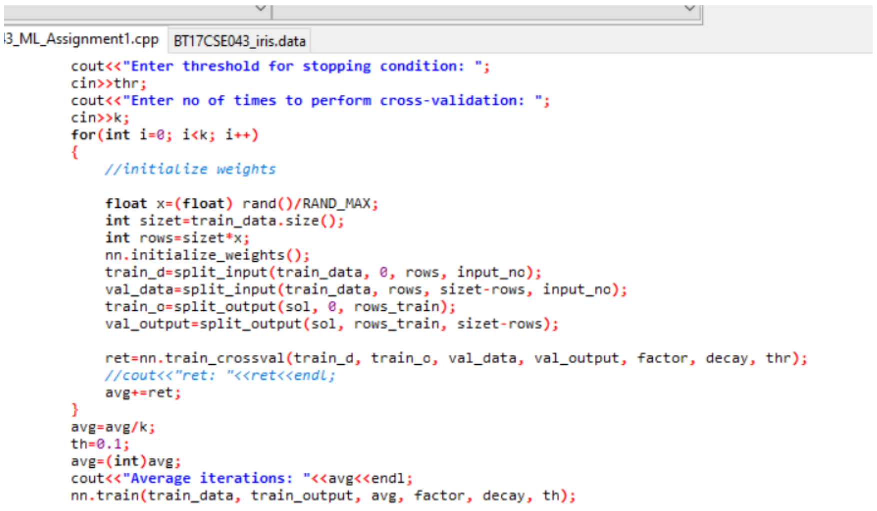
```
                if(act==1)
                {
                    val=_Sigmoid(lay[k][m].in_value);
                }
                else if(act==2)
                {
                    val=_tanh(lay[k][m].in_value);
                }
                else
                {
                    val=_ReLu(lay[k][m].in_value);
                }
                lay[k][m].value=val;
                //cout<<"hello"<<k<<m<<" "<<lay[k][m].value<<endl<<endl;
            }
        }
        //calc error
        for(int k=0; k<lay[layers-1].size(); k++)
        {
            sum+=(lay[layers-1][k].value-valsol[j])*(lay[layers-1][k].value-valsol[j]);
        }
    }
    curr_err=sum/2;
    cout<<"curr_err: "<<curr_err<<endl;
    if(err>curr_err)
    {
        for(int x=0; x<weights.size(); x++)
        {
            for(int y=0; y<weights[x].size(); y++)
```

Best weights are updated if current error is less and the function returns the no of iterations it performed when a threshold was reached as mentioned earlier

- test(): This function calculates the values for each input row by using the weights obtained in training and then according calculates the error.



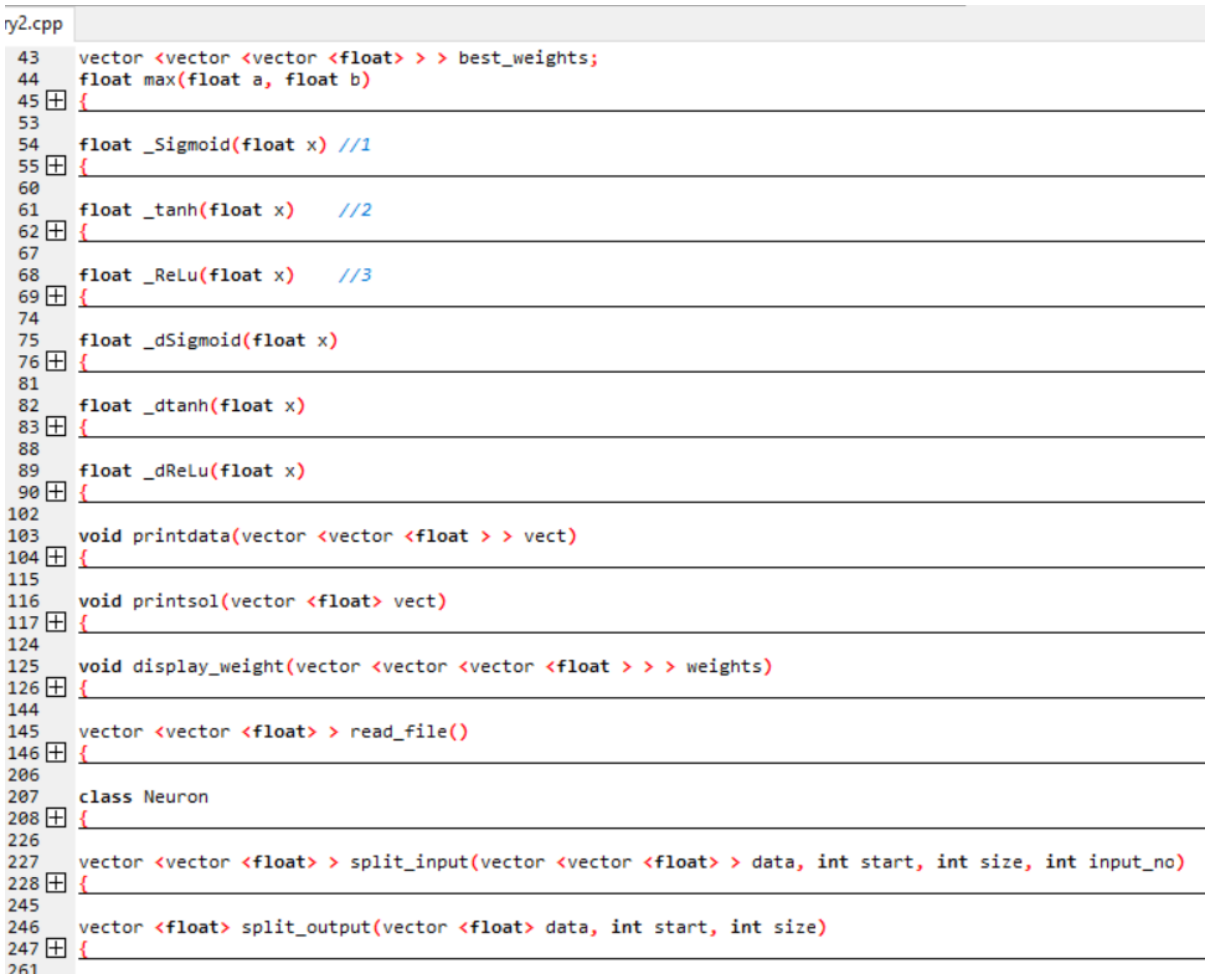
```
{
    //find connections between them for layer k
    for(int m=0; m<weights[k-1].size(); m++)
    {
        for(int n=0; n<weights[k-1][m].size(); n++)
        {
            //neuron
            lay[k][n].in_value+=weights[k-1][m][n]*lay[k-1][m].value;
        }
    }
    for(int m=0; m<lay[k].size(); m++)
    {
        //cout<<"hello"<<k<<m<<" "<<lay[k][m].in_value<<endl;
        int act=lay[k][m].act;
        float val;
        if(act==1)
        {
            val=_Sigmoid(lay[k][m].in_value);
        }
        else if(act==2)
        {
            val=_tanh(lay[k][m].in_value);
        }
        else
        {
            val=_ReLu(lay[k][m].in_value);
        }
        lay[k][m].value=val;
        //cout<<"hello"<<k<<m<<" "<<lay[k][m].value<<endl<<endl;
    }
}
//classifying values obtained into classes
```



3_ML_Assignment1.cpp BT17CSE043_iris.data

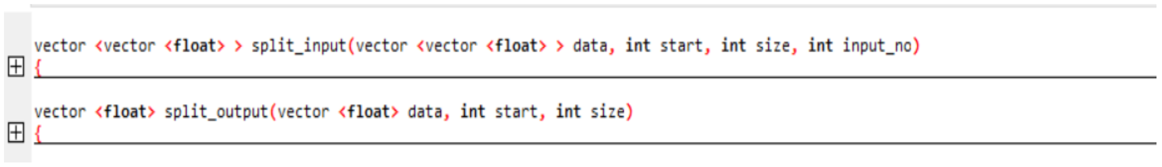
```
        cout<<"Enter threshold for stopping condition: ";
        cin>>thr;
        cout<<"Enter no of times to perform cross-validation: ";
        cin>>k;
        for(int i=0; i<k; i++)
        {
            //initialize weights

            float x=(float) rand()/RAND_MAX;
            int sizet=train_data.size();
            int rows=sizet*x;
            nn.initialize_weights();
            train_d=split_input(train_data, 0, rows, input_no);
            val_data=split_input(train_data, rows, sizet-rows, input_no);
            train_o=split_output(sol, 0, rows_train);
            val_output=split_output(sol, rows_train, sizet-rows);

            ret=nn.train_crossval(train_d, train_o, val_data, val_output, factor, decay, thr);
            //cout<<"ret: "<<ret<<endl;
            avg+=ret;
        }
        avg=avg/k;
        th=0.1;
        avg=(int)avg;
        cout<<"Average iterations: "<<avg<<endl;
        nn.train(train_data, train_output, avg, factor, decay, th);

    }
```

To perform k-fold cross validation: data is split k times into different sets at random and no of iterations are calculated in each case and the average is calculated. Each time cross validation is applied and then the entire training set is used for weight calculation for that average no. of iterations. Each time we initialize weights to avoid any bias since weights are associated with the neural network.



```
ry2.cpp
vector <vector <vector <float> > > best_weights;
float max(float a, float b)
{
float _Sigmoid(float x) //1
{
float _tanh(float x)    //2
{
float _ReLu(float x)    //3
{
float _dSigmoid(float x)
{
float _dtanh(float x)
{
float _dReLu(float x)
{
void printdata(vector <vector <float > > vect)
{
void printsol(vector <float> vect)
{
void display_weight(vector <vector <vector <float > > > weights)
{
vector <vector <float> > read_file()
{
class Neuron
{
vector <vector <float> > split_input(vector <vector <float> > data, int start, int size, int input_no)
{
vector <float> split_output(vector <float> data, int start, int size)
{
```

- max(): Takes input two float nos. and returns max out of them
- _Sigmoid(): Takes input a float value and return its sigmoid values by formula1/1+e^(-x)
- _tanh(): Takes input a float value and return its tanh values.
- _ReLu(): Takes input a float value and return same no if value is greater than 0 else return 0.
- _dSigmoid(): Takes input a float value and return its sigmoid values by formula: _Sigmoid(x)*(1-_Sigmoid(x))
- _dtanh(): Takes input a float value and return its sigmoid values by formula: (1-_tanh(x)*_tanh(x))
- _dReLu(): Takes input a float value and return 1 if value is greater than 0 else return 0
- printdata (): prints a vector of vector of floats
- display_weight (): prints vector of vector of vector of floats
- read_file (): Reads a file and output the values (decimals also) and created a 2-D matrix of those values and returns the vector of vector of floats.
-

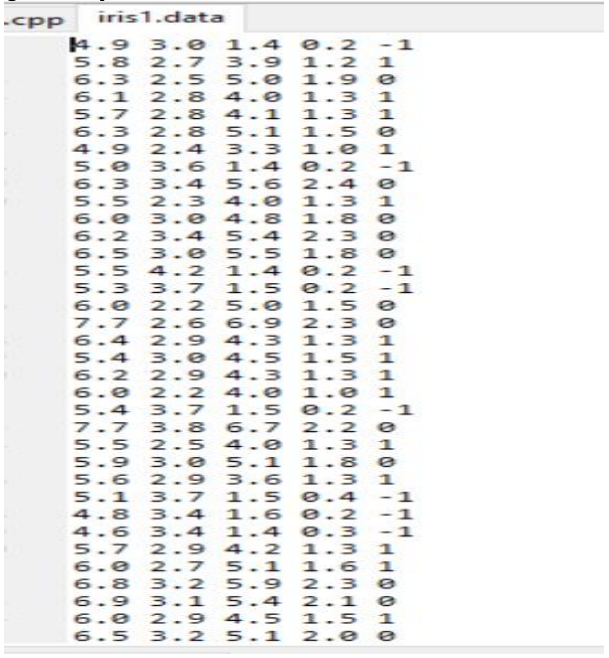


```
vector <vector <float> > split_input(vector <vector <float> > data, int start, int size, int input_no)
{

vector <float> split_output(vector <float> data, int start, int size)
{
```

- split_input(): Takes a 2d matrix as input along with the starting point and the size and return the matrix starting from the starting point with size given as input to the function(used in validating)
- split_output():Takes a 1d matrix as input along with the starting point and the size and return the vector starting from the starting point with size given as input to the function (used in validating)

5. Input format

The input file is taken as the standard iris data consisting of 150 rows and 3 classes. First the rows are randomly shuffles and then feed to the neural network.
The classes are 1 , 0 and -1.
Output unit is considered as one neuron which on which tanh activation function is used since it return values between -1 to 1.
When a certain threshold say > 0.2 value is obtained then we classify that row of input in class 1. If value obtained is between -0.2 to 0.2 we classify it as class 0 and if <-0.2 we classify it as class -1.
The activation function is given by #define OUTPUT_ACT . On changing the value of activation function we can change the conditions of classifying data accordingly as the range of values given by each activation function is different

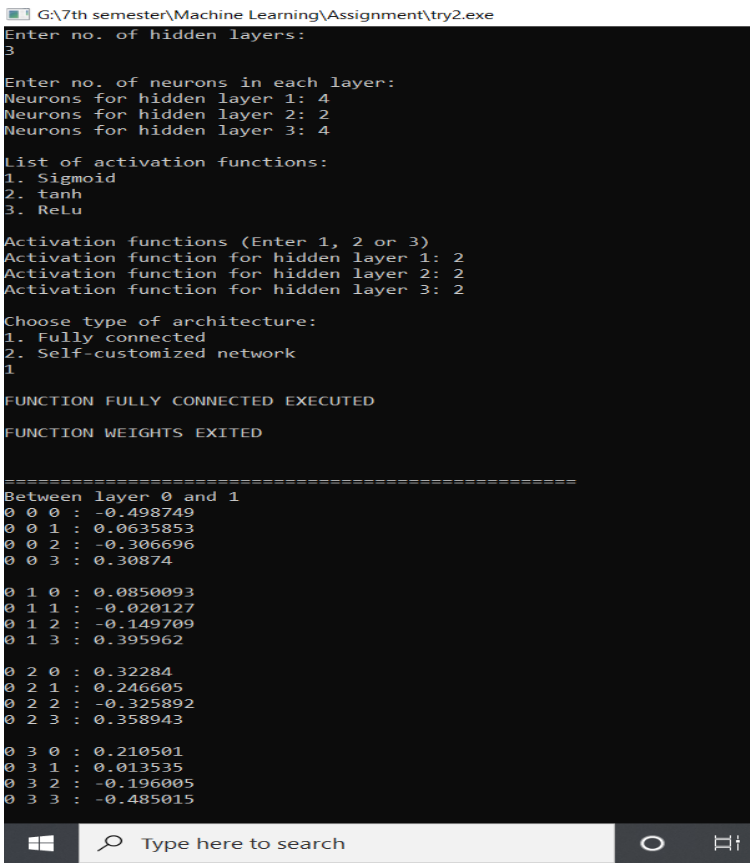


```
4.9 3.0 1.4 0.2 -1
5.8 2.7 3.9 1.2 1
6.3 2.5 5.0 1.9 0
6.1 2.8 4.0 1.3 1
5.7 2.8 4.1 1.3 1
6.3 2.8 5.1 1.5 0
4.9 2.4 3.3 1.0 1
5.0 3.6 1.4 0.2 -1
6.3 3.4 5.6 2.4 0
5.5 2.3 4.0 1.3 1
6.0 3.0 4.8 1.8 0
6.2 3.4 5.4 2.3 0
6.5 3.0 5.5 1.8 0
5.5 4.2 1.4 0.2 -1
5.3 3.7 1.5 0.2 -1
6.0 2.2 5.0 1.5 0
7.7 2.6 6.9 2.3 0
6.4 2.9 4.3 1.3 1
5.4 3.0 4.5 1.5 1
6.2 2.9 4.3 1.3 1
6.0 2.2 4.0 1.0 1
5.4 3.7 1.5 0.2 -1
7.7 3.8 6.7 2.2 0
5.5 2.5 4.0 1.3 1
5.9 3.0 5.1 1.8 0
5.6 2.9 3.6 1.3 1
5.1 3.7 1.5 0.4 -1
4.8 3.4 1.6 0.2 -1
4.6 3.4 1.4 0.3 -1
5.7 2.9 4.2 1.3 1
6.0 2.7 5.1 1.6 1
6.8 3.2 5.9 2.3 0
6.9 3.1 5.4 2.1 0
6.0 2.9 4.5 1.5 1
6.5 3.2 5.1 2.0 0
```

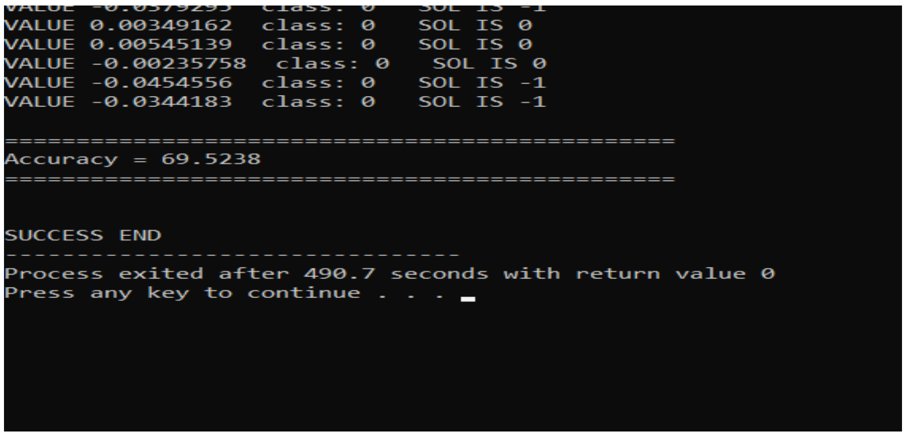
The no of features in this case are 4 and the output class is given by the last column

5. Output

On running the code on the following input:

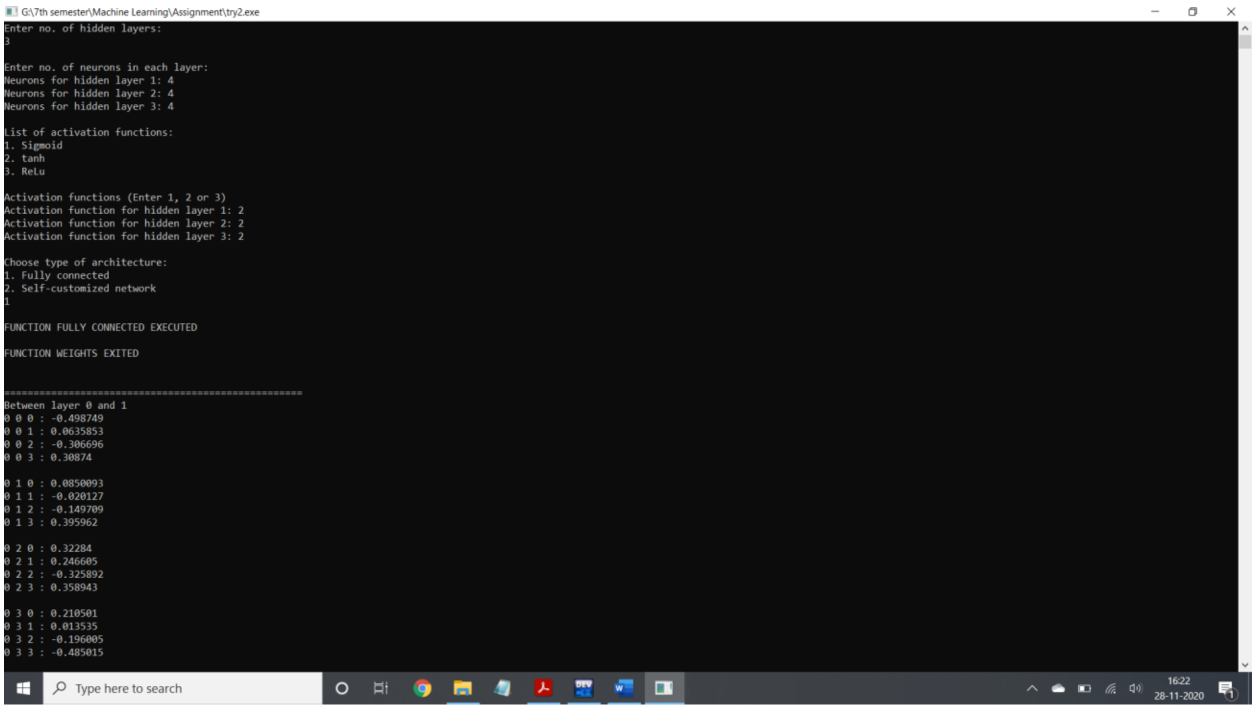


```
G:\7th semester\Machine Learning\Assignment\try2.exe
Enter no. of hidden layers:
3

Enter no. of neurons in each layer:
Neurons for hidden layer 1: 4
Neurons for hidden layer 2: 2
Neurons for hidden layer 3: 4

List of activation functions:
1. Sigmoid
2. tanh
3. ReLu

Activation functions (Enter 1, 2 or 3)
Activation function for hidden layer 1: 2
Activation function for hidden layer 2: 2
Activation function for hidden layer 3: 2

Choose type of architecture:
1. Fully connected
2. Self-customized network
1

FUNCTION FULLY CONNECTED EXECUTED

FUNCTION WEIGHTS EXITED


===================================================
Between layer 0 and 1
0 0 0 : -0.498749
0 0 1 : 0.0635853
0 0 2 : -0.306696
0 0 3 : 0.30874

0 1 0 : 0.0850093
0 1 1 : -0.020127
0 1 2 : -0.149709
0 1 3 : 0.395962

0 2 0 : 0.32284
0 2 1 : 0.246605
0 2 2 : -0.325892
0 2 3 : 0.358943

0 3 0 : 0.210501
0 3 1 : 0.013535
0 3 2 : -0.196005
0 3 3 : -0.485015
```



```
VALUE 0.00349162  class: 0   SOL IS 0
VALUE 0.00545139  class: 0   SOL IS 0
VALUE -0.00235758  class: 0   SOL IS 0
VALUE -0.0454556  class: 0   SOL IS -1
VALUE -0.0344183  class: 0   SOL IS -1

===============================================
Accuracy = 69.5238
===============================================


SUCCESS END
--------------------------------
Process exited after 490.7 seconds with return value 0
Press any key to continue . . .
```

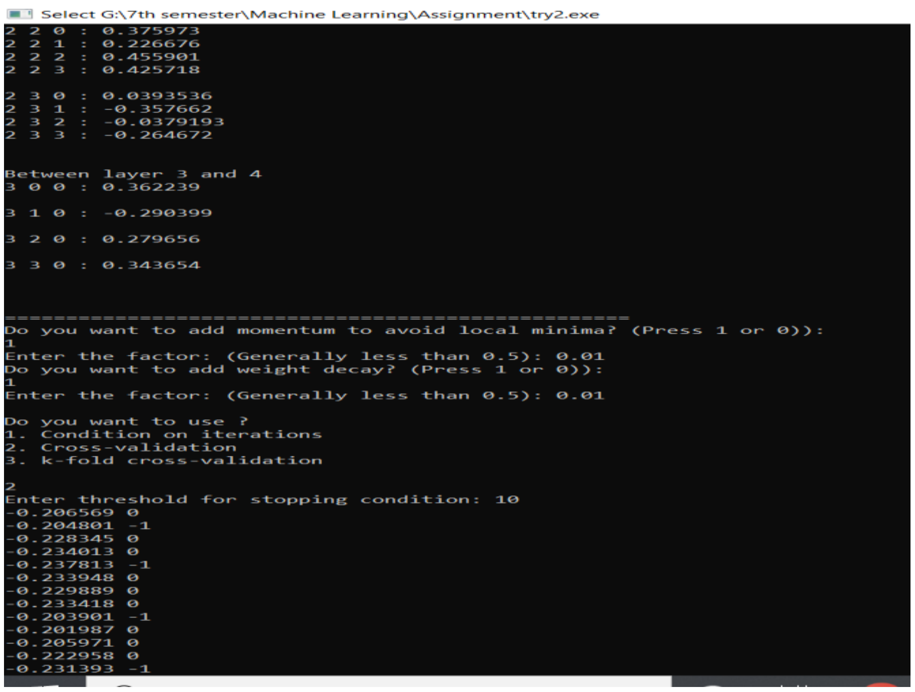
Accuracy of was **69.52%** obtained which is good considering there are 3 output classes further on increasing no of iterations and changing hyper parameters better results were obtained.



G:\7th semester\Machine Learning\Assignment\try2.exe

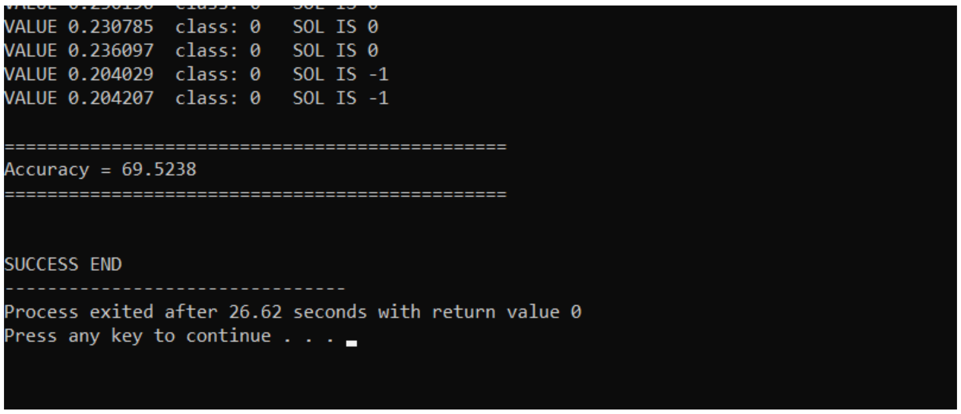
```
Enter no. of hidden layers:
3

Enter no. of neurons in each layer:
Neurons for hidden layer 1: 4
Neurons for hidden layer 2: 4
Neurons for hidden layer 3: 4

List of activation functions:
1. Sigmoid
2. tanh
3. ReLu

Activation functions (Enter 1, 2 or 3)
Activation function for hidden layer 1: 2
Activation function for hidden layer 2: 2
Activation function for hidden layer 3: 2

Choose type of architecture:
1. Fully connected
2. Self-customized network
1

FUNCTION FULLY CONNECTED EXECUTED

FUNCTION WEIGHTS EXITED


===================================================
Between layer 0 and 1
0 0 0 : -0.498749
0 0 1 : 0.0635853
0 0 2 : -0.306696
0 0 3 : 0.30874

0 1 0 : 0.0850093
0 1 1 : -0.020127
0 1 2 : -0.149709
0 1 3 : 0.395962

0 2 0 : 0.32284
0 2 1 : 0.246605
0 2 2 : -0.325892
0 2 3 : 0.358943

0 3 0 : 0.210501
0 3 1 : 0.013535
0 3 2 : -0.196005
0 3 3 : -0.485015
```

16:22
28-11-2020



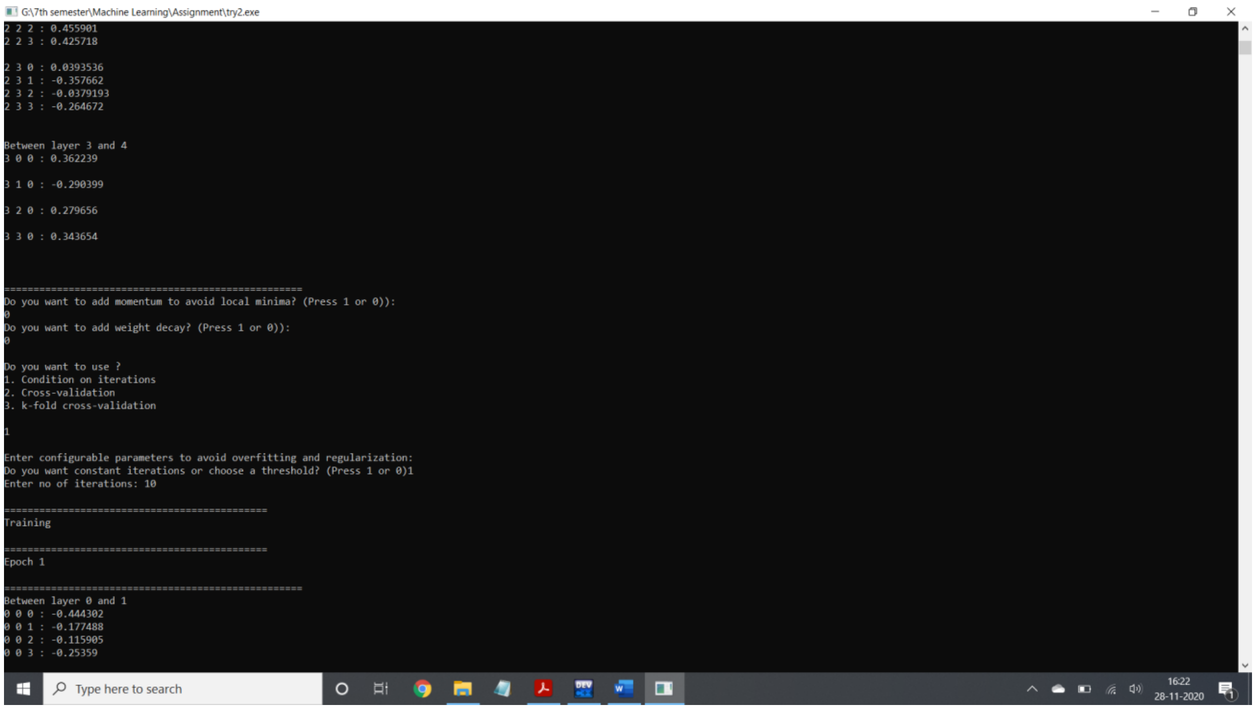
Select G:\7th semester\Machine Learning\Assignment\try2.exe

```
2 2 0 : 0.375973
2 2 1 : 0.226676
2 2 2 : 0.455901
2 2 3 : 0.425718

2 3 0 : 0.0393536
2 3 1 : -0.357662
2 3 2 : -0.0379193
2 3 3 : -0.264672


Between layer 3 and 4
3 0 0 : 0.362239

3 1 0 : -0.290399

3 2 0 : 0.279656

3 3 0 : 0.343654



===================================================
Do you want to add momentum to avoid local minima? (Press 1 or 0)):
1
Enter the factor: (Generally less than 0.5): 0.01
Do you want to add weight decay? (Press 1 or 0)):
1
Enter the factor: (Generally less than 0.5): 0.01

Do you want to use ?
1. Condition on iterations
2. Cross-validation
3. k-fold cross-validation

2
Enter threshold for stopping condition: 10
-0.206569 0
-0.204801 -1
-0.228345 0
-0.234013 0
-0.237813 -1
-0.233948 0
-0.229889 0
-0.233418 0
-0.203901 -1
-0.201987 0
-0.205971 0
-0.222958 0
-0.231393 -1
```



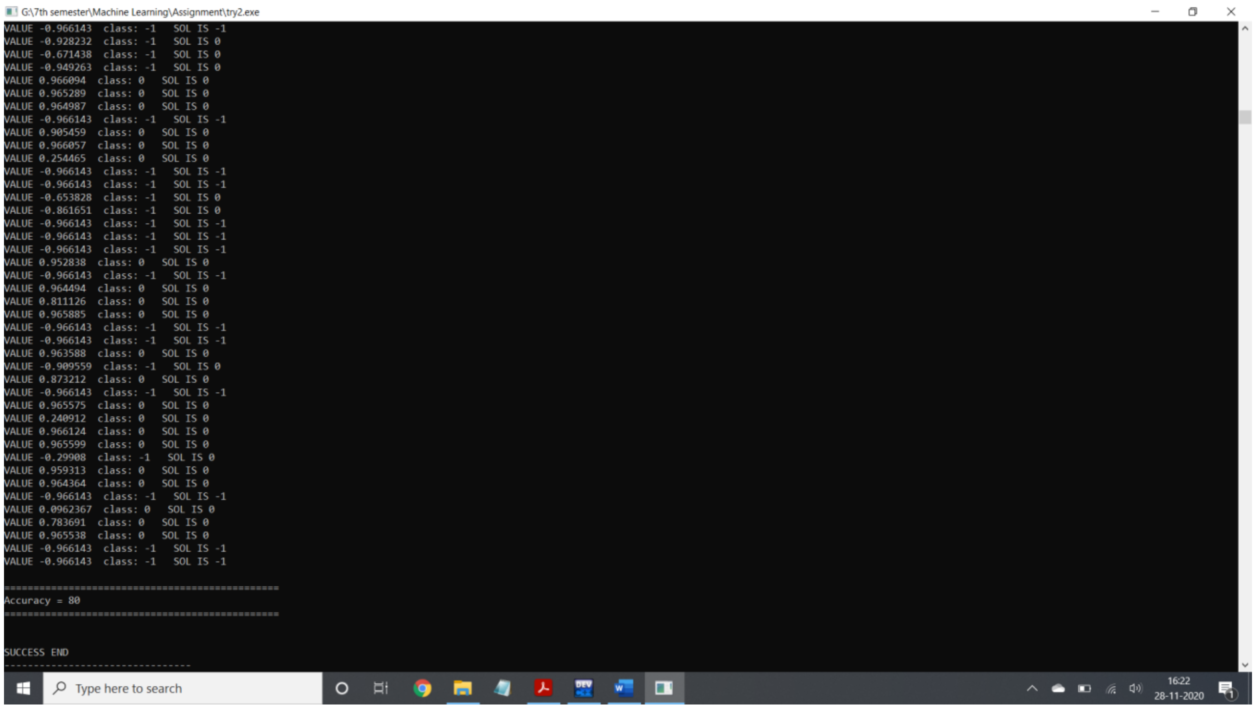
```
VALUE 0.230785  class: 0   SOL IS 0
VALUE 0.236097  class: 0   SOL IS 0
VALUE 0.204029  class: 0   SOL IS -1
VALUE 0.204207  class: 0   SOL IS -1

===============================================
Accuracy = 69.5238
===============================================


SUCCESS END
--------------------------------
Process exited after 26.62 seconds with return value 0
Press any key to continue . . .
```

On using cross validation accuracy was **69.52%**



G:\7th semester\Machine Learning\Assignment\try2.exe

```
2 2 2 : 0.455901
2 2 3 : 0.425718

2 3 0 : 0.0393536
2 3 1 : -0.357662
2 3 2 : -0.0379193
2 3 3 : -0.264672


Between layer 3 and 4
3 0 0 : 0.362239

3 1 0 : -0.290399

3 2 0 : 0.279656

3 3 0 : 0.343654



===================================================
Do you want to add momentum to avoid local minima? (Press 1 or 0)):
0
Do you want to add weight decay? (Press 1 or 0)):
0

Do you want to use ?
1. Condition on iterations
2. Cross-validation
3. k-fold cross-validation

1

Enter configurable parameters to avoid overfitting and regularization:
Do you want constant iterations or choose a threshold? (Press 1 or 0)1
Enter no of iterations: 10

=============================================
Training

=============================================
Epoch 1

===================================================
Between layer 0 and 1
0 0 0 : -0.444302
0 0 1 : -0.177488
0 0 2 : -0.115905
0 0 3 : -0.25359
```



```
VALUE -0.966143  class: -1   SOL IS -1
VALUE -0.928232  class: -1   SOL IS 0
VALUE -0.671438  class: -1   SOL IS 0
VALUE -0.949263  class: -1   SOL IS 0
VALUE 0.966094  class: 0   SOL IS 0
VALUE 0.965289  class: 0   SOL IS 0
VALUE 0.964987  class: 0   SOL IS 0
VALUE -0.966143  class: -1   SOL IS -1
VALUE 0.905459  class: 0   SOL IS 0
VALUE 0.966057  class: 0   SOL IS 0
VALUE 0.254465  class: 0   SOL IS 0
VALUE -0.966143  class: -1   SOL IS -1
VALUE -0.966143  class: -1   SOL IS -1
VALUE -0.653828  class: -1   SOL IS 0
VALUE -0.861651  class: -1   SOL IS 0
VALUE -0.966143  class: -1   SOL IS -1
VALUE -0.966143  class: -1   SOL IS -1
VALUE -0.966143  class: -1   SOL IS -1
VALUE 0.952838  class: 0   SOL IS 0
VALUE -0.966143  class: -1   SOL IS -1
VALUE 0.964494  class: 0   SOL IS 0
VALUE 0.811126  class: 0   SOL IS 0
VALUE 0.965885  class: 0   SOL IS 0
VALUE -0.966143  class: -1   SOL IS -1
VALUE -0.966143  class: -1   SOL IS -1
VALUE 0.963588  class: 0   SOL IS 0
VALUE -0.909559  class: -1   SOL IS 0
VALUE 0.873212  class: 0   SOL IS 0
VALUE -0.966143  class: -1   SOL IS -1
VALUE 0.965575  class: 0   SOL IS 0
VALUE 0.240912  class: 0   SOL IS 0
VALUE 0.966124  class: 0   SOL IS 0
VALUE 0.965599  class: 0   SOL IS 0
VALUE -0.29908  class: -1   SOL IS 0
VALUE 0.959313  class: 0   SOL IS 0
VALUE 0.964364  class: 0   SOL IS 0
VALUE -0.966143  class: -1   SOL IS -1
VALUE 0.0962367  class: 0   SOL IS 0
VALUE 0.783691  class: 0   SOL IS 0
VALUE 0.965538  class: 0   SOL IS 0
VALUE -0.966143  class: -1   SOL IS -1
VALUE -0.966143  class: -1   SOL IS -1

==================================================
Accuracy = 80
==================================================


SUCCESS END
--------------------------------
```

In this case we obtained the Accuracy of **80%** when we change the no of iterations to 10.

6. Conclusions

On changing values of hidden layers, activation functions type of validation to be used, no of iterations etc. we get different results. Also if the size of dataset was small saturation was obtained i.e. overfitting and on test data results obtained were not good Cross validation and k-fold cross validation were some methods to stop that. In which we broke down the training data into validation set and training set and error was calculated on validation set in each iteration. Also a difference in results was seen when weight decay or momentum was used. Momentum converged the result quickly whereas weight decay reduced. the overfitting from happening. Also since the code is very generalized easily we can modify parameters, add bias to each layer or add some more methods like drop node as future enhancements. However the person using the code must have knowledge on what values of threshold parameters to set for obtaining best results or this can be learnt by experimentation.