APPENDIX - C(1)

The algorithm of back propagation training method discussed in Section 3.4 is as follows :

Back-Propagation Training Algorithm (BPTA)

Back - propagation training rule contains training set H as follows :

$$H = \left\{ (\underline{x}_p , \underline{d}_p) \; ; \; p = 1,2,\ldots,P \right\}$$

where $\underline{x}_p$ is (n X 1), $\underline{d}_p$ is (R X 1). Note that n-th component of $\underline{x}_p$ is of value -1. Hidden layer of size J-1 having output $\underline{o}$ is selected. Note that J-th component of $\underline{o}$ is of value -1 and $\underline{y}$, final output vector is (R X 1).

Now, major steps involved in BPTA as follows :

Step 1 Select initial weights w and v are initialized at small random values. Here w is (R X J), v is (J X n). Set $\eta > 0$ and initialize E_{max} to small value (where E_{max} is prespecified small error quantity). Counter(k) and error(E) are initialized.

$$k \leftarrow 1, \quad p \leftarrow 1, \quad E \leftarrow 0$$

Step 2 Choose any input vector $\underline{x}_p$ and layer's output is computed

$$\underline{x} \leftarrow \underline{x}_p, \quad \underline{d} \leftarrow \underline{d}_p$$

$$o_j \leftarrow f(\underline{v}_j' \underline{x}), \quad \text{for} \quad j = 1,2,\ldots,J$$

where $\underline{v}_j$, a column vector, is the j-th row of v, and

$$y_r \leftarrow f(\underline{w}_r' \underline{o}) \quad \text{for} \quad r = 1,2,\ldots,R$$

where $\underline{w}_r$, is column vector, is the r-th row of w.

Step 3 Compute error value E as

$$E \leftarrow \frac{1}{2} (d_r - y_r) + E , \qquad \text{for } r = 1,2,\ldots,R$$

Step 4 Output layer weights are adjusted :

$$w_{rj} \leftarrow w_{rj} + \frac{1}{2} \eta (d_r - y_r) (1 - y_r^2) o_j$$

$$\text{for } r = 1,2,\ldots,R$$

$$j = 1,2,\ldots,J$$

Step 5 Hidden layer weights are adjusted :

$$v_{ji} \leftarrow v_{ji} + \frac{1}{4} \eta (1 - o_j^2) \sum_{r=1}^{R} (d_r - y_r)(1 - y_r^2) w_{rj} x_i ,$$

$$\text{for } r = 1,2,\ldots,R$$

$$j = 1,2,\ldots,J$$

$$i = 1,2,\ldots,p.$$

Step 6 If $p < P$ then $p \leftarrow p + 1$, $q \leftarrow q + 1$ and go to step 2; otherwise goto step 7.

Step 7 If $E < E_{max}$, then stop the iterative procedure. Otherwise initialize error E to zero and enter the new training cycle by going to step 2.

Note: Where f(.) is activation function as follows:

$$f(x) = \frac{2}{1 + \exp(-\lambda x)} - 1,$$

with $\lambda=1$ and

$$f'(x) = \frac{1}{2}(1 - f(x)*f(x)).$$

APPENDIX C(2)

Back-Prop : This software package can be used for training multilayer feed forward ANNs and the training procedure used here discussed in Section 3.4.

```
                                /* Back-prop */


                        /* Back_pro.c for Multilayer Feedforward ANN */



#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <dos.h>

void main()

  {

      float z[12][12],d[12][12],w[12][12],v[12][12],z1[12],d1[12];
      float e; int ip,r1,p,i,j,k,I,J,K,P,q; float m[12],n[12];
      float ed[12],ds[12];float ds1,s,ita,emax; int cnt,r,qmax,t1,t,t2;
      double y[12],o[12];
      char c;

                  FILE *fp1,*fp2,*fp3,*fp5;


     clrscr(); gotoxy(3,7);
   printf( "\t This softwre needs three Data files:\n ");
   printf("\n\ti) INPUT.dat : Input data file containing P set of input ");
   printf("\n                      vectors each of dim I. \n");
   printf( "\n\tii) D_OUT.dat : File containing K dim vector of Target ");
   printf("values.\n");
   printf( "\n\tiii) Test.dat : File containing I dim vector of Test val.");

                  fp1= fopen("SR.DAT","r");
                  fp2= fopen("SRD.DAT","r");
                  fp3= fopen("TEST.DAT","r");
                  fp5= fopen("SRN.DAT","w");

                  if(fp1==NULL)
               {
                    printf("\n Error open file-1");
                         exit(0);
                   }
```

```
    if(fp2==NULL)
{
     printf("\nError open file-2");
          exit(0);
    }

     if(fp3==NULL)
{
     printf("\Error open file-3");
          exit(0);
    }

   /* enter the data....*/
        printf( "\n\n Enter values of : ");
        printf("\n I= No. of Input neurons, K= Target val ");
        printf(" P= No of Patterns\n\t");
        scanf("%d",&I);
        printf("\t\t\n");
        scanf("%d",&K);
        printf("\t\t");
        scanf("%d",&P);
        printf("\n The val of I,K,P are =%d %d %d",I,K,P);
        I=I+1;   /* for fixed Input = -1. */

printf("\n Reading the values of z[][] and d[][]\n");

for (p=1;p<=P;p++)

{
   for(i=1;i<=I-1;i++)

     fscanf(fp1,"%f",&z[p][i]);
         z[p][I] =-1;
     }
         for (p=1;p<=P;p++)
           for(k=1;k<=K;k++)
           fscanf(fp2,"%f",&d[p][k]);

         /* Enter no.of neurons in the hidden layer */
              clrscr();
     gotoxy(3,15);
                c= ' ';
do
{
     printf("Enter No.of neurons in Hidden Layer J   =:    ");
     scanf("%d",&J);
     printf("\t\t\tLearning const    = :\t");
     scanf("%f",&ita);
     printf("\t\t\tMax. Error        = :\t");
     scanf("%f",&emax);
     printf("\t\t\tNo. of Iteraticns = :\t");
     scanf("%d",&qmax);
     J= J+1; /* For fixed hidden neuron = -1. */
```

```
 /* generate w[] and v[] */

        randomize();
for(k=1;k<=K;k++)
 for(j=1;j<=J;j++)
       w[k][j]= 2.0*((float)rand()/RAND_MAX)-1.0;

       randomize();
      for(j=1;j<=J;j++)
       for(i=1;i<=I;i++)
            v[j][i]=2.0*((float)rand()/RAND_MAX)-1.0;

        /*Iteration Starts */

        q=0;

while(q<=qmax)
 {
      e=0; q = q+1;

      printf(" \nIter no. = %d ",q);
   /* This portion randomises the order of Inputs */
      randomize();

      r1 = random(P);
       if(r1==0)
            p=1;
       else
            p=r1;

       ip=1;

      while  (ip <=P)
   {
            for(i=1;i<=I;i++)
              {
         z1[i]=z[p][i];
                }

            for(k=1;k<=K;k++)
                 {
            d1[k]=d[p][k];
                 }

  /* compute net*/
       for(j=1;j<= J-1;j++)
      {
            s=0;
       for(i=1;i<= I;i++)
          s=s+v[j][i]*z1[i];
               m[j]=s;
       }
```

```
/* compute f(net)*/
      for (j=1;j<=J-1;j++)

       y[j]= (2/(1.0+exp(-m[j])))-1;

                y[J]=-1;

    for(k=1;k<=K;k++)
  {
           s=0;
     for( j=1;j<=J;j++)
      s=s+w[k][j]*y[j];
            n[k]=s;
      }
    for(k=1;k<=K;k++)
  {
      o[k]=(2/(1.0+exp(-n[k])))-1;

      }

/* error computation*/
      for(k=1;k<=K;k++)
          e=e+(d1[k]-o[k])*(d1[k]-o[k])/2;

/* error correction */
      for(k=1;k<=K;k++)
          ed[k]= (d1[k]-o[k])*(1-o[k]*o[k])/2.0;

/* hidden layer error sig */
      for(j=1;j<=J;j++)
  {
            ds1=0;
       for(k=1;k<=K;k++)
       ds1= ds1 + ed[k]*w[k][j];
       ds[j]= ds1*(1-y[j]*y[j])/2.0;
      }

 /* output layer wts*/
     for (k=1;k<=K;k++)
        for(j=1;j<=J;j++)
        w[k][j]=w[k][j]+ita*ed[k]*y[j];

 /* hidden layer wts */
       for(j=1;j<=J;j++)
            for(i=1;i<=I;i++)
           v[j][i]=v[j][i]+ita*ds[j]*z1[i];

  /* input next z */

      p =p+1;
      if(p > P)
        p= 1;
        ip=ip+1;
}      /* end of for loop for inputting all z. */
```

```
            e=sqrt(e)/(P*K);
           gotoxy(5,60);
      printf("\t\t Error is e = %.5f \n",e);

if(( e<=emax) || ( q > qmax))
 {
              printf("\n Process is terminated..\n\n");
              printf("\n Error is = %.5f\n ",e);

           /* Print the Final wts  */
  printf("\n\t Final Hidden to Output layer wts are :\n\n");
       for(k=1;k<=K;k++)
           for(j=1;j<=J;j++)
           printf("\t w[%d][%d]=%.5f  ",k,j,w[k][j]);
           printf("\n");
           printf("\n\t Final Input to Hidden layer wts are :\n");
                 for(j=1;j<=J;j++)
                   for(i=1;i<=I;i++)
                   printf("\t v[%d][%d]=%.5f \n ",j,i,v[j][i]);

             break;


       }  /* End of for loop */

      }  /* end of while loop(q<qmax) */

           printf("\n To try one more J, Press any key; ");
           printf("otherwise to stop,type 'Y' or 'y' \n");
            c = getche();
            if(c == 'Y'|| c== 'y')
                 break;

   }  while((c != 'Y'));  /*end of outer while loop */


    /* printf(" \n Enter value of t:");
                scanf("%d",&t);
             if(t==0)
                      exit(0);
                else
                 {
      for (p=1;p<=t;p++)

            {
    */
          for(k=1;k<=K;k++)
            o[k]=0;
                  clrscr();
```

```
printf(" \n Reading the test data ..");

     for(i=1;i<=I-1;i++)
            fscanf(fp3,"%f",&z1[i]);
             z1[I]= -1;
                 for(i=1;i<=I;i++)
                    printf("z=%f",z1[i]);

 /*  printf("Enter # of sets of test data...\n");
           scanf("%d", &t);
           printf("Enter the test data...");
             for(k=1;k<=t;k++)
                  for(i=1;i<=I-1;i++)
                  scanf("%f",&z[k][i]);
           for(r=1;r<=t;r++)
       {
             for(i=1;i<I;i++)
             z1[i]=z[r][i];
             z1[I]=-1;       */

             for(j=1;j<=J-1;j++)
            {
                        s=0;
                  for(i=1;i<=I;i++)
                  s=s+v[j][i]*z1[i];
                       m[j]=s;

            }

                for(j=1;j<=J-1;j++)
                  y[j]= (2/(1.0+exp(-m[j])))-1;
                  y[J]= -1;

                   for(k=1;k<=K;k++)
                  {
                            s=0;
                       for(j=1;j<=J;j++)
                       s=s+w[k][j]*y[j];
                              n[k]=s;
                  }

          for(k=1;k<=K;k++)
     {
        o[k]=(2/(1.0+exp(-n[k])))-1;
        printf("\n O[%d]= %.3f  ",k,o[k]);

       fprintf(fp5,"\n the value of o[%d]= %f ", k,o[k]);
     }

                fclose(fp5);
             printf("\n program end");
}
```