

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 (M_2) for the given upstream Mach number (M_1) 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 M_2 must always be less than 1, in oblique shock M_2 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 (M_1) and corner angle (θ) to find the Mach number downstream of the shock (M_2).

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 M_1 , β , and γ , where γ 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.

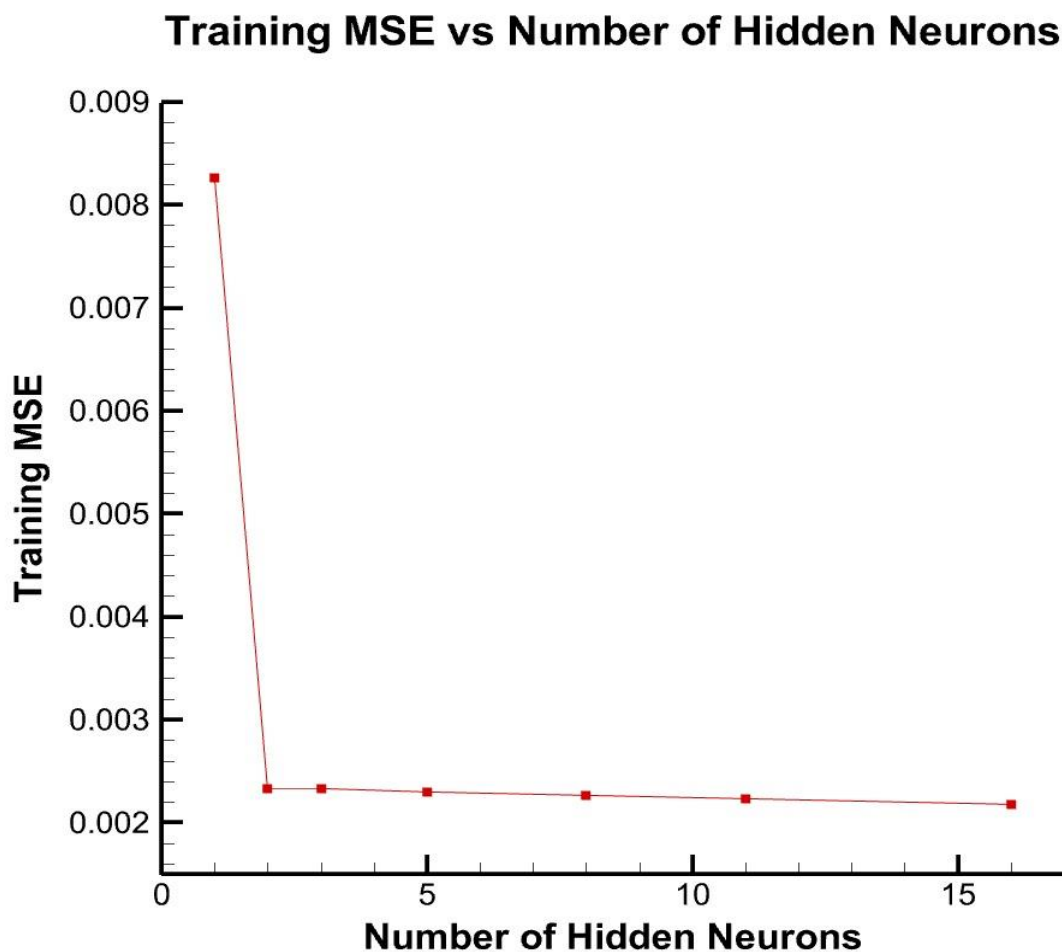
$$f(x) = \frac{1}{1 + e^{-x}}$$

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.

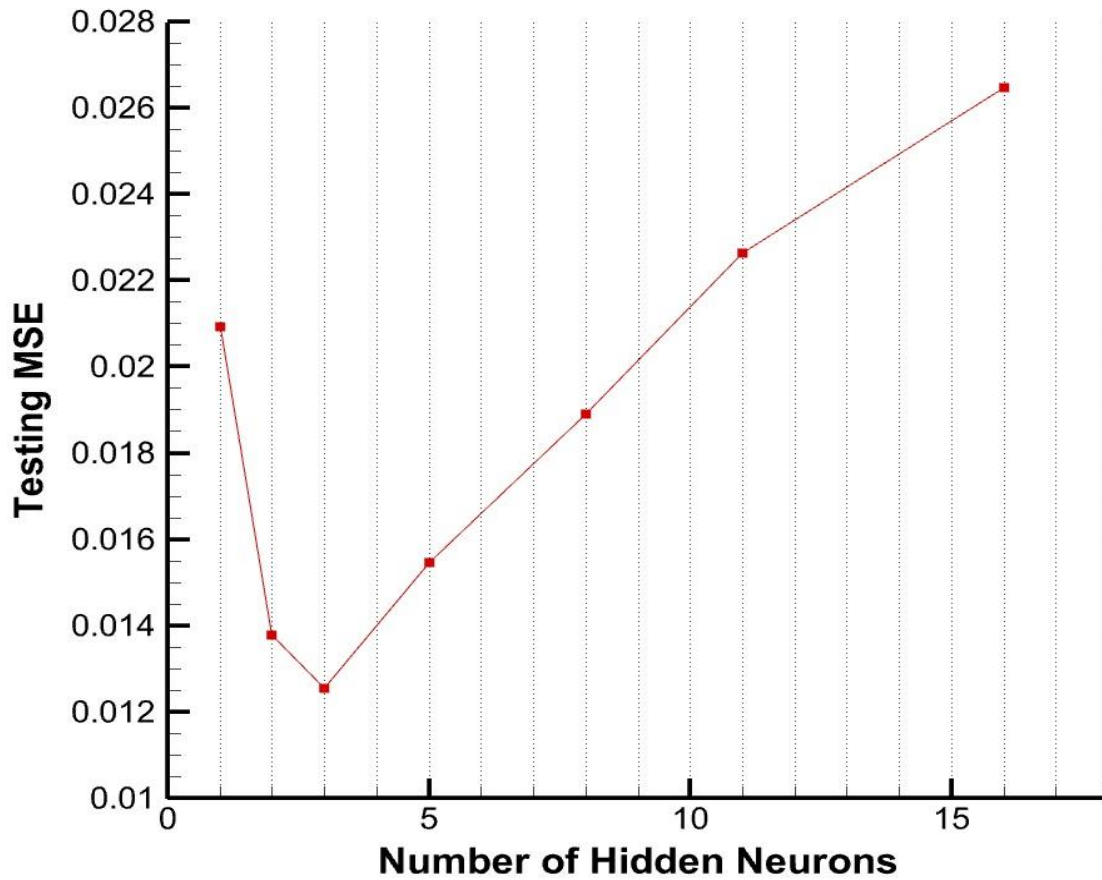
6. The final number of neurons in the hidden layer was taken as '3' based on the conclusion obtained after experimenting with different number of hidden neurons and observing the respective MSE.
7. Similarly the final value of learning rate was chosen to be '0.5'.

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.



Testing MSE vs Number of Hidden Neurons

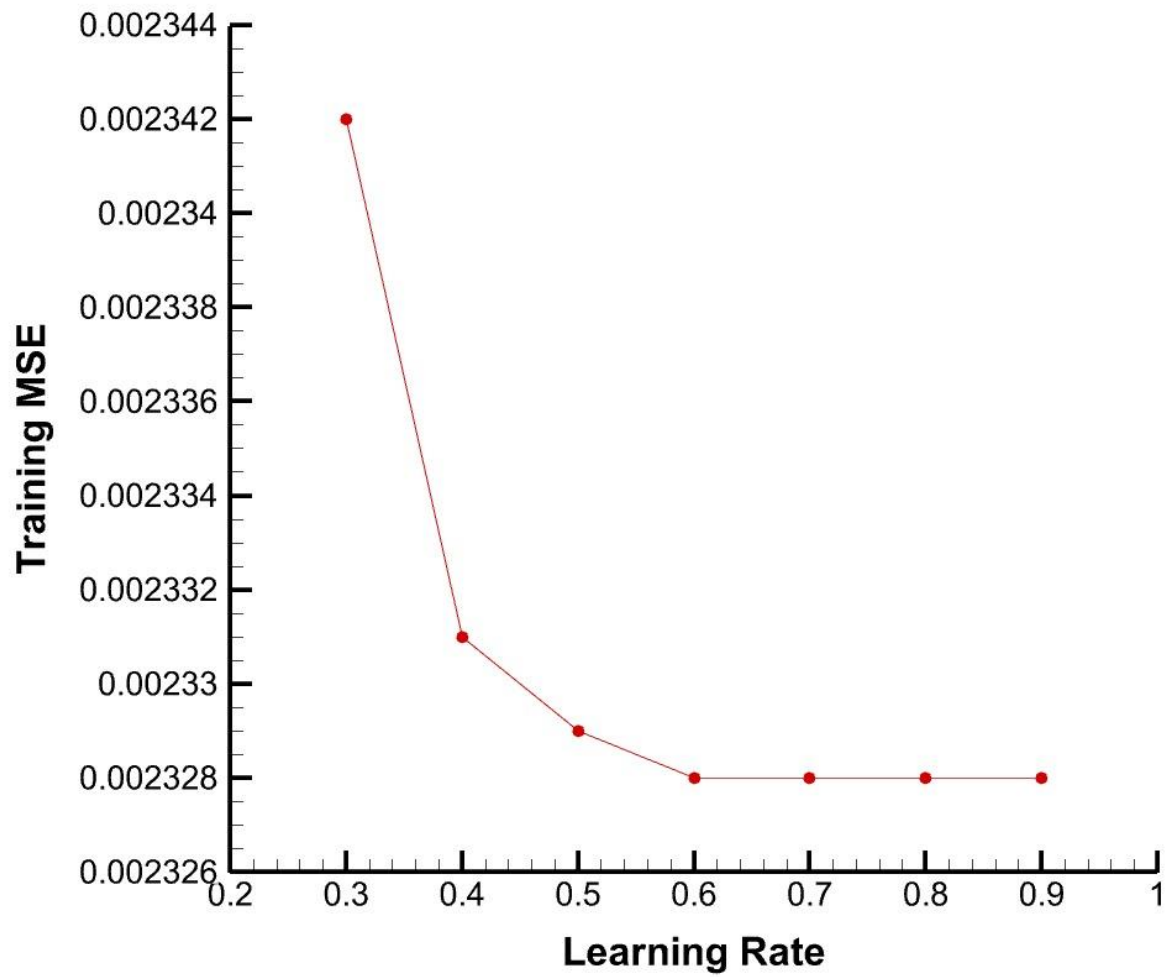


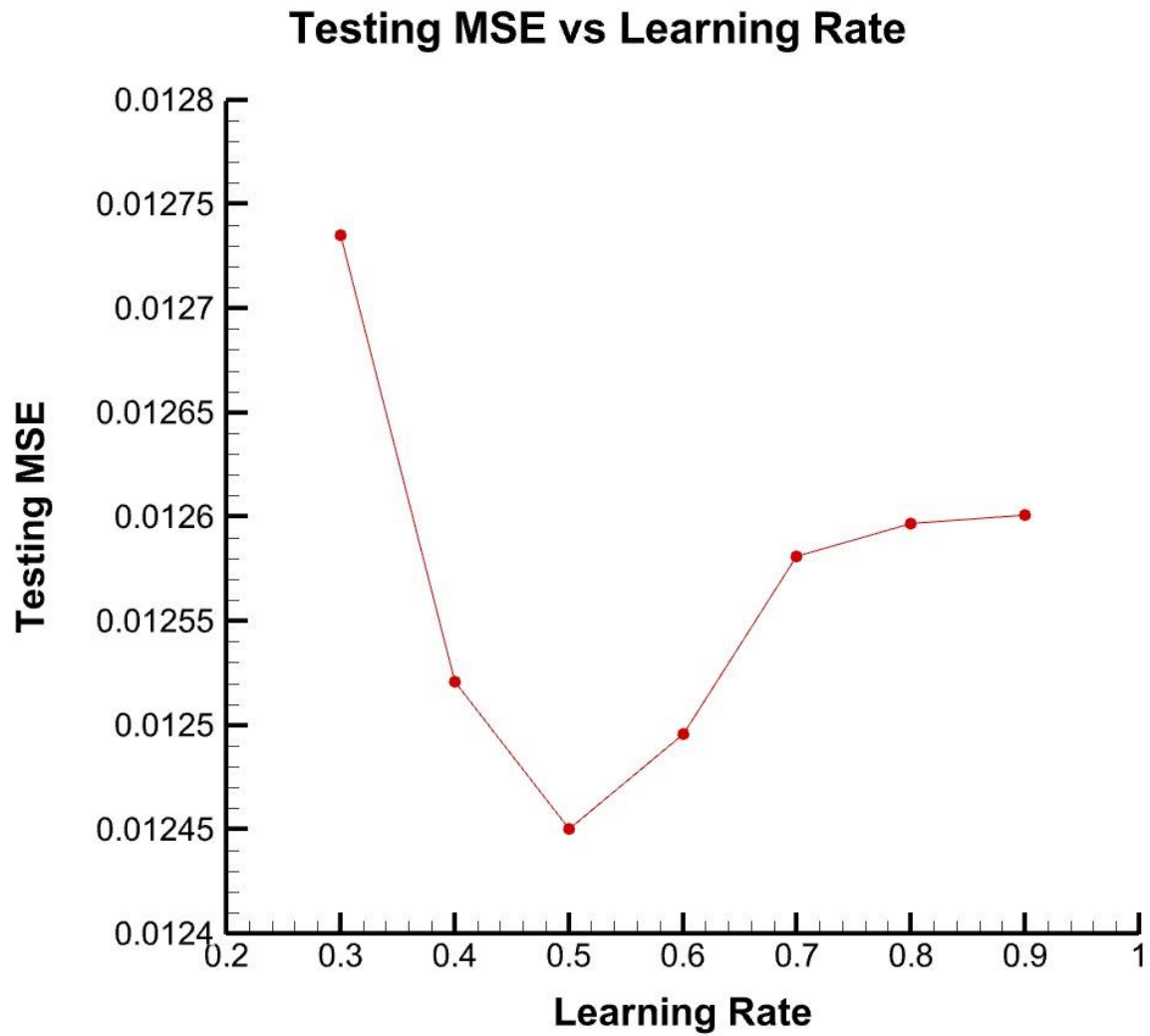
As the number of hidden neurons increases, the final mean square error decreases.

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



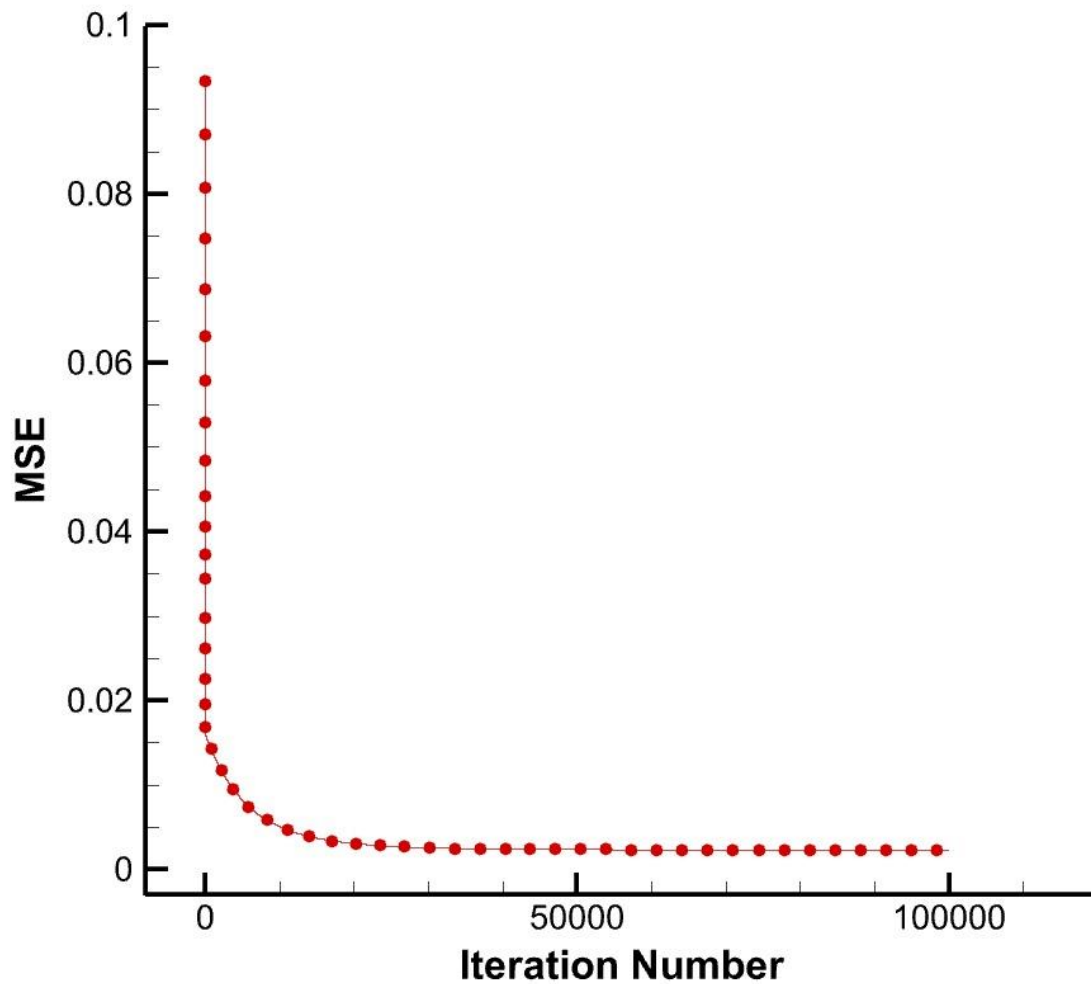


As the value of learning rate (η) increases, the minima is reached faster i.e. fewer iterations are required.

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

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

MSE vs Iteration Number



Conclusion:

A fully connected feed forward back propagating 3 layered ANN model is used to predict Mach number downstream of the shock (M_2) for the given upstream Mach number (M_1) 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.]

```

1  #include <stdio.h>
2  #include<conio.h>
3  #include <stdlib.h>
4  #include <math.h>
5
6  int main()
7  {
8      //Variable declaration
9      int L;        //No of inputs
10     int M = 3;    //No of hidden neurons
11     int N;        //No of outputs
12     int P;        //No of training patterns
13     int T;        //No of testing patterns
14
15
16     int i,j,k,p;
17     int iteration = 1;
18
19     float TMSE = 100;
20     float aTMSE;
21     float LR = 0.5; //Learning rate
22
23     FILE *input;
24     FILE *toutput;
25     FILE *ainput;
26     FILE *atoutput;
27     FILE *output1;
28     FILE *output2;
29     FILE *output3;
30
31     //Taking inputs from the user
32     printf("Enter the Number of inputs\n");
33     scanf("%d",&L);
34     printf("Enter the Number of outputs\n");
35     scanf("%d",&N);
36     printf("Enter the Number of training patterns\n");
37     scanf("%d",&P);
38     printf("Enter the Number of testing patterns\n");
39     scanf("%d",&T);
40
41     //Reading the input file
42     float I[P+1][L+1];
43     float Itemp[P+1][L+1];
44
45     input = fopen("input.txt","r");
46
47     for(p=1;p<=P;p++)
48     {
49         for(i=1;i<=L;i++)
50         {
51             fscanf(input,"%f",&I[p][i]);
52         }
53     }
54
55     fclose(input);
56
57     //Normalizing the input
58     float max, min;
59
60     for(p=1;p<=P;p++)
61     {
62         for(i=1;i<=L;i++)
63         {
64             Itemp[p][i] = I[p][i];
65         }
66     }

```



```

67
68     printf("\n");
69
70     for(i=1;i<=L;i++)
71     {
72         max = I[1][i];
73         min = I[1][i];
74
75         for(p=1;p<=P;p++)
76         {
77             if(max<=I[p][i])
78             {
79                 max = I[p][i];
80             }
81
82             if(min>=I[p][i])
83             {
84                 min = I[p][i];
85             }
86         }
87
88         for(p=1;p<=P;p++)
89         {
90             I[p][i] = (((Itemp[p][i]-min)*0.8)/(max-min)) + 0.1;
91         }
92     }
93
94     //Reading Target output
95     float TO[P+1][N+1];
96     float TTemp[P+1][N+1];
97
98     toutput = fopen("toutput.txt","r");
99
100    for(p=1;p<=P;p++)
101    {
102        for(k=1;k<=N;k++)
103        {
104            fscanf(toutput,"%f",&TO[p][k]);
105        }
106    }
107
108    fclose(toutput);
109
110    //Normalizing target output
111
112    printf("\n");
113
114    for(p=1;p<=P;p++)
115    {
116        for(k=1;k<=N;k++)
117        {
118            TTemp[p][k] = TO[p][k];
119        }
120    }
121
122    for(k=1;k<=N;k++)
123    {
124        max = TO[1][k];
125        min = TO[1][k];
126
127        for(p=1;p<=P;p++)
128        {
129            if(max<=TO[p][k])
130            {
131                max = TO[p][k];
132            }

```

```

133         if(min>=TO[p][k])
134         {
135             min = TO[p][k];
136         }
137     }
138
139     for(p=1;p<=P;p++)
140     {
141         TO[p][k] = (((Totemp[p][k]-min)*0.8)/(max-min)) + 0.1;
142     }
143 }
144
145 //Bias1
146 for(p=1;p<=P;p++)
147 {
148     I[p][0] = 1.0;
149 }
150
151 //Weights initialization
152 float V[L+1][M+1];
153 float W[M+1][N+1];
154
155 for(i=0;i<=L;i++)
156 {
157     for(j=1;j<=M;j++)
158     {
159         V[i][j] = (float)(rand()%10)/(float)10;
160     }
161 }
162
163 for(j=0;j<=M;j++)
164 {
165     for(k=1;k<=N;k++)
166     {
167         W[j][k] = (float)(rand()%10)/(float)10;
168     }
169 }
170
171 output1 = fopen("output1.txt","w");
172 fprintf(output1,"iteration\tTMSE\n");
173
174 output3 = fopen("output3.txt","w");
175
176 while(TMSE>0.001 && iteration<=100000)
177 {
178     //Input to the hidden layer
179     float IH[P+1][M+1];
180
181     for(p=1;p<=P;p++)
182     {
183         for(j=1;j<=M;j++)
184         {
185             IH[p][j] = 0;
186             for(i=0;i<=L;i++)
187             {
188                 IH[p][j] = IH[p][j] + (I[p][i] * V[i][j]);
189             }
190         }
191     }
192
193     //Output of the hidden layer (TF Log-sigmoid)
194     float OH[P+1][M+1];
195
196     for(p=1;p<=P;p++)
197     {
198         for(j=1;j<=M;j++)

```

```

199         {
200             OH[p][j] = 1.0/(1.0+exp(-1.0*I[p][j]));
201         }
202     }
203
204     //Bias2
205     for(p=1;p<=P;p++)
206     {
207         OH[p][0] = 1.0;
208     }
209
210     //Input to output layer
211     float IO[P+1][N+1];
212
213     for(p=1;p<=P;p++)
214     {
215         for(k=1;k<=N;k++)
216         {
217             IO[p][k] = 0;
218             for(j=0;j<=M;j++)
219             {
220                 IO[p][k] = IO[p][k] + (OH[p][j] * W[j][k]);
221             }
222         }
223     }
224
225     //Output of the output layer (TF Tan-sigmoid)
226     float OO[P+1][N+1];
227
228     for(p=1;p<=P;p++)
229     {
230         for(k=1;k<=N;k++)
231         {
232             OO[p][k] = 1.0/(1.0+exp(-1.0*IO[p][k]));
233         }
234     }
235
236     //MSE calculation
237     float MSE[P+1][N+1];
238     TMSE = 0;
239     for(p=1;p<=P;p++)
240     {
241         for(k=1;k<=N;k++)
242         {
243             MSE[p][k] = (0.5 * (TO[p][k]-OO[p][k]) * (TO[p][k]-OO[p][k]));
244
245             TMSE = TMSE + MSE[p][k];
246         }
247     }
248
249     TMSE = TMSE/P;
250
251     fprintf(output1,"%d\t\t%f\n",iteration,TMSE);
252     printf("%d\t\t%f\n",iteration,TMSE);
253
254     //Updating weights
255     float DV[L+1][M+1];
256     float DW[M+1][N+1];
257
258     for(j=0;j<=M;j++)
259     {
260         for(k=1;k<=N;k++)
261         {
262             DW[j][k] = 0.0;
263             for(p=1;p<=P;p++)
264             {

```

```

265         DW[j][k] = DW[j][k] + ((TO[p][k]-OO[p][k])*OO[p][k]*(1-OO[p][k])*OH[p][j]);
266     }
267     DW[j][k] = (LR*DW[j][k])/((float)P);
268 }
269 }
270
271 for(i=0;i<=L;i++)
272 {
273     for(j=1;j<=M;j++)
274     {
275         DV[i][j] = 0.0;
276         for(p=1;p<=P;p++)
277         {
278             for(k=1;k<=N;k++)
279             {
280                 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]));
281             }
282         }
283         DV[i][j] = (LR*DV[i][j])/((float)(P*N));
284     }
285 }
286
287 for(j=0;j<=M;j++)
288 {
289     for(k=1;k<=N;k++)
290     {
291         W[j][k] = W[j][k] + DW[j][k];
292     }
293 }
294
295 for(i=0;i<=L;i++)
296 {
297     for(j=1;j<=M;j++)
298     {
299         V[i][j] = V[i][j] + DV[i][j];
300     }
301 }
302
303 iteration = iteration + 1;
304 }
305
306 fprintf(output3,"\nFor the training number of iterations required = %d\nand the average mean square
error is = %f\n",iteration-1,TMSE);
307
308 fclose(output1);
309
310 fprintf(output3,"\n\nV values:\n");
311
312 for(i=0;i<=L;i++)
313 {
314     for(j=1;j<=M;j++)
315     {
316         fprintf(output3,"%f\t\t",V[i][j]);
317     }
318     fprintf(output3,"\n");
319 }
320
321 fprintf(output3,"\nW values:\n");
322
323 for(j=0;j<=M;j++)
324 {
325     for(k=1;k<=N;k++)
326     {
327         fprintf(output3,"%f\t\t",W[j][k]);
328     }

```

```

329         fprintf(output3, "\n");
330     }
331
332
333 //Testing:=====
334
335 //Reading the testing input file
336 float aI[T+1][L+1];
337 float aItemp[T+1][L+1];
338
339 ainput = fopen("ainput.txt", "r");
340
341 for(p=1; p<=T; p++)
342 {
343     for(i=1; i<=L; i++)
344     {
345         fscanf(ainput, "%f", &aI[p][i]);
346     }
347 }
348
349 fclose(ainput);
350
351 //Normalizing the testing input
352
353 for(p=1; p<=T; p++)
354 {
355     for(i=1; i<=L; i++)
356     {
357         aItemp[p][i] = aI[p][i];
358     }
359 }
360
361 for(i=1; i<=L; i++)
362 {
363     max = aI[1][i];
364     min = aI[1][i];
365
366     for(p=1; p<=T; p++)
367     {
368         if(max<=aI[p][i])
369         {
370             max = aI[p][i];
371         }
372
373         if(min>=aI[p][i])
374         {
375             min = aI[p][i];
376         }
377     }
378
379     for(p=1; p<=T; p++)
380     {
381         aI[p][i] = (((aItemp[p][i]-min)*0.8)/(max-min)) + 0.1;
382     }
383 }
384
385 for(p=1; p<=T; p++)
386 {
387     for(k=1; k<=N; k++)
388     {
389         printf("\n%f\n", aI[p][k]);
390     }
391 }
392

```

```

393 //Reading Target output
394 float aTO[T+1][N+1];
395 float aTOtemp[T+1][N+1];
396
397 atoutput = fopen("atoutput.txt","r");
398
399 for(p=1;p<=T;p++)
400 {
401     for(k=1;k<=N;k++)
402     {
403         fscanf(atoutput,"%f",&aTO[p][k]);
404     }
405 }
406
407 fclose(atoutput);
408
409 //Normalizing target output
410
411 for(p=1;p<=T;p++)
412 {
413     for(k=1;k<=N;k++)
414     {
415         aTOtemp[p][k] = aTO[p][k];
416     }
417 }
418
419 for(k=1;k<=N;k++)
420 {
421     max = aTO[1][k];
422     min = aTO[1][k];
423
424     for(p=1;p<=T;p++)
425     {
426         if(max<=aTO[p][k])
427         {
428             max = aTO[p][k];
429         }
430         if(min>=aTO[p][k])
431         {
432             min = aTO[p][k];
433         }
434     }
435
436     for(p=1;p<=T;p++)
437     {
438         aTO[p][k] = (((aTOtemp[p][k]-min)*0.8)/(max-min)) + 0.1;
439     }
440 }
441
442 for(p=1;p<=T;p++)
443 {
444     for(k=1;k<=N;k++)
445     {
446         printf("\n%f\n",aTO[p][k]);
447     }
448 }
449
450
451 //Input to the hidden layer
452 float aIH[T+1][M+1];
453
454 for(p=1;p<=T;p++)
455 {
456     for(j=1;j<=M;j++)
457     {
458         aIH[p][j] = 0.0;

```

```

459         for(i=0;i<=L;i++)
460         {
461             aIH[p][j] = aIH[p][j] + (aI[p][i] * v[i][j]);
462         }
463     }
464 }
465
466 //Output of the hidden layer (TF Log-sigmoid)
467 float aOH[T+1][M+1];
468
469 for(p=1;p<=T;p++)
470 {
471     for(j=1;j<=M;j++)
472     {
473         aOH[p][j] = 1.0/(1.0+exp(-1.0*aI[p][j]));
474     }
475
476     if(p==1)
477     {
478         printf("\n%f\n",aOH[p][j]);
479     }
480 }
481
482 //Bias2
483
484 for(p=1;p<=T;p++)
485 {
486     aOH[p][0] = 1.0;
487 }
488
489 //Input to output layer
490 float aIO[T+1][N+1];
491
492 for(p=1;p<=T;p++)
493 {
494     for(k=1;k<=N;k++)
495     {
496         aIO[p][k] = 0.0;
497         for(j=0;j<=M;j++)
498         {
499             aIO[p][k] = aIO[p][k] + (aOH[p][j]*W[j][k]);
500         }
501         if(p==1)
502         {
503             printf("\n%f\n",aIO[p][k]);
504         }
505     }
506 }
507
508 //Output of the output layer (TF Tan-sigmoid)
509 float aOO[T+1][N+1];
510
511 for(p=1;p<=T;p++)
512 {
513     for(k=1;k<=N;k++)
514     {
515         aOO[p][k] = 1.0/(1.0+exp(-1.0*aIO[p][k]));
516
517         if(p==1)
518         {
519             printf("\n%f\n",aOO[p][k]);
520         }
521     }
522 }
523
524 output2 = fopen("output2.txt", "w");

```

```

525
526 fprintf(output2,"i = iteration\nTO = target output\nOO = obtained output\n\n");
527 fprintf(output2,"\tTO\t\t\t OO\n");
528
529 for(p=1;p<=T;p++)
530 {
531     for(k=1;k<=N;k++)
532     {
533         fprintf(output2,"\t%f\t%f",aTO[p][k],aOO[p][k]);
534     }
535     fprintf(output2,"\n");
536 }
537
538 fclose(output2);
539
540 //MSE calculation
541 float aMSE[T+1][N+1];
542
543 for(p=1;p<=T;p++)
544 {
545     for(k=1;k<=N;k++)
546     {
547         aMSE[p][k] = (0.5 * (aTO[p][k]-aOO[p][k]) * (aTO[p][k]-aOO[p][k]));
548
549         aTMSE = aTMSE + aMSE[p][k];
550     }
551 }
552
553 aTMSE = aTMSE/((float)(N*T));
554
555 fprintf(output3,"\n\nThe MSE for 'testing' is %f\n",aTMSE);
556 printf("\n\nThe MSE for 'testing' is %f\n",aTMSE);
557
558 fclose(output3);
559
560 return 0;
561 }

```