INDIAN INSTITUTE OF TECHNOLOGY,GUWAHATI

ME 674 [Soft Computing]

MULTI-LAYER FEED FORWARD NEURAL WITH BACK PROPOGATION

Assignment -1

Submitted by : Aman Shrivastava

Roll No : 224103209

Course Instructor :Sukhomay Pal

INTRODUCTION

Weld quality for shielded metal arc weldin gas compared to air welding have made very necessary to model an artificial neural network (ANN) which is capable of solving difficult and complex problems .

Shielded metal arc welding (SMAW) is a welding process in which coalescence of metals is produced by heat from an electric arc maintained between the tip of a consumable electrode and the surface of the base material in the joint being welded.It mostly used in under water welding.

The weld bead geometry of an underwater welding can be predicted by the neural network control of the input parameters. The water surrounding the weld metal results in a fast cooling of the weld, there by reducing the ductility and tensile strength.

The main goal is to achieve weldbead geometry which will give the weld metal the recommended structural integrity as prescribed by the underwater welding specification.

INPUT PARAMETER :

1.Welding current –I(A)

2.Welding voltage-U(V)

3.Welding speed-v(m/s)

4.Water depth-D(M)

5.Contact tube to work distance-H(m)

OUTPUT PARAMETER :

1.Bead width-W (m)

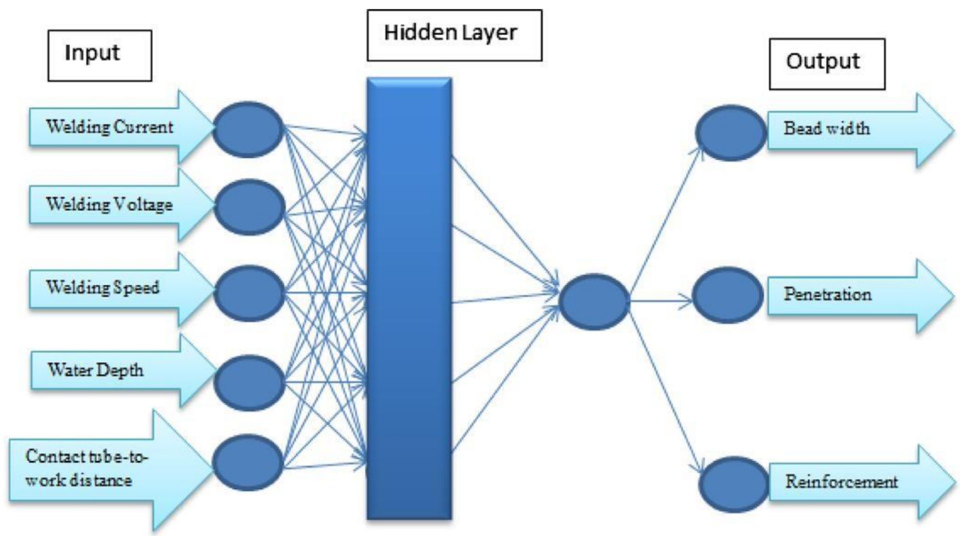

Figure 1

PROCESS TO DEVELOP THE NEURAL NETWORK FOR THIS PROJECT

There are many method to model a ANN.out of I have choosen the sequential multi layer feed forward neural network by back propagation mode of training.

In this we are havingthe five parameters in the input layer so there will be five input in input layer and number of output we must consider is three.so in output layer we have to consider three neuron and in between there is hidden layer. No of hidden layer isconsider and we will se the iteration and accuracy and based on that we will decide the fixed number of hidden layer .there is connection weight between input and hidden layer between hidden and output layer that initially we generate it randomly .For more clear figure is give as below in next page..

	I	U	v	D	H	W
1.	280	28	10	20	40	10.4
2.	320	32	6	20	20	12.5
3.	300	32	10	22	60	10.4
4.	340	28	6	22	0.1	13.9
5.	280	30	6	24	60	12.9
6.	320	26	10	24	0.1	11.6
7.	300	26	6	18	40	12
8.	340	30	10	18	20	9.4
9.	280	26	12	22	20	8.9
10.	320	30	8	22	40	11.8
11.	300	30	12	20	0.1	12.8
12.	340	26	8	20	60	9.5
13.	280	32	8	18	0.1	12.5
14.	320	28	12	18	60	7.9
15.	300	28	8	24	20	10.1
16.	340	32	12	24	40	10

Figure 2

Out of these sixteen experimented data given above 12 data I have used to train the model And four last have used for the testing the ANN model.

METHOLOGY FOR SELECTING THE PARAMETER

Number of Input Neurons: 5(Fixed for this Problem)

Number of Output Neurons: 3(Fixed for this Problem)

Number of Hidden Neurons: 9(Through hit and trial for optimum value)

Learning rate: 0.5

After taking multiple values as a starting point for the learning rate found that it effect the speed of learning rate but also near convergence it causes the MSE to fluctuate. Thus selected the optimum value as 0.6 without sacrificing speed of training.

Transfer Function for Hidden Layer: - log Sigmoidal

Transfer Function for Output Layer: - tan Sigmoidal

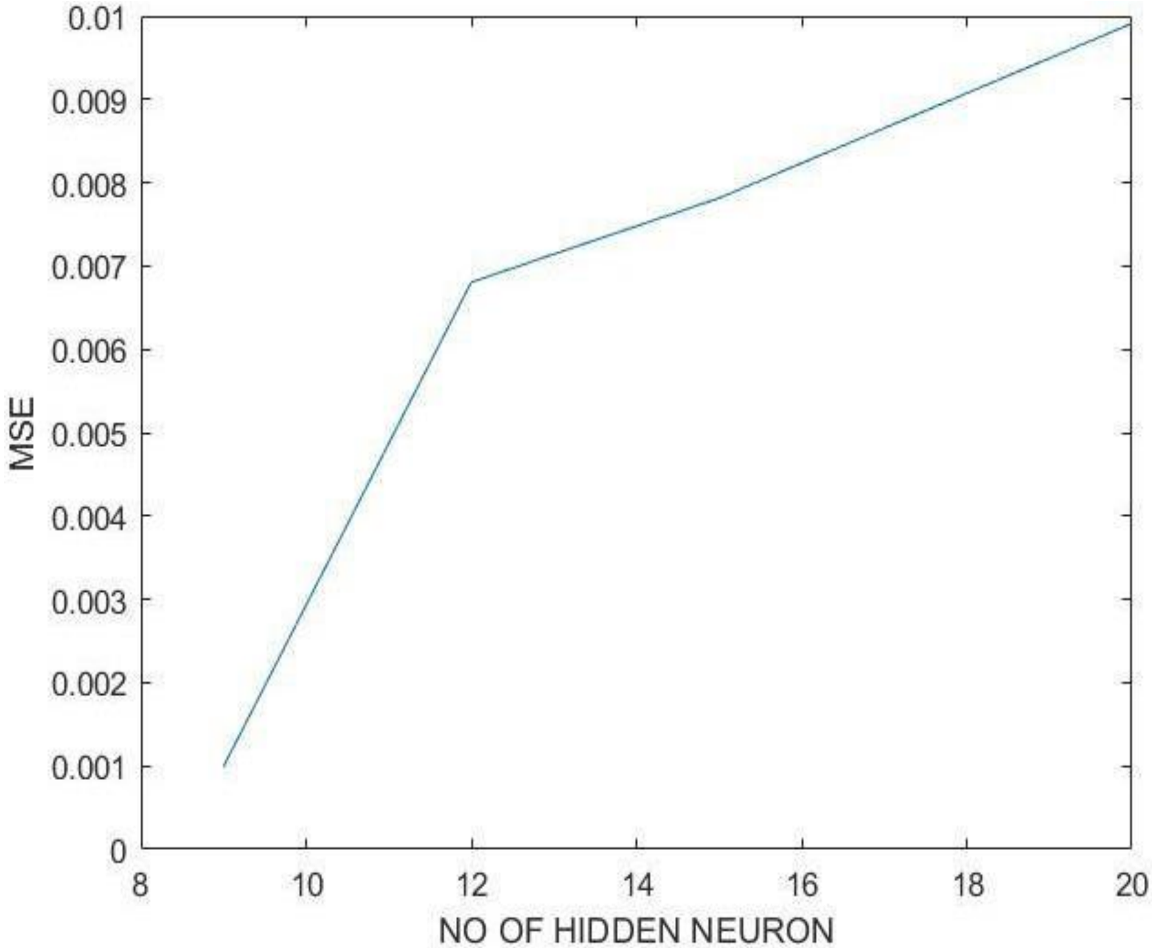
0.01
0.009
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
MSE
8
10
12
14
16
18
20
NO OF HIDDEN NEURON

Figure_3

Conclusion : -

No of Patterns (P) = 16

No of Input Neurons (L) = 5

No of Hidden Neurons (M) = 10

No of Output Neurons (N) = 1

The Reduction in mean square error after certain value increases the number of iteration drastically which require more computing power. Hence to reduce the Prediction error and improvement in prediction values more numbers of training pattern can be considered for training.

Mean Square Error: 0.00099 Number of iteration: 11501

Mean Absolute Error = | Target – predicted| / Number of samples

And using different combination of transfer function for hidden and output layer and learning rate the variation can be seen in mean square error and number of iteration.

CODE

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#define TOL pow(10,-3)

int main()
{

	int P,L,M,N;
	int i,j,k,p;
	int itrn=1;
	srand(time(NULL));

	FILE *OP1,*OP2,*OP3,*IP1,*IP2;

	IP1=fopen("IP_INPUT.txt","r");
	IP2=fopen("IP_TO.txt","r");
	OP1=fopen("Iteration_OUTPUT.txt","w");
	OP2=fopen("RESULT.txt","w");
	OP3=fopen("MSE_vs_Iterations.dat","w");

	double
I[100][100],IH[100][100],OH[100][100],IO[100][100],OO[100][100],TO[100][100],V[100][100],W[100][100];
	double Imax[100],Imin[100],TOmax[100],TOmin[100];
	double del_W[100][100],del_V[100][100],err=0,MSE=0,eta=0.5;

	fscanf(IP1,"%d%d%d%d",&P,&L,&M,&N);
	fprintf(OP2,"No of Patterns (P) = %d\nNo of Input Neurons (L) = %d\nNo of Hidden Neurons (M) =
%d\nNo of Output Neurons (N) = %d\n",P,L,M,N);

//Scaning and Printing Input of Input layer from IP_INPUT.txt

	for(p=1;p<=P;p++)
	{
		for(i=1;i<=L;i++)
		{
			fscanf(IP1,"%lf",&I[i][p]);
		}
	}

	fprintf(OP2,"\nI[L][P] matrix of order (%dX%d)\n",L,P);

	for(i=1;i<=L;i++)
	{
		for(p=1;p<=P;p++)
		{
			fprintf(OP2,"%lf\t",I[i][p]);
		}
```

```
		fprintf(OP2,"\n");
	}

//Normalizing and Printing Input of Input layer

	for(i=1;i<=L;i++)
	{
		Imax[i]=-5000;Imin[i]=5000;

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

	for(p=1;p<=P;p++)
	{
		for(i=1;i<=L;i++)
		{
			I[i][p]=0.1+0.8*((I[i][p]-Imin[i])/(Imax[i]-Imin[i]));
		}
	}

	fprintf(OP2,"\nNormalized I[L][P] matrix of order (%dX%d)\n",L,P);

	for(i=1;i<=L;i++)
	{
		for(p=1;p<=P;p++)
		{
			fprintf(OP2,"%lf\t",I[i][p]);
		}
		fprintf(OP2,"\n");
	}

//Scaning and Printing Target output of Output layer from IP_TO.txt

	fprintf(OP2,"\nTO[P][N] matrix of order (%dX%d)\n",P,N);

	for(p=1;p<=P;p++)
	{
		for(k=1;k<N+1;k++)
		{
			fscanf(IP2,"%lf",&TO[k][p]);
			fprintf(OP2,"%lf\t",TO[k][p]);
```

```
		}
		fprintf(OP2,"\n");
	}
```

//Normalizing and Printing Target output of Output layer

```
	for(k=1;k<N+1;k++)
	{
		TOmax[k]=-5000;TOmin[k]=5000;

		for(p=1;p<=P;p++)
		{
			if(TO[k][p]>TOmax[k])
			{
				TOmax[k]=TO[k][p];
			}
			if(TO[k][p]<TOmin[k])
			{
				TOmin[k]=TO[k][p];
			}
		}
	}

	for(p=1;p<=P;p++)
	{
		for(k=1;k<N+1;k++)
		{
			TO[k][p]=0.1+(0.8*((TO[k][p]-TOmin[k])/(TOmax[k]-TOmin[k])));
		}
	}

	fprintf(OP2,"\nNormalized TO[P][N] matrix of order (%dX%d)\n",P,N);

	for(p=1;p<=P;p++)
	{
		for(k=1;k<N+1;k++)
		{
			fprintf(OP2,"%lf\t",TO[k][p]);
		}
		fprintf(OP2,"\n");
	}
```

//Randomly Generating and Printing Inital Guess for V

```
	fprintf(OP2,"\nV[L+1][M] matrix of order (%dX%d)\n",L+1,M);

	for(i=0;i<L+1;i++)
	{
		for(j=1;j<=M;j++)
		{
			if(i==0)
```

```
				{
					V[i][j]=0;
				}
				else
				V[i][j]=1.0*rand()/RAND_MAX;
			}
		}

		for(i=0;i<=L;i++)
		{
			for(j=1;j<=M;j++)
			{
				fprintf(OP2,"%lf\t",V[i][j]);
			}
			fprintf(OP2,"\n");
		}
		fprintf(OP2,"\n");

	//Randomly Generating and Printing Inital Guess for W

		fprintf(OP2,"\nW[M+1][N] matrix of order (%dX%d)\n",M+1,N);

		for(j=0;j<M+1;j++)
		{
			for(k=1;k<=N;k++)
			{
				if(i==0)
				{
					W[j][k]=0;
				}
				else
				W[j][k]=1.0*rand()/RAND_MAX;
			}
		}

		for(j=0;j<M+1;j++)
		{
			for(k=1;k<=N;k++)
			{
				fprintf(OP2,"%lf\t",W[j][k]);
			}
			fprintf(OP2,"\n");
		}

	//Training of Model using Do-While Loop

		do
		{
			//Calculation of forward Pass

			for(p=1;p<=P-4;p++)
```

```
{
	IH[j][p]=0;
	for(j=1;j<M+1;j++)
	{
		for(i=1;i<L+1;i++)
		{
			IH[j][p]=IH[j][p]+(I[i][p]*V[i][j]);
		}
		IH[j][p]=IH[j][p]+(1.0);
		OH[j][p]=1/(1+exp(-IH[j][p]));
		IH[j][p]=0;
	}
}

//Output of Output Layer

for(p=1;p<=P-4;p++)
{
	IO[k][p]=0;
	for(k=1;k<N+1;k++)
	{
		for(j=1;j<M+1;j++)
		{
			IO[k][p]=IO[k][p]+OH[j][p]*W[j][k];
		}
		IO[k][p]=IO[k][p]+1.0;
		OO[k][p]=(exp(IO[k][p])-exp(-1*IO[k][p]))/(exp(IO[k][p])+exp(-1*IO[k][p]));
		IO[k][p]=0;
	}
}

//Calculation for del_Wjk

fprintf(OP1,"\ndel_Wjk matrix of order (%dX%d)\n",M+1,N);

for(j=0;j<=M;j++)
{
	for(k=1;k<=N;k++)
	{
		del_W[j][k]=0;
		for(p=1;p<=P-4;p++)
		{
			del_W[j][k]=del_W[j][k]+((eta/P)*(TO[k][p]-OO[k][p])*(1-
(OO[k][p]*OO[k][p]))*OH[j][p]);
		}
		fprintf(OP1,"%lf\t",del_W[j][k]);
	}
	fprintf(OP1,"\n");
}

//Calculation for del_Vij
```

```
fprintf(OP1,"\ndel_Vij matrix of order (%dX%d)\n",L+1,M);

for(i=0;i<=L;i++)
{
	for(j=1;j<=M;j++)
	{
		del_V[i][j]=0;
		for(p=1;p<=P-4;p++)
		{
			for(k=1;k<=N;k++)
			{
				del_V[i][j]=del_V[i][j]+((eta/(P*N))*((TO[k][p]-
OO[k][p])*(1-(OO[k][p]*OO[k][p]))*W[j][k]*OH[j][p]*(1-OH[j][p])*I[i][p]));
			}
		}
		fprintf(OP1,"%lf\t",del_V[i][j]);
	}
	fprintf(OP1,"\n");
}

//Calculation of Error

MSE=0;

for(p=1;p<=P-4;p++)
{
	for(k=1;k<=N;k++)
	{
		err=pow((TO[k][p]-OO[k][p]),2)/2;
		MSE=MSE+err;
	}
}

MSE=MSE/P;

fprintf(OP1,"\nMSE=%lf\titrn=%d\n",MSE,itrn);
fprintf(OP3,"%d\t%.10lf\n",itrn,MSE);

//Update Vij

fprintf(OP1,"\nUpdated V Matrix\n");

for(i=0;i<=L;i++)
{
	for(j=1;j<=M;j++)
	{
		V[i][j]=V[i][j]+del_V[i][j];
		fprintf(OP1,"%lf\t",V[i][j]);
	}
	fprintf(OP1,"\n");
```

```
		}
		fprintf(OP1,"\n");

		//Update Wjk

		fprintf(OP1,"\nUpdated W Matrix\n");

		for(j=0;j<=M;j++)
		{
			for(k=1;k<=N;k++)
			{
				W[j][k]=W[j][k]+del_W[j][k];
				fprintf(OP1,"%lf\t",W[j][k]);
			}
			fprintf(OP1,"\n");
		}

		printf("Iteration %d completed\n",itrn);
		itrn++;

	}
	while(MSE>TOL);

//Printing V matrix after Training
	fprintf(OP2,"\nV[L+1][M] matrix of order (%dX%d) after Training\n",L+1,M);

	for(i=0;i<=L;i++)
		{
			for(j=1;j<=M;j++)
			{
				fprintf(OP2,"%lf\t",V[i][j]);
			}
			fprintf(OP2,"\n");
		}
		fprintf(OP2,"\n");

//Printing W matrix after Training
	fprintf(OP2,"\nW[M+1][N] matrix of order (%dX%d) after Training\n",M+1,N);

		for(j=0;j<=M;j++)
		{
			for(k=1;k<=N;k++)
			{
				fprintf(OP2,"%lf\t",W[j][k]);
			}
			fprintf(OP2,"\n");
		}

//Testing of Model

		//Calculation of forward Pass
```

```
		for(p=21;p<=24;p++)
		{
			IH[j][p]=0;
			for(j=1;j<M+1;j++)
			{
				for(i=1;i<L+1;i++)
				{
					IH[j][p]=IH[j][p]+(I[i][p]*V[i][j]);
				}
				IH[j][p]=IH[j][p]+1.0;
				OH[j][p]=1/(1+exp(-IH[j][p]));
//				fprintf(OP2,"\nIH[%d][%d]:%lf\tOH[%d][%d]:%lf",j,p,IH[j][p],j,p,OH[j][p]);
				IH[j][p]=0;
			}

		}
		fprintf(OP2,"\n");

		//Output of Output Layer

		fprintf(OP2,"\nOutput of Model with Error\n");

		for(p=21;p<=24;p++)
		{
			IO[k][p]=0;
			for(k=1;k<N+1;k++)
			{
				for(j=1;j<=M+1;j++)
				{
					IO[k][p]=IO[k][p]+OH[j][p]*W[j][k];
				}
				IO[k][p]=IO[k][p]+1.0;
				OO[k][p]=(exp(IO[k][p])-exp(-1*IO[k][p]))/(exp(IO[k][p])+exp(-1*IO[k][p]));

	fprintf(OP2,"\nIO[%d][%d]:%lf\tOO[%d][%d]:%lf\tTO[%d][%d]:%lf\tError:%lf",k,p,IO[k][p],k,p,OO[k
][p],k,p,TO[k][p],fabs(OO[k][p]-TO[k][p]));
				IO[k][p]=0;
			}
		}

	fclose(IP1);
	fclose(IP2);
	fclose(OP1);
	fclose(OP2);
	fclose(OP3);

	printf("\n\nResults have been stored inn respective Output files.");

	return 0;
}
```