KHADAKE VARUN DILIP

214103001

M.TECH. - A&P

Soft Computing

Coding Assignment – Report

Multi-Layer Feed Forward Neural Network with Back Propagation Training Algorithm

Introduction:

In this assignment, the application of artificial neural networks (ANNs) for predicting the Mach number downstream of the shock (M2) for the given upstream Mach number (M1) and corner angle (θ).

An oblique shock wave, in contrast to normal shock wave, is a shock wave that is inclined with respect to the incident upstream flow direction. It occurs when a supersonic flow encounters a corner that effectively turns the flow into itself and compresses.

The upstream streamlines are uniformly deflected after the shock wave. The most common way to produce an oblique shock wave is to place a wedge into supersonic, compressible flow. Similar to a normal shock wave, the oblique shock wave consists of a very thin region across which nearly discontinuous changes in the thermodynamic properties of a gas occur.

The upstream and downstream flow directions are unchanged across a normal shock. However, they are different for flow across an oblique shock wave. Unlike after a normal shock where M2 must always be less than 1, in oblique shock M2 can be supersonic (weak shock wave) or subsonic (strong shock wave). Weak solutions are often observed in flow geometries open to atmosphere (such as on the outside of a flight vehicle). Strong solutions may be observed in confined geometries (such as inside a nozzle intake)

For our ANN model we are using two parameters – upstream Mach number (M1) and corner angle (θ) to find the Mach number downstream of the shock (M2).

ANN model is trained with 210 data-points out of the total of 300 data-points, (70% of the total) and the validation & testing of the trained ANN were performed with the remaining 90 data-points.

The θ-β-M equation:

Using the continuity equation and the fact that the tangential velocity component does not change across the shock, trigonometric relations eventually lead to the θ-β-M equation which shows θ as a function of M1 β, and ɣ, where ɣ is the Heat capacity ratio.

$$\tan\theta = 2\cot\beta\frac{M_1^2\sin^2\beta - 1}{M_1^2(\gamma + \cos 2\beta) + 2}$$

Data:

Initial data is generated using the θ-β-M equation.

Inputs and output used in the ANN are:

Variable Representation	Variable Name
X1 (input)	Upstream Mach no. (M1)
X2 (input)	Corner angel (θ)
Y1 (output)	Downstream Mach Number (M2)

Methodology:

For the above parameters, a fully connected, feed forward, back propagating, 3-layered ANN model was used to predict the two outputs based on the eight inputs.

The key features of the model are:

1. 210 patterns were taken as the training data while the remaining 90 patterns were used to verify the accuracy of the trained model.
2. Transfer functions in the hidden layer were taken as log-sigmoid with constant 'a' as 1 i.e.

$$f(x) = \frac{1}{1 + e^{-x}}$$

3. Transfer functions in the output layer were taken as log-sigmoid with constant 'a' as 1 i.e.

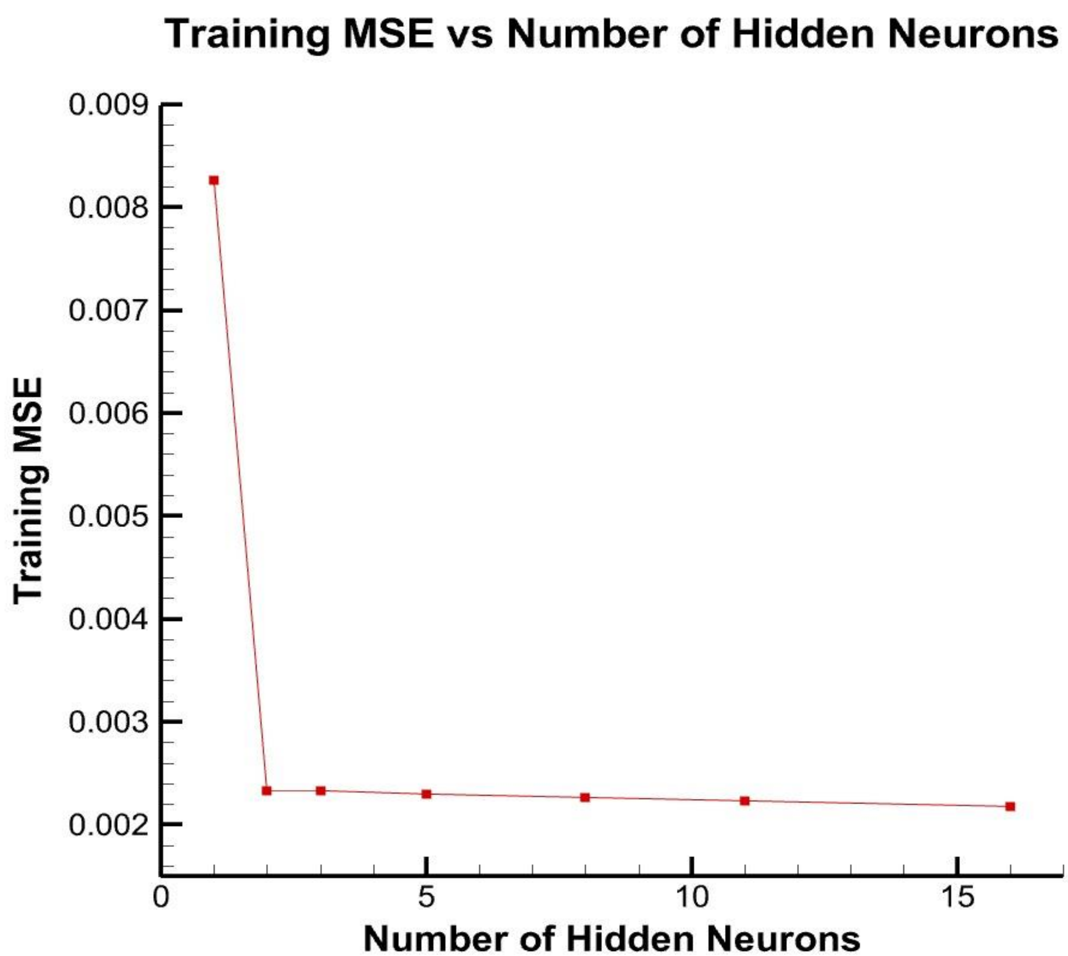
$$f(x) = \frac{1}{1 + e^{-x}}$$

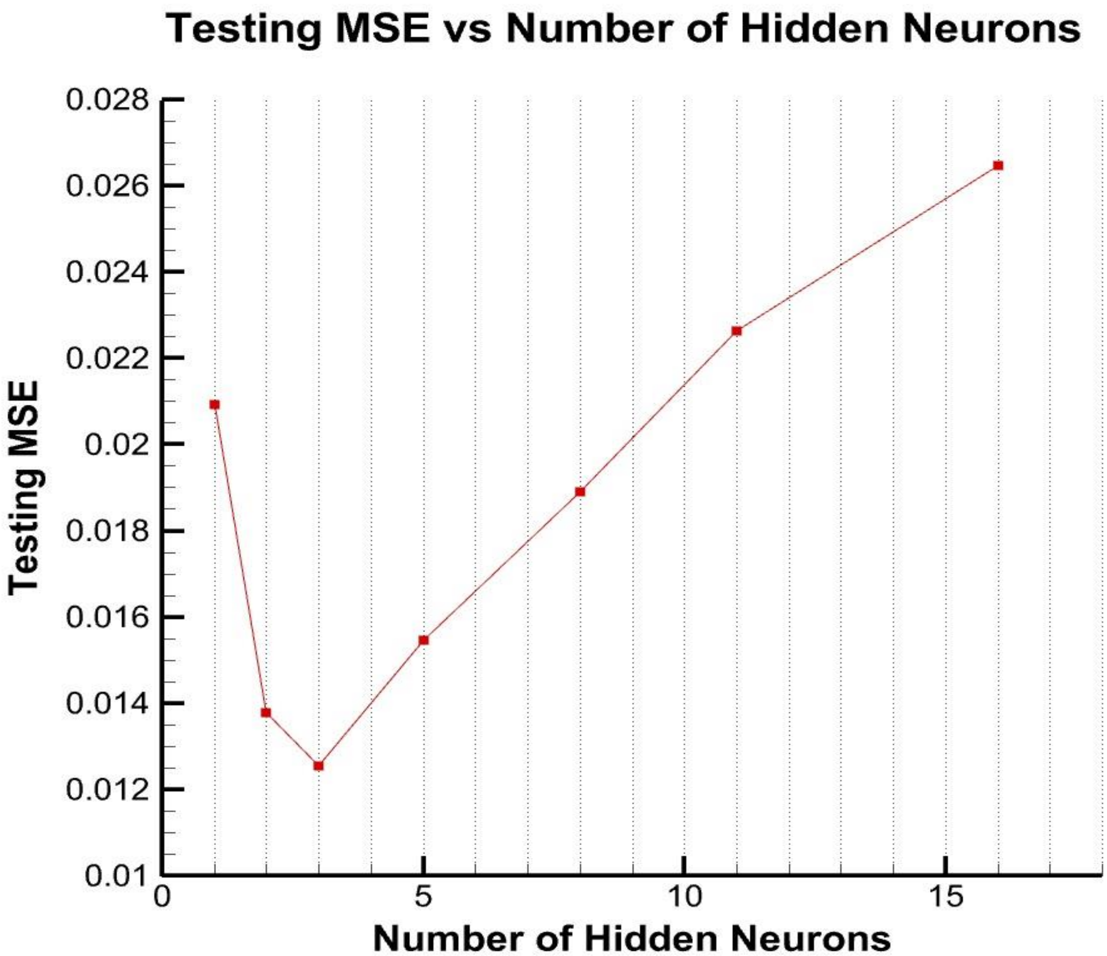
4. The input data and output data was normalized between the limits 0.1 to 0.9 in order to accommodate their values within the range of the transfer function.
5. The end condition of training was set to be mean square error less than 0.001 or maximum number of iterations less than or equal to 1,00,000 as the error curve flattens after reaching a certain point.

6. The final number of neurons in the hidden layer was taken as ‘3’ based on the conclusion obtained after experimenting with different number of hidden neurons and observing the respective MSE.
7. Similarly the final value of learning rate was chosen to be ‘0.5’.

Results:

Using the experimental trial and error analysis, the optimum number of hidden neurons and learning rate was found taking the lowest mean square error as the principle criteria.

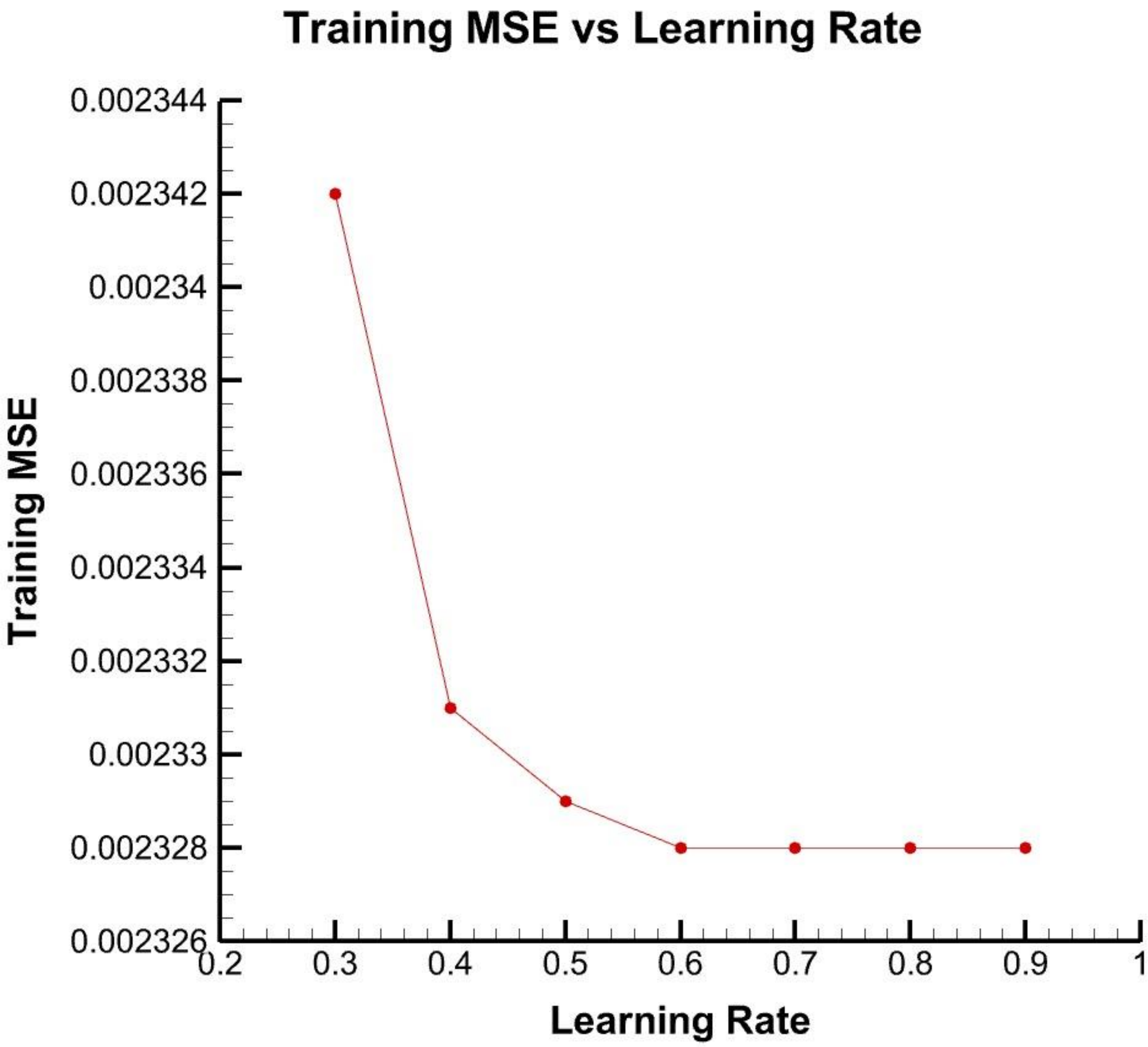


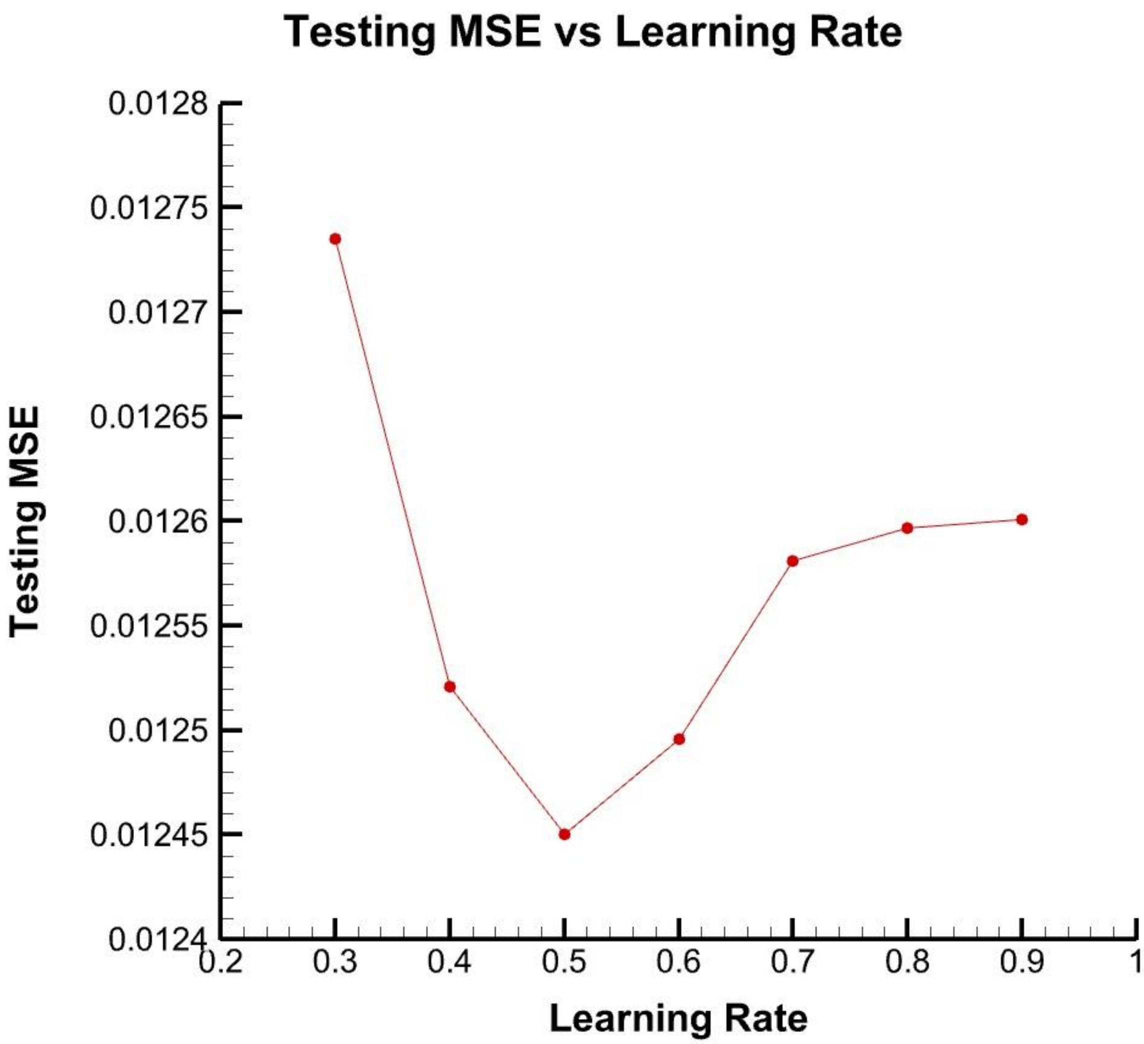


As the number of hidden neurons increases, the final mean square error decreases.

Increase in the number of hidden neurons does smoothen the curve and there are fewer oscillations for the same values of learning rate.

As proper fitting of both testing and training is obtained at 3 hidden neurons, the optimum no. of hidden neurons is 3.

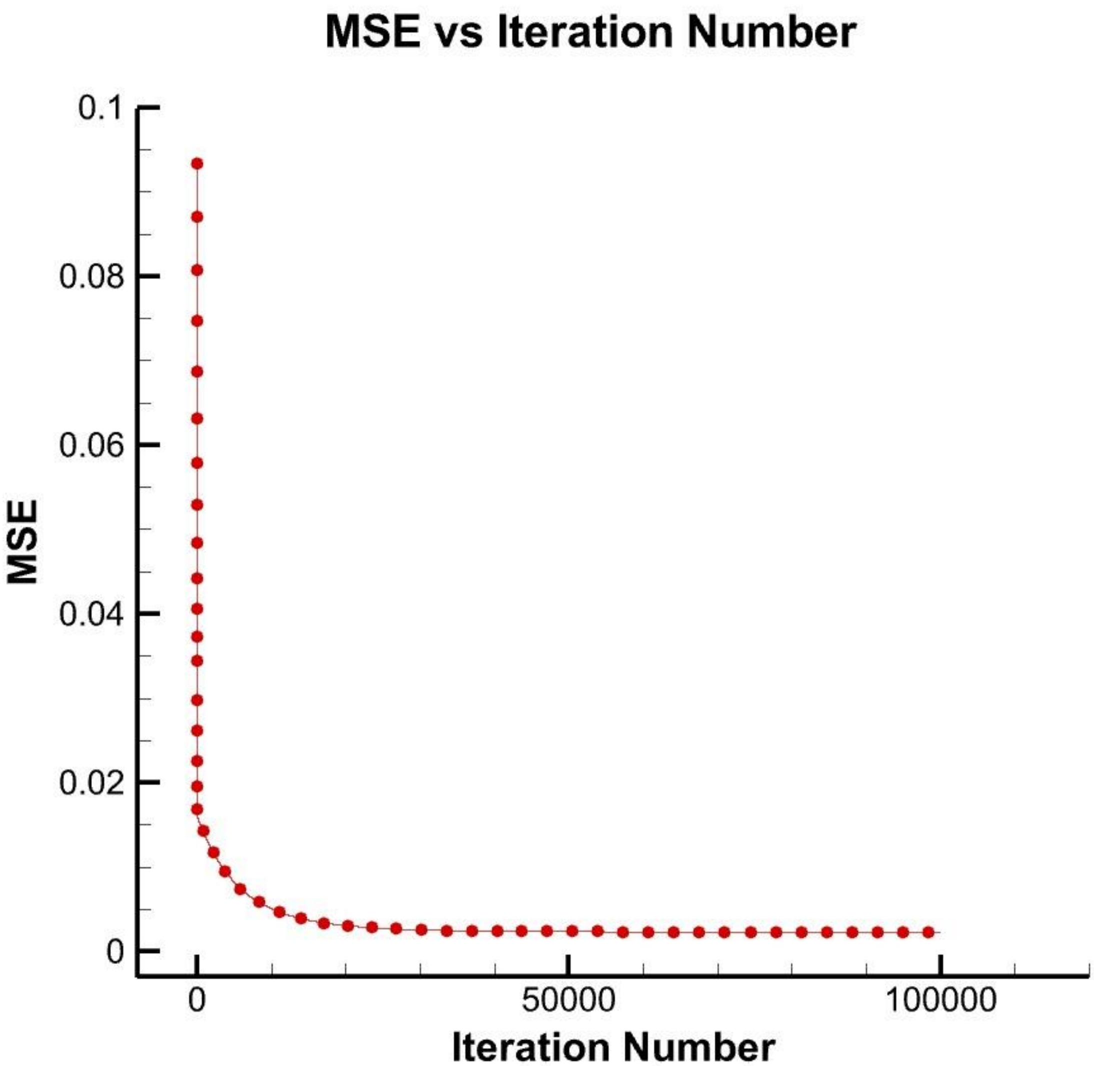

Training MSE vs Learning Rate
0.002344
0.002342
0.00234
0.002338
0.002336
0.002334
0.002332
0.00233
0.002328
0.002326
Training MSE
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Learning Rate



As the value of learning rate (eta) increases, the minima is reached faster i.e. fewer iterations are required.

The mean square error flattens at nearly 0.00239 with negligible change as iterations progress after for eta = 0.5.

Optimum eta was taken as 0.5 as it gives minimum mean square error.



Conclusion:

A fully connected feed forward back propagating 3 layered ANN model is used to predict Mach number downstream of the shock (M2) for the given upstream Mach number (M1) and corner angle (θ). The ANN has been trained using 210 training patterns and 90 testing patterns with 3 hidden layer neurons. The learning rate is set to 0.5. Training provides a mean square error of 0.002 and this trained ANN can be used to predict oblique shock wave angle for given inputs with a mean square error of 0.012.

Code:

[P.T.O.]

```c
#include <stdio.h>
#include<conio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
    //Variable declaration
    int L;      //No of inputs
    int M = 3;  //No of hidden neurons
    int N;      //No of outputs
    int P;      //No of training patterns
    int T;      //No of testing patterns


    int i,j,k,p;
    int iteration = 1;

    float TMSE = 100;
    float aTMSE;
    float LR = 0.5; //Learning rate

    FILE *input;
    FILE *toutput;
    FILE *ainput;
    FILE *atoutput;
    FILE *output1;
    FILE *output2;
    FILE *output3;

    //Taking inputs from the user
    printf("Enter the Number of inputs\n");
    scanf("%d",&L);
    printf("Enter the Number of outputs\n");
    scanf("%d",&N);
    printf("Enter the Number of training patterns\n");
    scanf("%d",&P);
    printf("Enter the Number of testing patterns\n");
    scanf("%d",&T);

    //Reading the input file
    float I[P+1][L+1];
    float Itemp[P+1][L+1];

    input = fopen("input.txt","r");

    for(p=1;p<=P;p++)
    {
        for(i=1;i<=L;i++)
        {
            fscanf(input,"%f",&I[p][i]);
        }
    }

    fclose(input);

   //Normalizing the input
    float max, min;

    for(p=1;p<=P;p++)
    {
        for(i=1;i<=L;i++)
        {
            Itemp[p][i] = I[p][i];
        }
    }
```

```
    printf("\n");

    for(i=1;i<=L;i++)
    {
        max = I[1][i];
        min = I[1][i];

        for(p=1;p<=P;p++)
        {
            if(max<=I[p][i])
            {
                max = I[p][i];
            }

            if(min>=I[p][i])
            {
                min = I[p][i];
            }
        }

        for(p=1;p<=P;p++)
        {
            I[p][i] = (((Itemp[p][i]-min)*0.8)/(max-min)) + 0.1;
        }
    }

    //Reading Target output
        float TO[P+1][N+1];
        float TOtemp[P+1][N+1];

        toutput = fopen("toutput.txt","r");

        for(p=1;p<=P;p++)
        {
            for(k=1;k<=N;k++)
            {
                fscanf(toutput,"%f",&TO[p][k]);
            }
        }

        fclose(toutput);

    //Normalizing target output

        printf("\n");

        for(p=1;p<=P;p++)
        {
            for(k=1;k<=N;k++)
            {
                TOtemp[p][k] = TO[p][k];
            }
        }

        for(k=1;k<=N;k++)
        {
            max = TO[1][k];
            min = TO[1][k];

            for(p=1;p<=P;p++)
            {
                if(max<=TO[p][k])
                {
                    max = TO[p][k];
                }
```

```
                if(min>=TO[p][k])
                {
                    min = TO[p][k];
                }
            }

            for(p=1;p<=P;p++)
            {
                TO[p][k] = (((TOtemp[p][k]-min)*0.8)/(max-min)) + 0.1;
            }
        }

    //Bias1
    for(p=1;p<=P;p++)
    {
        I[p][0] = 1.0;
    }

    //Weights initialization
    float V[L+1][M+1];
    float W[M+1][N+1];

    for(i=0;i<=L;i++)
    {
        for(j=1;j<=M;j++)
        {
            V[i][j] = (float)(rand()%10)/(float)10;
        }
    }

    for(j=0;j<=M;j++)
    {
        for(k=1;k<=N;k++)
        {
            W[j][k] = (float)(rand()%10)/(float)10;
        }
    }

    output1 = fopen("output1.txt","w");
    fprintf(output1,"iteration\tTMSE\n");

    output3 = fopen("output3.txt","w");

    while(TMSE>0.001 && iteration<=100000)
    {
        //Input to the hidden layer
        float IH[P+1][M+1];

        for(p=1;p<=P;p++)
        {
            for(j=1;j<=M;j++)
            {
                IH[p][j] = 0;
                for(i=0;i<=L;i++)
                {
                    IH[p][j] = IH[p][j] + (I[p][i] * V[i][j]);
                }
            }
        }

        //Output of the hidden layer (TF Log-sigmoid)
        float OH[P+1][M+1];

        for(p=1;p<=P;p++)
        {
            for(j=1;j<=M;j++)
```

```
            {
                OH[p][j] = 1.0/(1.0+exp(-1.0*I[p][j]));
            }
        }

        //Bias2
        for(p=1;p<=P;p++)
        {
            OH[p][0] = 1.0;
        }

        //Input to output layer
        float IO[P+1][N+1];

        for(p=1;p<=P;p++)
        {
            for(k=1;k<=N;k++)
            {
                IO[p][k] = 0;
                for(j=0;j<=M;j++)
                {
                    IO[p][k] = IO[p][k] + (OH[p][j] * W[j][k]);
                }
            }
        }

        //Output of the output layer (TF Tan-sigmoid)
        float OO[P+1][N+1];

        for(p=1;p<=P;p++)
        {
            for(k=1;k<=N;k++)
            {
                OO[p][k] = 1.0/(1.0+exp(-1.0*IO[p][k]));
            }
        }

        //MSE calculation
        float MSE[P+1][N+1];
        TMSE = 0;
        for(p=1;p<=P;p++)
        {
            for(k=1;k<=N;k++)
            {
                MSE[p][k] = (0.5 * (TO[p][k]-OO[p][k]) * (TO[p][k]-OO[p][k]));

                TMSE = TMSE + MSE[p][k];

            }
        }
        TMSE = TMSE/P;

        fprintf(output1,"%d\t\t%f\n",iteration,TMSE);
        printf("%d\t\t%f\n",iteration,TMSE);

        //Updating weights
        float DV[L+1][M+1];
        float DW[M+1][N+1];

        for(j=0;j<=M;j++)
        {
            for(k=1;k<=N;k++)
            {
                DW[j][k] = 0.0;
                for(p=1;p<=P;p++)
                {
```

```c
                    DW[j][k] = DW[j][k] + ((TO[p][k]-OO[p][k])*OO[p][k]*(1-OO[p][k])*OH[p][j]);
                }
                DW[j][k] = (LR*DW[j][k])/((float)P);
            }
        }

        for(i=0;i<=L;i++)
        {
            for(j=1;j<=M;j++)
            {
                DV[i][j] = 0.0;
                for(p=1;p<=P;p++)
                {
                    for(k=1;k<=N;k++)
                    {
                        DV[i][j] = DV[i][j] + ((TO[p][k]-OO[p][k])*OO[p][k]*(1-OO[p][k])*W[j][k]*I[p][i]*OH[p][j]*(1-OH[p][j]));
                    }
                }
                DV[i][j] = (LR*DV[i][j])/((float)(P*N));
            }
        }

        for(j=0;j<=M;j++)
        {
            for(k=1;k<=N;k++)
            {
                W[j][k] = W[j][k] + DW[j][k];
            }
        }

        for(i=0;i<=L;i++)
        {
            for(j=1;j<=M;j++)
            {
                V[i][j] = V[i][j] + DV[i][j];
            }
        }

        iteration = iteration + 1;
    }

    fprintf(output3,"\nFor the training number of iterations required = %d\nand the average mean square error is  = %f\n",iteration-1,TMSE);

    fclose(output1);

    fprintf(output3,"\n\nV values:\n");

    for(i=0;i<=L;i++)
    {
        for(j=1;j<=M;j++)
        {
            fprintf(output3,"%f\t\t",V[i][j]);
        }
        fprintf(output3,"\n");
    }

    fprintf(output3,"\nW values:\n");

    for(j=0;j<=M;j++)
    {
        for(k=1;k<=N;k++)
        {
            fprintf(output3,"%f\t\t",W[j][k]);
        }
```

```
        fprintf(output3,"\n");
    }


//Testing:=========================================================================================================

    //Reading the testing input file
    float aI[T+1][L+1];
    float aItemp[T+1][L+1];

    ainput = fopen("ainput.txt","r");

    for(p=1;p<=T;p++)
    {
        for(i=1;i<=L;i++)
        {
            fscanf(ainput,"%f",&aI[p][i]);
        }
    }

    fclose(ainput);

    //Normalizing the testing input

    for(p=1;p<=T;p++)
    {
        for(i=1;i<=L;i++)
        {
            aItemp[p][i] = aI[p][i];
        }
    }

    for(i=1;i<=L;i++)
    {
        max = aI[1][i];
        min = aI[1][i];

        for(p=1;p<=T;p++)
        {
            if(max<=aI[p][i])
            {
                max = aI[p][i];
            }

            if(min>=aI[p][i])
            {
                min = aI[p][i];
            }
        }

        for(p=1;p<=T;p++)
        {
            aI[p][i] = (((aItemp[p][i]-min)*0.8)/(max-min)) + 0.1;
        }
    }

        for(p=1;p<=T;p++)
        {
            for(k=1;k<=N;k++)
            {
                printf("\n%f\n",aI[p][k]);
            }
        }


```

```
    //Reading Target output
        float aTO[T+1][N+1];
        float aTOtemp[T+1][N+1];

        atoutput = fopen("atoutput.txt","r");

        for(p=1;p<=T;p++)
        {
            for(k=1;k<=N;k++)
            {
                fscanf(atoutput,"%f",&aTO[p][k]);
            }
        }

        fclose(atoutput);

    //Normalizing target output

        for(p=1;p<=T;p++)
        {
            for(k=1;k<=N;k++)
            {
                aTOtemp[p][k] = aTO[p][k];
            }
        }

        for(k=1;k<=N;k++)
        {
            max = aTO[1][k];
            min = aTO[1][k];

            for(p=1;p<=T;p++)
            {
                if(max<=aTO[p][k])
                {
                    max = aTO[p][k];
                }
                if(min>=aTO[p][k])
                {
                    min = aTO[p][k];
                }
            }

            for(p=1;p<=T;p++)
            {
                aTO[p][k] = (((aTOtemp[p][k]-min)*0.8)/(max-min)) + 0.1;
            }
        }

        for(p=1;p<=T;p++)
        {
            for(k=1;k<=N;k++)
            {
                printf("\n%f\n",aTO[p][k]);
            }
        }


    //Input to the hidden layer
    float aIH[T+1][M+1];

    for(p=1;p<=T;p++)
    {
        for(j=1;j<=M;j++)
        {
            aIH[p][j] = 0.0;
```

```
                for(i=0;i<=L;i++)
                {
                    aIH[p][j] = aIH[p][j] + (aI[p][i] * V[i][j]);
                }
            }
        }

        //Output of the hidden layer (TF Log-sigmoid)
        float aOH[T+1][M+1];

        for(p=1;p<=T;p++)
        {
            for(j=1;j<=M;j++)
            {
                aOH[p][j] = 1.0/(1.0+exp(-1.0*aI[p][j]));
            }

            if(p==1)
                {
                    printf("\n%f\n",aOH[p][j]);
                }
        }

        //Bias2

        for(p=1;p<=T;p++)
        {
            aOH[p][0] = 1.0;
        }

        //Input to output layer
        float aIO[T+1][N+1];

        for(p=1;p<=T;p++)
        {
            for(k=1;k<=N;k++)
            {
                aIO[p][k] = 0.0;
                for(j=0;j<=M;j++)
                {
                    aIO[p][k] = aIO[p][k] + (aOH[p][j]*W[j][k]);
                }
                if(p==1)
                {
                    printf("\n%f\n",aIO[p][k]);
                }
            }
        }

        //Output of the output layer (TF Tan-sigmoid)
        float aOO[T+1][N+1];

        for(p=1;p<=T;p++)
        {
            for(k=1;k<=N;k++)
            {
                aOO[p][k] = 1.0/(1.0+exp(-1.0*aIO[p][k]));

                if(p==1)
                {
                    printf("\n%f\n",aOO[p][k]);
                }
            }
        }

        output2 = fopen("output2.txt","w");
```

```

    fprintf(output2,"i = iteration\nTO = target output\nOO = obtained output\n\n");
    fprintf(output2,"\tTO\t\t  OO\n");

    for(p=1;p<=T;p++)
    {
        for(k=1;k<=N;k++)
        {
            fprintf(output2,"\t%f\t%f",aTO[p][k],aOO[p][k]);
        }
        fprintf(output2,"\n");
    }

    fclose(output2);

    //MSE calculation
    float aMSE[T+1][N+1];

    for(p=1;p<=T;p++)
    {
        for(k=1;k<=N;k++)
        {
            aMSE[p][k] = (0.5 * (aTO[p][k]-aOO[p][k]) * (aTO[p][k]-aOO[p][k]));

            aTMSE = aTMSE + aMSE[p][k];
        }
    }

    aTMSE = aTMSE/((float)(N*T));

    fprintf(output3,"\n\nThe MSE for 'testing' is %f\n",aTMSE);
    printf("\n\nThe MSE for 'testing' is %f\n",aTMSE);

    fclose(output3);

    return 0;
}
```