

MULTI-LAYER FEED FORWARD NEURAL NETWORK WITH BACK PROPAGATION TRAINING ALGORITHM

Soft computing Assignment Report

Course instructor: **Prof. SUKHOMAY PAL**

DEPARTMENT OF MECHANICAL ENGINEERING

NAME: **SOURABH SINGH THAKUR**

Roll No: 224103009



Department of Mechanical Engineering,
Indian Institute of Technology Guwahati,
Assam, India-781039

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 number of neurons in the hidden layer is taken as '3' .
7. Similarly the final value of learning rate was chosen to be '0.6' as training mse graph flattens after this point.

Results:

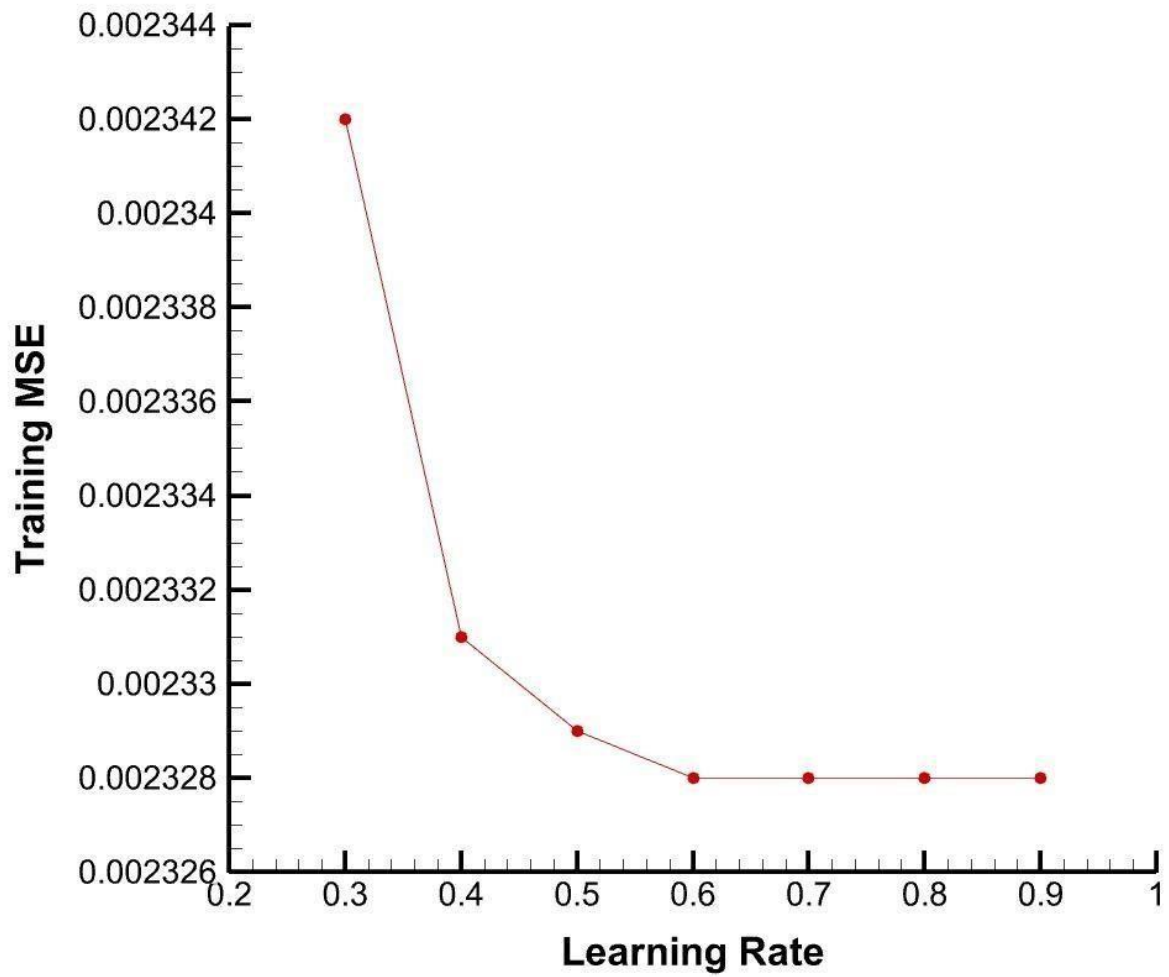
Using the experimental trial and error analysis, the optimum learning rate was found taking the lowest mean square error as the principle criteria.

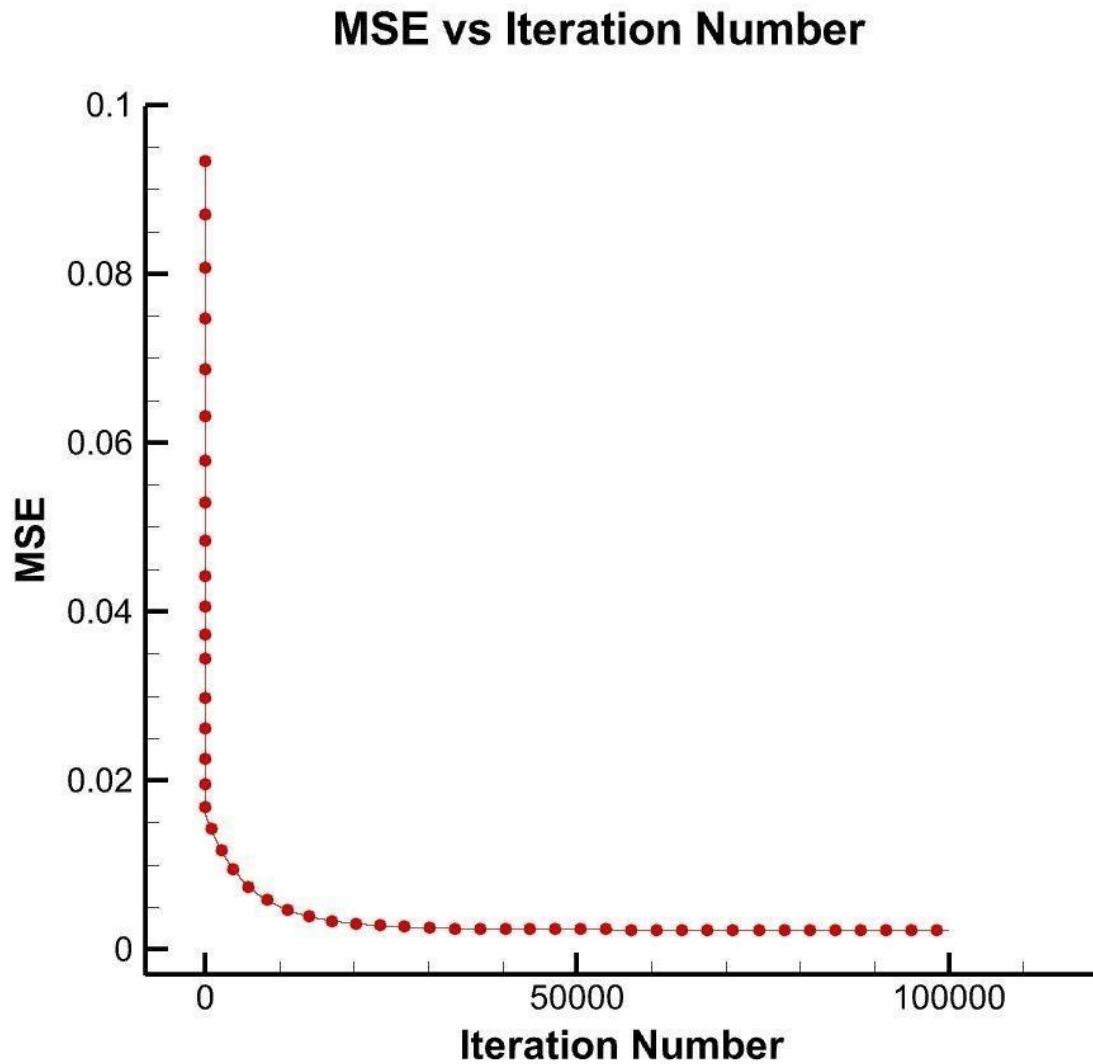
As the value of learning rate (eta) increases, the minima is reached faster i.e. fewer iterations are required.

The mean square error flattens at nearly 0.00239 with negligible change as iterations progress after for eta = 0.6.

Optimum eta was taken as 0.6 as it gives minimum mean square error.

Training MSE vs Learning Rate





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 0.6 .

Code:

```

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     int i,j,k,p;
16     int iteration = 1;
17
18     float TMSE = 100;
19     float aTMSE;
20     float LR = 0.5; //Learning rate
21
22     FILE *input;
23     FILE *toutput;
24     FILE *ainput;
25     FILE *atoutput;
26     FILE *output1;
27     FILE *output2;
28     FILE *output3;
29
30     //Taking inputs from the user
31     printf("Enter the Number of inputs\n");
32     scanf("%d",&L);
33     printf("Enter the Number of outputs\n");
34     scanf("%d",&N);
35     printf("Enter the Number of training patterns\n");
36     scanf("%d",&P);
37     printf("Enter the Number of testing patterns\n");
38     scanf("%d",&T);
39
40     //Reading the input file
41     float I[P+1][L+1];
42     float Itemp[P+1][L+1];
43
44     input = fopen("input.txt","r");
45
46     for(p=1;p<=P;p++)
47     {
48         for(i=1;i<=L;i++)
49         {
50             fscanf(input,"%f",&I[p][i]);
51         }

```

```

52     }
53
54     fclose(input);
55
56     //Normalizing the input
57     float max, min;
58
59     for(p=1;p<=P;p++)
60     {
61         for(i=1;i<=L;i++)
62         {
63             Itemp[p][i] = I[p][i];
64         }
65     }
66
67     printf("\n");
68
69     for(i=1;i<=L;i++)
70     {
71         max = I[1][i];
72         min = I[1][i];
73
74         for(p=1;p<=P;p++)
75         {
76             if(max<=I[p][i])
77             {
78                 max = I[p][i];
79             }
80
81             if(min>=I[p][i])
82             {
83                 min = I[p][i];
84             }
85         }
86
87         for(p=1;p<=P;p++)
88         {
89             I[p][i] = (((Itemp[p][i]-min)*0.8)/(max-min)) + 0.1;
90         }
91     }
92
93     //Reading Target output
94     float TO[P+1][N+1];
95     float TOfemp[P+1][N+1];
96
97     toutput = fopen("toutput.txt","r");
98
99     for(p=1;p<=P;p++)
100    {
101        for(k=1;k<=N;k++)
102        {

```



```

103         fscanf(toutput, "%f", &TO[p][k]);
104     }
105 }
106
107 fclose(toutput);
108
109 //Normalizing target output
110
111 printf("\n");
112
113 for(p=1;p<=P;p++)
114 {
115     for(k=1;k<=N;k++)
116     {
117         T0temp[p][k] = TO[p][k];
118     }
119 }
120
121 for(k=1;k<=N;k++)
122 {
123     max = TO[1][k];
124     min = TO[1][k];
125
126     for(p=1;p<=P;p++)
127     {
128         if(max<=TO[p][k])
129         {
130             max = TO[p][k];
131         }
132         if(min>=TO[p][k])
133         {
134             min = TO[p][k];
135         }
136     }
137
138     for(p=1;p<=P;p++)
139     {
140         TO[p][k] = (((T0temp[p][k]-min)*0.8)/(max-min)) + 0.1;
141     }
142 }
143
144 //Bias1
145 for(p=1;p<=P;p++)
146 {
147     I[p][0] = 1.0;
148 }
149
150 //Weights initialization
151 float V[L+1][M+1];
152 float W[M+1][N+1];
153

```

```

154     for(i=0;i<=L;i++)
155     {
156         for(j=1;j<=M;j++)
157         {
158             V[i][j] = (float)(rand()%10)/(float)10;
159         }
160     }
161
162     for(j=0;j<=M;j++)
163     {
164         for(k=1;k<=N;k++)
165         {
166             W[j][k] = (float)(rand()%10)/(float)10;
167         }
168     }
169
170     output1 = fopen("output1.txt","w");
171     fprintf(output1,"iteration\tTMSE\n");
172
173     output3 = fopen("output3.txt","w");
174
175     while(TMSE>0.001 && iteration<=100000)
176     {
177         //Input to the hidden layer
178         float IH[P+1][M+1];
179
180         for(p=1;p<=P;p++)
181         {
182             for(j=1;j<=M;j++)
183             {
184                 IH[p][j] = 0;
185                 for(i=0;i<=L;i++)
186                 {
187                     IH[p][j] = IH[p][j] + (I[p][i] * V[i][j]);
188                 }
189             }
190         }
191
192         //Output of the hidden layer (TF Log-sigmoid)
193         float OH[P+1][M+1];
194
195         for(p=1;p<=P;p++)
196         {
197             for(j=1;j<=M;j++)
198             {
199                 OH[p][j] = 1.0/(1.0+exp(-1.0*I[p][j]));
200             }
201         }
202
203         //Bias2
204         for(p=1;p<=P;p++)

```

```

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

```

```

256
257     for(j=0;j<=M;j++)
258     {
259         for(k=1;k<=N;k++)
260         {
261             DW[j][k] = 0.0;
262             for(p=1;p<=P;p++)
263             {
264                 DW[j][k] = DW[j][k] + ((TO[p][k]-OO[p][k])*OO[p][k]*(1-
OO[p][k])*OH[p][j]);
265             }
266             DW[j][k] = (LR*DW[j][k])/((float)P);
267         }
268     }
269
270     for(i=0;i<=L;i++)
271     {
272         for(j=1;j<=M;j++)
273         {
274             DV[i][j] = 0.0;
275             for(p=1;p<=P;p++)
276             {
277                 for(k=1;k<=N;k++)
278                 {
279                     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]));
280                 }
281             }
282             DV[i][j] = (LR*DV[i][j])/((float)(P*N));
283         }
284     }
285
286     for(j=0;j<=M;j++)
287     {
288         for(k=1;k<=N;k++)
289         {
290             W[j][k] = W[j][k] + DW[j][k];
291         }
292     }
293
294     for(i=0;i<=L;i++)
295     {
296         for(j=1;j<=M;j++)
297         {
298             V[i][j] = V[i][j] + DV[i][j];
299         }
300     }
301
302     iteration = iteration + 1;
303 }
304

```

```

305     fprintf(output3, "\nFor the training number of iterations required = %d\nand the
average mean square error is  = %f\n", iteration-1, TMSE);
306
307     fclose(output1);
308
309     fprintf(output3, "\n\nV values:\n");
310
311     for(i=0; i<=L; i++)
312     {
313         for(j=1; j<=M; j++)
314         {
315             fprintf(output3, "%f\t\t", V[i][j]);
316         }
317         fprintf(output3, "\n");
318     }
319
320     fprintf(output3, "\nW values:\n");
321
322     for(j=0; j<=M; j++)
323     {
324         for(k=1; k<=N; k++)
325         {
326             fprintf(output3, "%f\t\t", W[j][k]);
327         }
328         fprintf(output3, "\n");
329     }
330
331     //Testing:=====
=====
332
333     //Reading the testing input file
334     float aI[T+1][L+1];
335     float aItemp[T+1][L+1];
336
337     ainput = fopen("ainput.txt", "r");
338
339     for(p=1; p<=T; p++)
340     {
341         for(i=1; i<=L; i++)
342         {
343             fscanf(ainput, "%f", &aI[p][i]);
344         }
345     }
346
347     fclose(ainput);
348
349     //Normalizing the testing input
350
351     for(p=1; p<=T; p++)
352     {
353         for(i=1; i<=L; i++)

```

```

354     {
355         aItemp[p][i] = aI[p][i];
356     }
357 }
358
359 for(i=1;i<=L;i++)
360 {
361     max = aI[1][i];
362     min = aI[1][i];
363
364     for(p=1;p<=T;p++)
365     {
366         if(max<=aI[p][i])
367         {
368             max = aI[p][i];
369         }
370
371         if(min>=aI[p][i])
372         {
373             min = aI[p][i];
374         }
375     }
376
377     for(p=1;p<=T;p++)
378     {
379         aI[p][i] = (((aItemp[p][i]-min)*0.8)/(max-min)) + 0.1;
380     }
381 }
382
383 for(p=1;p<=T;p++)
384 {
385     for(k=1;k<=N;k++)
386     {
387         printf("\n%f\n",aI[p][k]);
388     }
389 }
390
391 //Reading Target output
392 float aTO[T+1][N+1];
393 float aTOtemp[T+1][N+1];
394
395 atoutput = fopen("atoutput.txt","r");
396
397 for(p=1;p<=T;p++)
398 {
399     for(k=1;k<=N;k++)
400     {
401         fscanf(atoutput,"%f",&aTO[p][k]);
402     }
403 }
404

```

```

405     fclose(atoutput);
406
407     //Normalizing target output
408
409     for(p=1;p<=T;p++)
410     {
411         for(k=1;k<=N;k++)
412         {
413             aT0temp[p][k] = aTO[p][k];
414         }
415     }
416
417     for(k=1;k<=N;k++)
418     {
419         max = aTO[1][k];
420         min = aTO[1][k];
421
422         for(p=1;p<=T;p++)
423         {
424             if(max<=aTO[p][k])
425             {
426                 max = aTO[p][k];
427             }
428             if(min>=aTO[p][k])
429             {
430                 min = aTO[p][k];
431             }
432         }
433
434         for(p=1;p<=T;p++)
435         {
436             aTO[p][k] = (((aT0temp[p][k]-min)*0.8)/(max-min)) + 0.1;
437         }
438     }
439
440     for(p=1;p<=T;p++)
441     {
442         for(k=1;k<=N;k++)
443         {
444             printf("\n%f\n",aTO[p][k]);
445         }
446     }
447
448     //Input to the hidden layer
449     float aIH[T+1][M+1];
450
451     for(p=1;p<=T;p++)
452     {
453         for(j=1;j<=M;j++)
454         {
455             aIH[p][j] = 0.0;

```

```

456         for(i=0;i<=L;i++)
457         {
458             aIH[p][j] = aIH[p][j] + (aI[p][i] * V[i][j]);
459         }
460     }
461 }
462
463 //Output of the hidden layer (TF Log-sigmoid)
464 float aOH[T+1][M+1];
465
466 for(p=1;p<=T;p++)
467 {
468     for(j=1;j<=M;j++)
469     {
470         aOH[p][j] = 1.0/(1.0+exp(-1.0*aI[p][j]));
471     }
472
473     if(p==1)
474     {
475         printf("\n%f\n",aOH[p][j]);
476     }
477 }
478
479 //Bias2
480
481 for(p=1;p<=T;p++)
482 {
483     aOH[p][0] = 1.0;
484 }
485
486 //Input to output layer
487 float aIO[T+1][N+1];
488
489 for(p=1;p<=T;p++)
490 {
491     for(k=1;k<=N;k++)
492     {
493         aIO[p][k] = 0.0;
494         for(j=0;j<=M;j++)
495         {
496             aIO[p][k] = aIO[p][k] + (aOH[p][j]*W[j][k]);
497         }
498         if(p==1)
499         {
500             printf("\n%f\n",aIO[p][k]);
501         }
502     }
503 }
504
505 //Output of the output layer (TF Tan-sigmoid)
506 float aOO[T+1][N+1];

```



```

507
508     for(p=1;p<=T;p++)
509     {
510         for(k=1;k<=N;k++)
511         {
512             a00[p][k] = 1.0/(1.0+exp(-1.0*aT0[p][k]));
513
514             if(p==1)
515             {
516                 printf("\n%f\n",a00[p][k]);
517             }
518         }
519     }
520
521     output2 = fopen("output2.txt","w");
522
523     fprintf(output2,"i = iteration\nT0 = target output\nO0 = obtained output\n\n");
524     fprintf(output2,"\tT0\t\t\t O0\n");
525
526     for(p=1;p<=T;p++)
527     {
528         for(k=1;k<=N;k++)
529         {
530             fprintf(output2,"\t%f\t%f",aT0[p][k],a00[p][k]);
531         }
532         fprintf(output2,"\n");
533     }
534
535     fclose(output2);
536
537     //MSE calculation
538     float aMSE[T+1][N+1];
539
540     for(p=1;p<=T;p++)
541     {
542         for(k=1;k<=N;k++)
543         {
544             aMSE[p][k] = (0.5 * (aT0[p][k]-a00[p][k]) * (aT0[p][k]-a00[p][k]));
545
546             aTMSE = aTMSE + aMSE[p][k];
547         }
548     }
549
550     aTMSE = aTMSE/((float)(N*T));
551
552     fprintf(output3,"\n\nThe MSE for 'testing' is %f\n",aTMSE);
553     printf("\n\nThe MSE for 'testing' is %f\n",aTMSE);
554
555     fclose(output3);
556
557     return 0;

```

```
558 }
```

```
559
```

560