Rapport : TP Softcomputing:

Master IASE : semestre 1.

Réaliser par :

Edgdougui Ayoub.

Lasri Kawtar.

Lasri Wafae.

UMV 2015/1016

Table du contenu :

Liste des figures :

Introduction :

Les réseaux de neurones ont une histoire relativement jeune (environ 50 ans) et les applications intéressantes des réseaux de neurones n'ont vu le jour qu'il à une vingtaine d'année (développement de l'informatique).

Les réseaux de neurones sont composés d'éléments simples (ou neurones) fonctionnant en parallèle. Ces éléments ont été fortement inspirés par le système nerveux biologique. Comme dans la nature, le fonctionnement du réseau (de neurone) est fortement influencé par la connections des éléments entre eux. On peut entraîner un réseau de neurone pour une tâche spécifique (reconnaissance de caractères ou des nombres par exemple) en ajustant les valeurs des connections (ou poids) entre les éléments (neurone).

En général, l'apprentissage des réseaux de neurones est effectué de sorte que pour une entrée particulière présentée au réseau corresponde une cible spécifique. L'ajustement des poids se fait par comparaison entre la réponse du réseau (ou sortie obtenue) et la cible, jusqu'à ce que la sortie corresponde à la cible (sortie désirée).

On s'intéressera dans ce TP aux algorithmes d'apprentissages suivant :

- Algorithme du perceptron *« Rosemblatt » 1958-1962.*
- Algorithme de l'Adaline (régle de Delta) *« Widrow-Hoff » 1960* .
- Algorithme de la rétro-propagation .

I-Methode d'apprentissage du perceptron « Rosemblatt »

Algorithme d'apprentissage monocouche (perceptron) permet de résoudre des problèmes linéairement séparable .

Algorithme :

1- Initialisation des poids et du seuil (biais) à des valeurs petites .
2- Présentation de la base d'apprentissage : couple (entrées et sortie désirée).
3- Calcul de la somme pondérer:

$$s=\sum w_i x_i$$

4- Calcule de la sortie obtenue en utilisant la fonction d'activation (Heaviside) :

$$O=\text{signe}(s) \begin{cases} s > 0 \Rightarrow o = 1 \\ s \leq 0 \Rightarrow o = 0 \end{cases}$$

5- Comparaison de la sortie obtenue avec la sortie désirée :

S'ils sont différents changement des poids tel que :

$$w_i(t+1) = w_i(t) + \mu((d_i - o_i)x_i)$$

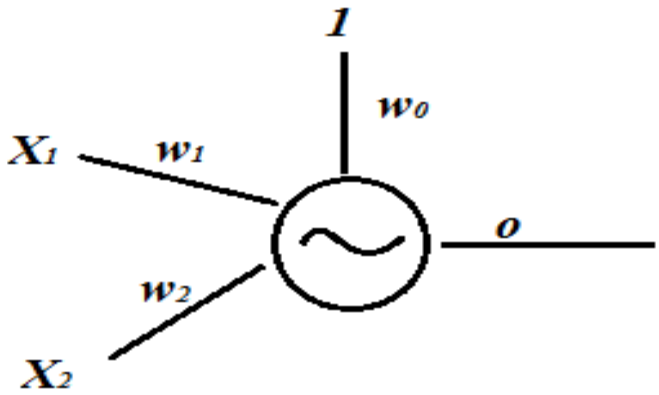
6- Tant qu'on a pas converger et on a pas atteint le nombre d'ittération maximal on retourne a l'étape 3.

Convergence : toutes le obtenues de toutes les entrées sont égaux aux sorties desirées.

Exemple (OR) : Soit le perceptron suivant :

Les poids initiaux : W = (-0.2 ;0 ;1) avec $\mu = 1$ (Pas d'apprentissage)

.



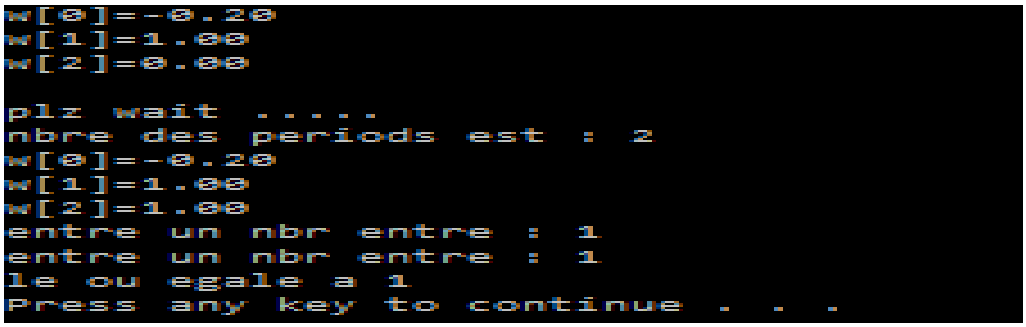
❖ **Figure 1 : Perceptron « OR » :**

Tableau regroupant les résultats :

X_1	X_2	S	O	D	W
0	1	0.8	1	1	(-0.2 ;0 ;1)
1	0	-0.2	0	1	(0.8 ;1 ;1)
1	1	2.8	1	1	(0.8 ;1 ;1)
0	0	0.8	1	0	(-0.2 ;1 ;1)

- Ce perceptron permet d'effectuer le OU logique.
- On a effectuée deux changement le premier changement au niveau de la troisième ligne du tableau (sortie obtenue est différente de la sortie desirée d=1) et le deuxième changement est au niveau de la dernière ligne)
- Pour la phase d'apprentissage on a au moins besoins de deux périodes, dans cette exemple (OR) on converge après deux periodes.

- Analysons le problème : on a un perceptron avec 3 poids donc lors de la programmation il nous faudra un vecteur 3 dimension w[3] , on a aussi trois entrées biais=1(tjrs)+X1+X2 qui changent 4 fois , donc la base d'apprentissage : Entrées=une matrice 4x3 et un vecteurs de désirées de 4.



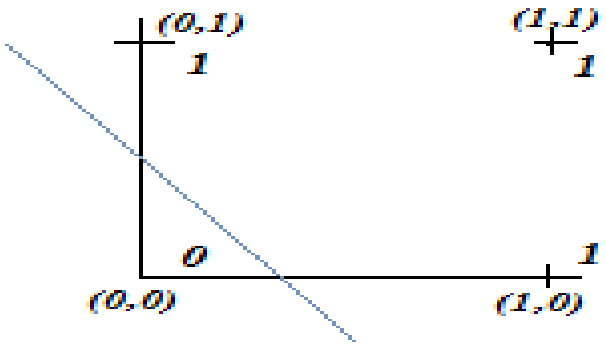
❖ **Figure 2 :Screenshot du « OR » (voir Annexe):**

Remarque :

- On a convergence après deux période (le nombre des périodes et lié étroitement aux poids initiaux).
- μ est une variable empirique telle que :
  - Si μ est grande on converge rapidement ainsi le nombre de période diminue mais il faut faire attention au cas d'oscillation : si on prend μ très grande on risque d'osciller autour de la valeure des poids de convergence.
  - Si μ est petite la convergence est lente, ce qui fait que le nombre de période augmente (précision augmente).
- Cette algorithme est valable que pour les problèmes linéairement séparable telle est le cas de cette exemple :

L'hyperplan est la droite d'équation :

$$\sum W_i X_i = 0 \Longrightarrow X_1 + X_2 - 0.2 = 0$$



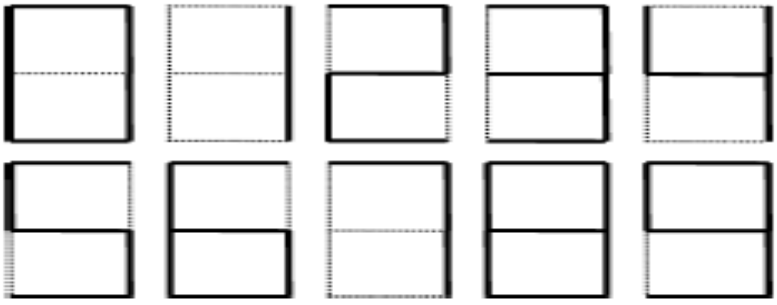
❖ **Figure 3 :L'hyperplan qui divise le plan en deux classes :**

Trois couples (x1,x2) ayant $X_1 + X_2 - 0.2 \rangle 0$ sont de classe 1 et un couples (x1,x2) ayant $X_1 + X_2 - 0.2 \langle 0$.

Exemple :

- **Reconnaissance de la parité des chiffres:**

Trouver les poids permettant de décider si un chiffre est pair ou non .On code les chiffres comme suit (afficheur 7 segments) :

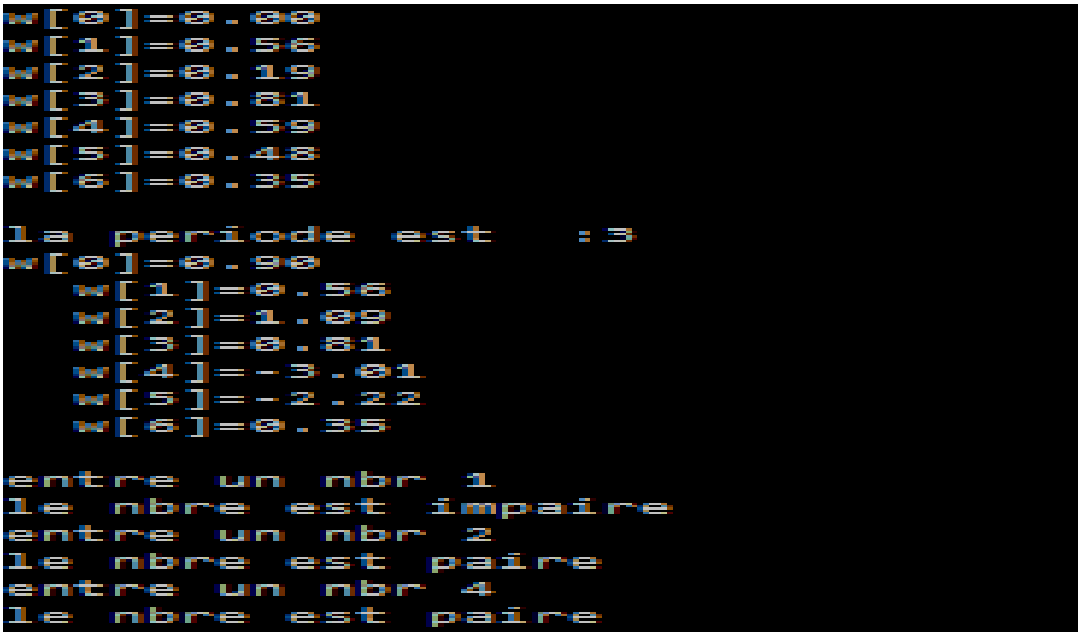


Exemple de codage : 0=(11111011) 1=(01100001).

- En entrées on a dix vecteurs de 8 ,car on code un chiffre suivant l'afficheur 7 segments.
- La sortie desiré est un vecteur de 10 qui represente l'état (parité du nombre) de la sortie d'un nombre désiré : d=(0101010101) avec 0=paire et 1=impaire.

Remarque :

-Si on perturbe les entrées et obtient la sortie desirée alors on peut dire que la méthode est robuste.



❖ **Figure 4: Screenshot « Parité d'un chiffre » (voir annexe):**

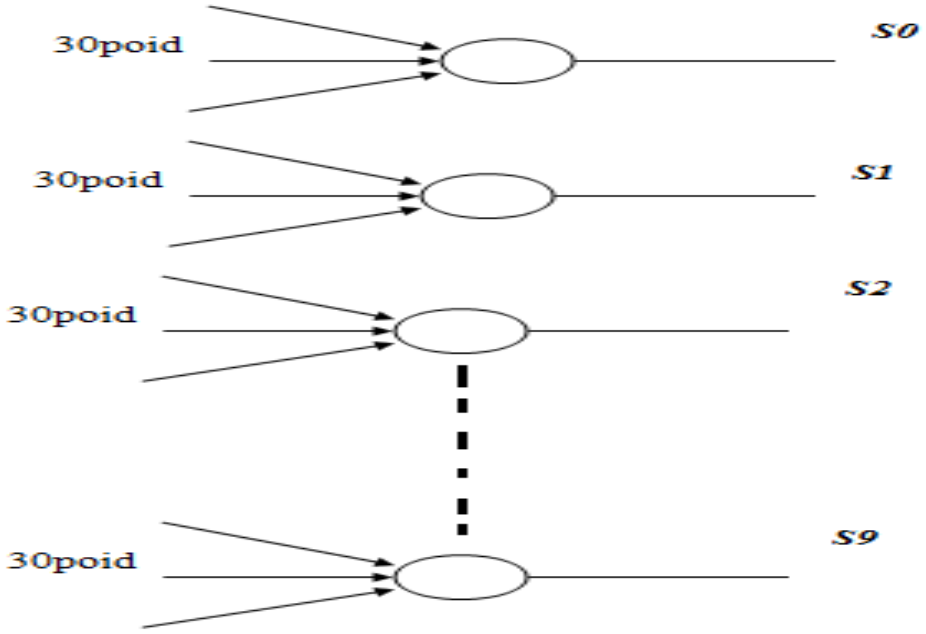
Exemple :

➢ **Reconnaissance des chiffres:**

Pour l'algorithme de la reconnaissance des chiffres on constate que la séparation ne peut se faire qu'en utilisant un réseau de neurone monocouche.

Chaque neurone reconnait un seul chiffre ce qui permet de séparer le chiffre désirée des autres . Ainsi on aura besoin de 10 neurones vue qu'on a dix chiffres (de 0 à 9).

Les poids dépendent fortement de la modélisation des entrées or on a choisit de modéliser les chiffres par des matrices (5x6) ou encore des vecteurs de dimenssion 30 ,ce qui implique que chaque neurone a 30 poids équivalent à 30 entrées donc en total on a 300 poids pour tout le réseau.



❖ **Figure 5 :Réseaux reconnaissance des chiffres:**

```
w[7][10]=-0.84  w[7][11]=-0.24  w[7][12]=-0.86  w[7][13]=0.59   w[7][14]=-1.15
w[7][15]=-1.29  w[7][16]=0.26   w[7][17]=0.26   w[7][18]=1.25   w[7][19]=-0.08
w[7][20]=-1.19  w[7][21]=0.75   w[7][22]=-0.07  w[7][23]=1.39   w[7][24]=-1.62
w[7][25]=-1.26  w[7][26]=-1.52  w[7][27]=-1.99  w[7][28]=-0.30  w[7][29]=-1.26

w[8][0]=-5.39   w[8][1]=-1.70   w[8][2]=-3.03   w[8][3]=-2.26   w[8][4]=-1.72
w[8][5]=2.47    w[8][6]=0.97    w[8][7]=-0.38   w[8][8]=0.21    w[8][9]=-1.92
w[8][10]=-4.57  w[8][11]=6.22   w[8][12]=5.12   w[8][13]=3.44   w[8][14]=-1.85
w[8][15]=7.91   w[8][16]=-2.46  w[8][17]=-2.83  w[8][18]=-1.82  w[8][19]=2.38
w[8][20]=7.64   w[8][21]=0.41   w[8][22]=-0.90  w[8][23]=-1.93  w[8][24]=4.61
w[8][25]=-4.40  w[8][26]=-2.10  w[8][27]=-3.18  w[8][28]=-4.99  w[8][29]=-2.06

w[9][0]=-2.59   w[9][1]=-1.47   w[9][2]=-2.67   w[9][3]=-1.30   w[9][4]=-0.95
w[9][5]=16.00   w[9][6]=0.57    w[9][7]=-0.09   w[9][8]=0.56    w[9][9]=3.18
w[9][10]=-2.68  w[9][11]=2.31   w[9][12]=1.01   w[9][13]=1.44   w[9][14]=-0.87
w[9][15]=-4.44  w[9][16]=0.05   w[9][17]=-1.50  w[9][18]=-0.72  w[9][19]=-1.17
w[9][20]=-4.61  w[9][21]=0.88   w[9][22]=-0.83  w[9][23]=-0.83  w[9][24]=-0.73
w[9][25]=-2.11  w[9][26]=-1.25  w[9][27]=-2.45  w[9][28]=-2.54  w[9][29]=-1.02

 nbr de period pr Cv : 29
comapraison

1 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
```

❖ **Figure 6 :Screenshot de la reconnaissance des chiffres:**

Le programme a reconnu le chiffre 0 ,1 et 2 .

II-Methode d'Adaline -régle Delta (Widrow-Hoff)

Cette méthode consiste à minimiser l'erreur quadratique en fonction des poids en utilisant la méthode de la descente du gradient :

$$e = 1/2 \sum (d - a)^2$$

Algorithme :

1- Initialisation des poids et du seuil (biais) à des valeurs petites (dans ce tp entre 0 et 1).
2- Présentation de la base d'apprentissage : les entrées et la sortie désirée.
3- Calcul de la somme pondérer:

$$\mathbf{a} = \sum \mathbf{w_i x_i}$$

4 Calcule de la sortie obtenue en utilisant la fonction d'activation (Heaviside) :

$$\text{O=signe(a)} \begin{cases} a > 0 \Rightarrow o = 1 \\ a \leq 0 \Rightarrow o = 0 \end{cases}$$

5 Comparaison de la sortie obtenu avec la sortie désirée :

S'ils sont différents changement des poids :

$$w_i(t+1) = w_i(t) + \mu(d - a)x_i$$

6 tant qu'on a pas converger et on a pas atteint le nombre d'itération maximal on retourne a l'étape 3.

Convergence=tout le obtenues de toutes les entrées sont égaux aux sorties désirées ou l'erreur est inférieur a une erreur donnée. .

Exemple :

- **Le OU logique . (voir Annexe)**
- **Reconnaissance de la parité des chiffres .**
- **Reconnaissance des chiffres .**

Pour ces exemples : on a les mêmes données que celles du chapitre précèdent (perceptron *Rosemblath*) , la seul différence est le calcul de l'erreur quadratique et la formule de la modification des poids.(voir Annexe).

Remarque :

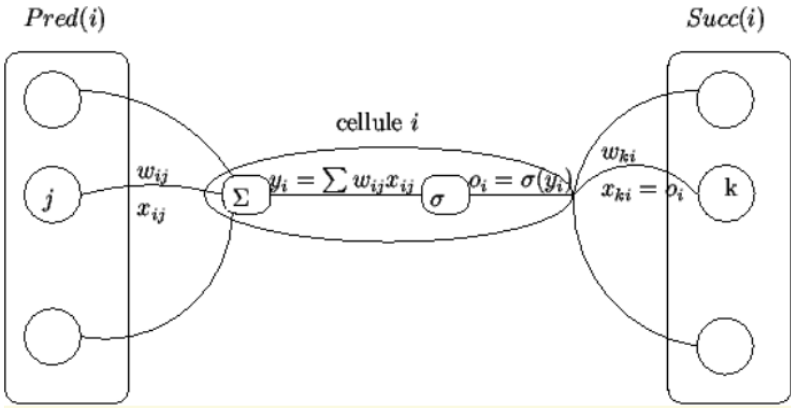
- On a convergence absolue avec cette méthode pour les problèmes linéairement séparable.

- si l'erreur est tres grande on aura un résultat faux.

- si les poids et le seuil prennent deux fois les mêmes valeurs sans que le perceptron ait appris et alors que tous les exemples ont été présentés cela signifie qu'on a une convergence at donc le problème n'est pas linéairement séparable .

III-Methode d'apprentissage « rétropropagation » :

Pour pouvoir résoudre des problèmes complexes (multicouches) et non linéaires ,ou les méthodes d'apprentissage précédentes ne sont pas valables parceque la superposition de plusieurs perceptrons nous pause problème puisqu'on ne connait pas la sortie désirée des couches cachées . On a introduit la rétro-propagation du gradient (Delta Généralisée) qui est une méthode d'apprentissage pour les réseaux multicouches.



❖ **Figure 7 : Réseaux multicouche :**

Algorithme :

Initialiser aléatoirement les coefficients wij dans [-0.5 ; 0.5]

• Répéter

• Prendre un exemple (x, d) de S

• Calculer la sortie o

• Pour toute cellule de sortie i

• $\delta i = \sigma'(yi) * (di - oi) = oi * (1 - oi) * (di - oi)$

• Fin Pour

• Pour chaque couche de q − 1 à 1

• Pour chaque cellule i de la couche courante

• $\delta i = \sigma'(yi) * [\Sigma(k \in Succ(i)) (\delta k * wki)]$

$= oi * (1 - oi) * [\Sigma(k \in Succ(i)) (\delta k * wki)]$

• Fin Pour

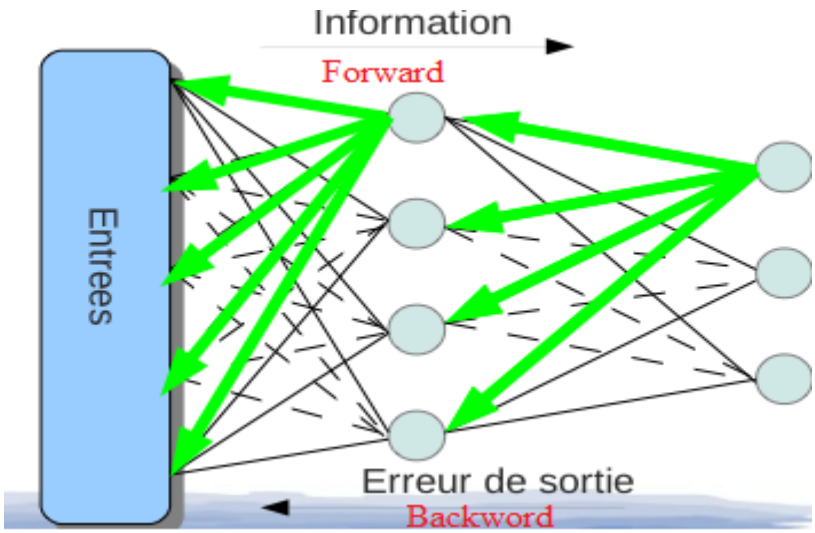
•Fin Pour

• Pour tout poids wij

• $wij = wij + \varepsilon * \delta i * xij$

• Fin Pour

• Fin Répéter

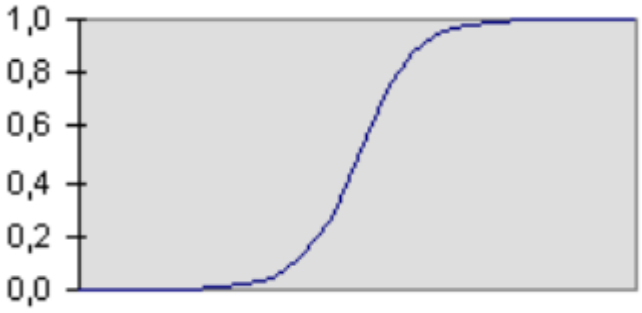


❖ **Figure 8 :Principe de la Rétro-propagation :**

Cette méthode consiste à rétropropagée l'erreur de sortie aux couches cachées afin de faire une mise à jours des poids :

- Propagation de l'entrée jusqu'à la sortie.
- Calcul de l'erreur en sortie.
- Rétro-propagation de l'erreur jusqu'aux entrées.

On a besoin d'une fonction dérivable et continue pour pouvoire retropropager l'erreur .(Fonction sigmoid permet une séparation meilleur) :



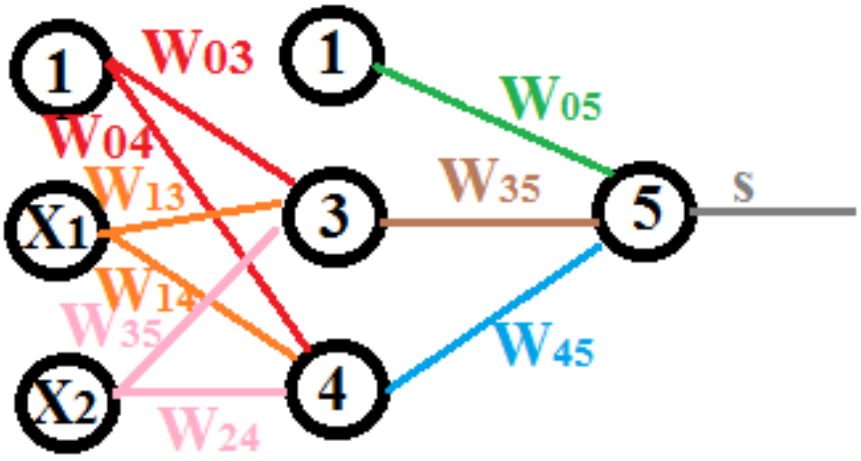
$$sigm(x) = \frac{1}{1+e^{-kx}}$$ tel que : K>0 (on prend k=1).

Remarque : Vue la nature réelle de la sortie de la couche de décision ,on adopte comme condition d'arrêt l'erreur quadratique.

Exemple :

- **OU exclusive « XOR »:**

Soit le réseaux :



❖ **Figure 9 : XOR (Rétro-propagation) :**

Les poids initiaux : $W_{13} = 0.5 / W_{14} = 0.9$
$W_{23} = 0.4 / W_{24} = 1$
$W_{35} = -1.2 / W_{45} = 1.1$
$W_{03} = -0.8 / W_{04} = 0.1$
$W_{05} = -0.3$

Les calculs d'une seul ittération :

- Propagation de l'entrée jusqu'à la sortie :

$y_3 = 1\times -0.8 + 1\times 0.5 + 1\times 0.4 + = 0.1 \rightarrow o_1 = sigm(-0.1) = 0.524.$
$y_4 = 2 \rightarrow o_2 = 0.880.$
$y_5 = 0.039 \rightarrow o_5 = 0.5097.$

- Calcul de l'erreur de sortie : $\delta_i = (d - o_i) o_i (1 - o_i)$

$$\delta_5 = -0.127$$

- Rétropropagation de l'erreur jusqu'aux entrées :

$$\delta_i = o_i (1 - o_i) \sum_{k = succesde(i)} w_{ik} \delta_k$$

$$\delta_4 = -0.0147$$
$$\delta_3 = 0.038$$

❖ Mise à jours des poids : $W_{ij} = W_{ij} + \Delta W_{ij}$

Calcul des ΔW_{ij} :
$\Delta W_{05} = \mu \times \delta_5 \times 1 = -0.0127$
$\Delta W_{35} = \mu \times \delta_5 \times o_3 = -0.0067$
$\Delta W_{45} = \mu \times \delta_4 \times o_4 = -0.0112$
$\Delta W_{13} = \mu \times \delta_3 \times 1 = 0.0038$
$\Delta W_{23} = \Delta W_{03} = 0.0038$
$\Delta W_{14} = \Delta W_{24} = \Delta W_{04} = -0.0015$

Changement des poids :

$W_{05} = -0.3127$

$W_{04} = 0.098$

$W_{05} = -0.796$

$W_{24} = 0.998$

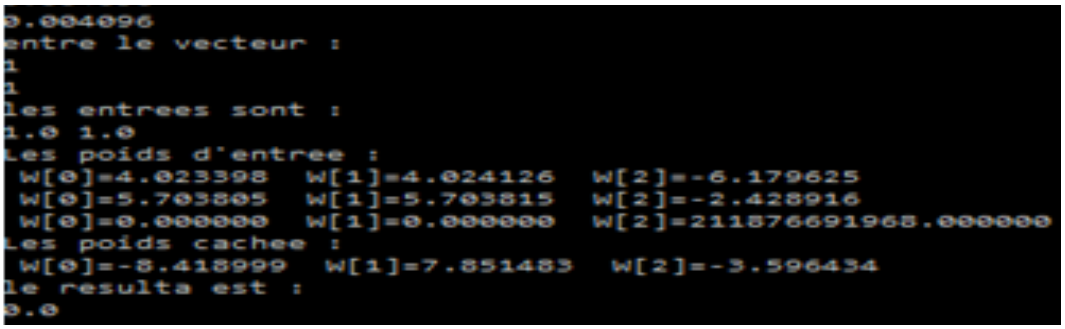
$W_{35} = -1.2067$

$W_{45} = 1.088$

$W_{15} = 0.5038$

$W_{14} = 0.898$

$W_{23} = 0.403$

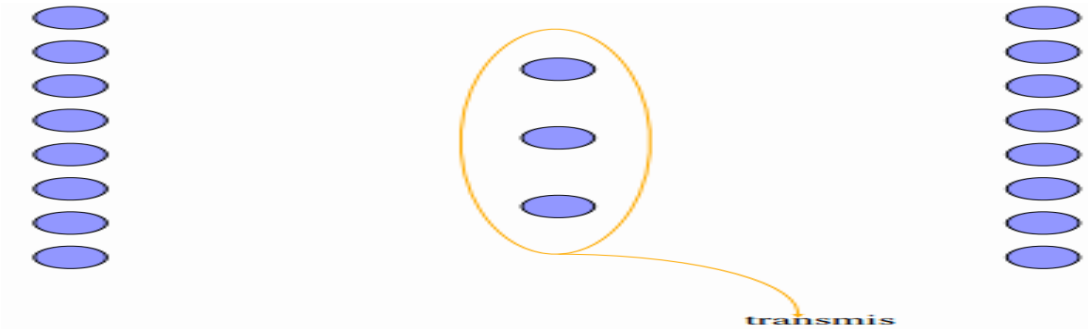


❖ **Figure 10 : Screenshot « XOR » :**

➢ **Encodeur (8-3-8) :**

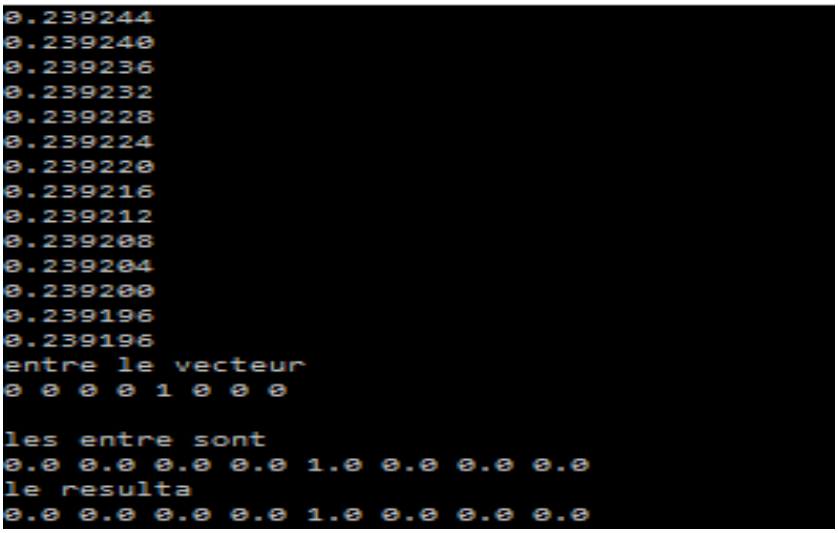
C'est modèle réduit de l'exemple Diabolo qui compresser les images .Il permet de coder 8 bits en 3 bits et les décoder ensuite en 8 bits.

On a la possibilité de coder (2^3)-1=7 chiffres.



❖ **Figure 11 : Architecture de l' Encodeur 8-3-8 :**

La couche d'entrée et la couche cachée permet de coder les 8 bits d'entrées en 3 bits , qui seront décoder en 8 bits par la couche de sortie .



❖ **Figure 12 : Screenshot Encodeur (avec valeurs majorées) :**

Conclusion :

Dans ce TP on a pu résoudre des problèmes tel :OR , XOR et autres à l'aide de plusieurs méthodes d'apprentissage supervisé , comme le perceptron linéaire (*Rosemblath* 1958-1962) , la rétro-propagation du gradient. Chaque méthode est caractérisée par :

Perceptron linéaire permet de résoudre des problèmes linéairement séparables (réseaux monocouches) .

Rétro-propagation du gradient permet de résoudre des problèmes qui ne sont pas linéairement séparable en plus celles linéairement séparables (réseaux multicouches).

Annexe :

❖ Perceptron « OU » logique :

```
#include <stdio.h>
#include <stdlib.h>
float w[3];/*les poids*/
float u=1;/*variable empirique generalement comprise entre 0 et 1*/
/*Base d'apprentissage(etape 2)*/
int E[4][3]={{1,0,1},
             {1,1,0},
             {1,1,1},
             {1,0,0}};/*les entrées*/
int des[4]={1,1,1,0}; /*sorties desirées*/

/* Phase d'apprentissage    */
void phase_apprentissage(){
     int flag=1;/*vérification des changements des poids*/
     int i,j,period=0;
     float som,ob;
     while(flag){
          flag=0;
          for(i=0;i<4;i++){
               som=0;
               for(j=0;j<3;j++)
                    som+=E[i][j]*w[j]; /*etape 3*/
                    ob=(som<0)?0:1; /*etape 4*/
                    if(ob!=des[i]) /*etape 5*/
                    {    for(j=0;j<3;j++) w[j]=w[j]+u*((des[i]-ob)*E[i][j]);
                      flag=1;
                    }
          }period++;
     }
     printf(" Le nombre des periodes est : %d ",period);
}
void phase_utilisation(void){
      int j,n1,n2;float som=0,o;
     printf("entre un nbr entre : ");
     scanf("%d",&n1);
     printf("entre un nbr entre : ");
     scanf("%d",&n2);
     som+=w[0]+w[1]*n1+w[2]*n2;
     o=(som<0)?0:1;
      if(o==1)
          printf("le ou egale a 1\n");
      else
          printf("le ou est egale a zero \n");
}
int main(){int i,k;
/*initialisation des poids (etape1)*/
w[0]=-0.2;
w[1]=1;
w[2]=0;
puts("Les poids initiaux sont :");
for(i=0;i<3;i++){
printf("w[%d]=%.2f\n",i,w[i]);
}
printf("\n");
phase_apprentissage();
printf("\n");
puts("Les poids aprés apprentissage sont :");
for(i=0;i<3;i++) printf("w[%d]=%.2f\n",i,w[i]);
do{
phase_utilisation();
getchar();
system("cls");
printf("si vous voulez continue tapez 1 si no tapez zero ");
scanf("%d",&k);
}while(k!=0);
return 0;
}
```

❖ Adaline « OU » logique :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
float w[3];/*les poids*/
float u=1;/*variable empirique generalement comprise entre 0 et 1*/
float eps=0.1;/*erreur*/
/*Base d'apprentissage(etape 2)*/
int E[4][3]={{1,0,1},
             {1,1,0},
             {1,1,1},
             {1,0,0}};/*les entrées*/
int des[4]={1,1,1,0}; /*sorties desirées*/

/* Phase d'apprentissage    */
void phase_apprentissage(){
     int flag=1;/*vérification des changements des poids*/
     int i,j,period=0;
     float som,ob,er;
     while(flag||er<eps){
          er=0;
          flag=0;
          for(i=0;i<4;i++){
               som=0;
               for(j=0;j<3;j++)
                    som+=E[i][j]*w[j];/*etape 3*/
                    er+=0.5*pow((des[i]-som),2);
                    ob=(som<0)?0:1; /*etape 4*/
                    if(ob!=des[i]) /*etape 5*/
                    {    for(j=0;j<3;j++) w[j]=w[j]+u*((des[i]-som)*E[i][j]);
                      flag=1;
                    }
          }period++;
     }
      printf(" Le nombre des periodes est : %d ",period);
}

void phase_utilisation(void){
      int j,n1,n2;float som=0,o;
      printf("entre un nbr entre : ");
      scanf("%d",&n1);
      printf("entre un nbr entre : ");
      scanf("%d",&n2);
      som+=w[0]+w[1]*n1+w[2]*n2;
      o=(som<0)?0:1;
       if(o==1)
          printf("le ou egale a 1\n");
       else
          printf("le ou est egale a zero \n");
}
int main(){int i,k;
/*initialisation des poids (etape1)*/
w[0]=-1;
w[1]=0;
w[2]=-0.6;
puts("Les poids initiaux sont :");
for(i=0;i<3;i++){
printf("w[%d]=%.2f\n",i,w[i]);
}
printf("\n");
phase_apprentissage();
printf("\n");
puts("Les poids aprés apprentissage sont :");
for(i=0;i<3;i++) printf("w[%d]=%.2f\n",i,w[i]);
do{
phase_utilisation();
getchar();
system("cls");
printf("si vous voulez continue tapez 1 si no tapez zero ");
scanf("%d",&k);
}while(k!=0);
return 0;
}
```

❖ Perceptron reconnaissance de la parité d'un chiffre :

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#define Nmax 10
#define Mmax 8
float w[7],seuil=1,u=0.5;
/*base d'apprentissage*/
int E[Nmax][Mmax]={{1,1,1,1,1,1,0,1},
                   {0,1,1,0,0,0,0,1},
                   {1,1,0,1,1,0,1,1},
                   {1,1,1,1,0,0,1,1},
                   {0,1,1,0,0,1,1,1},
                   {1,0,1,1,0,1,1,1},
                   {1,0,1,1,1,1,1,1},
                   {1,1,1,0,0,0,0,1},
                   {1,1,1,1,1,1,1,1},
                   {1,1,1,1,0,1,1,1}};
int des[10]={0,1,0,1,0,1,0,1,0,1};
```

```
'*phase d'apprentissage*/
oid phase_apprentissage(){
    int flag=1,i,j,period=0;
    float som,ob;
    while(flag){
        flag=0;
        for(i=0;i<Nmax;i++){
            som=0;
            for(j=0;j<Mmax;j++)
                som+=E[i][j]*w[j];
                som=som-seuil;
                ob=(som<0)?0:1;
                if(ob!=des[i]){
                    for(j=0;j<Mmax;j++) w[j]=w[j]+u*((des[i]-ob)*E[i][j]);
                flag=1;
                }
        }period++;
    }
    printf("la periode est  :%d\n",period);

oid phase_utilisation(){
     int j,n;
     float som=0,ob;
    do {som=0;
        printf("entre un nombre ");
        scanf("%d",&n);
            for(j=0;j<Mmax;j++)  som+=E[n][j]*w[j];
            som=som-seuil;
            ob=(som<0)?0:1;
            if(ob==1)
              printf("le nombre est impaire\n");
            else
             printf("le nombre est paire\n");}while(n>0&&n<10);
```

```
int main(){
int i;
/*initialisation des poids      */
for(i=0;i<7;i++){
     w[i]=(double)(rand())  /  RAND_MAX;
}
puts("Les poids initiaux sont :");
for(i=0;i<7;i++){
printf("w[%d]=%.2f\n",i,w[i]);
}
puts("");
phase_apprentissage();
puts("Les poids aprés apprentissage sont :");
for(i=0;i<7;i++){
printf("w[%d]=%.2f\n   ",i,w[i]);
}
puts("");
phase_utilisation();
return 0;
}
```

❖ Perceptron reconnaissance des chiffres :

```
#include<time.h>
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
# define l 10 // 10 entrees
# define c 30 // 30 poids pour chaque neurone
int k=0;
float u=0.9;
int sortie[4][10];//sortie apres phase d'utilisation
/*base d'apprentissage*/
float des[l][l]={{1,0,0,0,0,0,0,0,0,0},
                 {0,1,0,0,0,0,0,0,0,0},
                 {0,0,1,0,0,0,0,0,0,0},
                 {0,0,0,1,0,0,0,0,0,0},
                 {0,0,0,0,1,0,0,0,0,0},
                 {0,0,0,0,0,1,0,0,0,0},
                 {0,0,0,0,0,0,1,0,0,0},
                 {0,0,0,0,0,0,0,1,0,0},
                 {0,0,0,0,0,0,0,0,1,0},
                 {0,0,0,0,0,0,0,0,0,1},
                 };
```

```
float E[l][c]={{1,1,1,1,1,
               1,0,0,0,1,
               1,0,0,0,1,
               1,0,0,0,1,
               1,0,0,0,1,
               1,1,1,1,1},

               {0,0,1,0,0,
               0,0,1,0,0,
               0,0,1,0,0,
               0,0,1,0,0,
               0,0,1,0,0,
               0,0,1,0,0},

               {1,1,1,1,1,
               0,0,0,0,1,
               1,1,1,1,1,
               1,0,0,0,0,
               1,0,0,0,0,
               1,1,1,1,1},

               {1,1,1,1,1,
               0,0,0,0,1,
               1,1,1,1,1,
               0,0,0,0,1,
               0,0,0,0,1,
               1,1,1,1,1},


              {1,0,0,0,0,
              1,0,0,0,0,
              1,0,0,1,0,
              1,1,1,1,1,
              1,0,0,1,0,
              1,0,0,1,0},

              {1,1,1,1,1,
              1,0,0,0,0,
              1,1,1,1,1,
              0,0,0,0,1,
              0,0,0,0,1,
              1,1,1,1,1},
```

```
{1,1,1,1,1,//6
1,0,0,0,0,
1,0,0,0,0,
1,1,1,1,1,
1,0,0,0,1,
1,1,1,1,1},

{1,1,1,1,1,
0,0,0,1,0,
0,0,0,1,0,
0,0,1,1,1,
0,0,0,1,0,
0,0,0,1,0},
```

```
{1,1,1,1,1,
1,0,0,0,1,
1,1,1,1,1,
1,0,0,0,1,
1,0,0,0,1,
1,1,1,1,1},

{1,1,1,1,1,
1,0,0,0,1,
1,1,1,1,1,
0,0,0,0,1,
0,0,0,0,1,
1,1,1,1,1},};
```

```
//les entrees a tester apres apprentissage avec quelque perturbation
 float F[l][c]={{1,1,0,1,1, //0
                 1,0,0,0,1,
                 1,0,0,0,0,
                 1,0,0,0,1,
                 1,0,0,0,1,
                 1,1,0,1,1},

                 {0,0,1,0,0,//1
                 0,0,1,0,0,
                 0,0,0,0,1,
                 0,0,0,1,0,
                 0,0,1,0,0,
                 0,0,1,0,0},
```

```
                {1,1,1,1,1,//2
                0,0,0,0,1,
                1,1,1,1,1,
                1,0,1,0,0,
                0,0,0,0,1,
                1,1,1,0,1}};
float w[l][e];
float som,o;
int flag=1,periode=0;int i,j;


main(){
 srand(time(NULL));
  for(i=0;i<l;i++)
    for(j=0;j<c;j++)
  w[i][j]=(float)rand()/RAND_MAX;
 printf("l'apprentissage\n");
 while(flag&&periode<100)
 {flag=0;
  for(j=0;j<10;j++)
  {for(k=0;k<10;k++)
        {som=0.;
          for(i=0;i<30;i++)
             som=som+E[j][i]*w[k][i];
          o=(som<0)?0:1;
        if(o!=des[j][k])
        {for(i=0;i<30;i++)
          w[k][i]=w[k][i]+u*((des[j][k]-o)*E[j][i]);
        flag=1;
        }}
  }periode++;
  printf("\n  Les nouveaux poids :\n");
  for(i=0;i<l;i++){
    printf("\n");
   for(j=0;j<c;j++) printf("w[%d][%d]=%.2f \t",i,j, w[i][j]);}
   printf("\n Nombre de periode est : %d\n",periode);
        }
```

```
        }
printf(" Phase d'utilisation : \n");
     for(j=0;j<3;j++)
 for(k=0;k<10;k++)
 {som=0.;
          for(i=0;i<30;i++)
         som=som+F[j][i]*w[k][i];
                sortie[j][k] = (som<0)?0:1;
                  }
        for(j=0;j<3;j++)
            { printf("\n");
                for(k=0;k<10;k++){
                    printf("%d ",sortie[j][k]);
                }
            }
        getch();
}
```

❖ Rétropropagation « XOR » :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define sigm(x)     1/(1 + exp(-(double)x))
#define entre          3
#define cache     3
#define sortie         1
#define EPSILON     0.001
#define entre_test    4
#define u 0.1

int desire[entre_test]={0,1,1,0};
int E[entre_test][entre]={{1,1,1},{1,0,1},{0,1,1},{0,0,1}};
float er;
typedef struct noreune{
float *w,o,delta;
}noreune;

noreune couche_cache[cache],couche_sortie[sortie];

void intialise(){
int i,j;
couche_cache[cache-1].o=1.;
for(i=0;i<cache;i++){
    couche_cache[i].w=(float*)malloc(entre*sizeof(float));
    couche_cache[i].delta=0.;
}
for(i=0;i<sortie;i++){
    couche_sortie[i].w=(float*)malloc(cache*sizeof(float));
    couche_sortie[i].delta=0.;
}
}
```

```
void forward(float x[]){
int i,j;
float som;

for(i=0;i<cache-1;i++){
    som=0.;
    for(j=0;j<entre;j++){
       som+=x[j]*couche_cache[i].w[j];
    }
   couche_cache[i].o=sigm(som);

}

for(i=0;i<sortie;i++){
    som=0.;
    for(j=0;j<cache;j++){
      som+=couche_cache[j].o*couche_sortie[i].w[j];
    }
   couche_sortie[i].o=sigm(som);
}

}
float backward(int k){
int i,j;
float som;
for(i=0;i<sortie;i++){
    couche_sortie[i].delta=(desire[k]-couche_sortie[i].o)*couche_sortie[i].o*(1-couche_sortie[i].o);
    er+=0.5*(desire[k]-couche_sortie[i].o)*(desire[k]-couche_sortie[i].o);
}
for(i=0;i<cache;i++){
    som=0;
   for(j=0;j<sortie;j++){
    som+=couche_sortie[j].delta*couche_sortie[j].w[i];
   }

  couche_cache[i].delta=couche_cache[i].o*(1-couche_cache[i].o)*som;
}
 return er;
}
void correction(float x[]){
 int i,j;
for(i=0;i<cache-1;i++){
    for(j=0;j<entre;j++){
        couche_cache[i].w[j]+=u*couche_cache[i].delta*x[j];
    }
}

for(i=0;i<sortie;i++){
    for(j=0;j<cache;j++){
        couche_sortie[i].w[j]+=u*couche_sortie[i].delta*couche_cache[j].o;
    }
}

}

void aprentissage(){
int i,j,periode=1,t;
float x[entre];
```

```
intialise();
couche_cache[0].w[0]=0.5;
couche_cache[0].w[1]=0.4;
couche_cache[0].w[2]=-0.8;
couche_cache[1].w[0]=0.9;
couche_cache[1].w[1]=1.;
couche_cache[1].w[2]=0.1;
couche_sortie[0].w[0]=-1.2;
couche_sortie[0].w[1]=1.1;
couche_sortie[0].w[2]=-0.3;
do{
er=0.;
for(j=0;j<entre_test;j++){
for(i=0;i<entre;i++)x[i]=(float)E[j][i];
forward(x);
er=backward(j);
if(er<EPSILON){
    t=0;
    break;}
else
    t=1;
correction(x);
}
printf("%f ",er);
puts("");
periode++;
}while(t&&periode<10000);}

void test(){
float y[entre];
int i;
y[2]=1.;
printf("entre le vecteur \n");
for(i=0;i<entre-1;i++){
    scanf("%f",&y[i]);
}
for(i=0;i<entre-1;i++){
    printf("%0.1f ",y[i]);
}
puts("");
forward(y);
for(i=0;i<sortie;i++){
    printf("%0.1f ",couche_sortie[i].o);
}
}
int main()
```

```
int main()
{int i,j;
     aprentissage();
     printf("%f ",er);
     do{
     test();
     puts("");
    getchar();
     puts("");
}while(1);

     return 0;
}
```

❖ Rétropropagation Encodeur :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define sigm(x)     1/(1 + exp(-(double)x))
#define entre          8
#define cache     3
#define sortie         8
#define EPSILON     0.001
#define entre_test    8
#define u 0.9

int desire[entre_test][entre]={{1,0,0,0,0,0,0,0},
                               {0,1,0,0,0,0,0,0},
                               {0,0,1,0,0,0,0,0},
                               {0,0,0,1,0,0,0,0},
                               {0,0,0,0,1,0,0,0},
                               {0,0,0,0,0,1,0,0},
                               {0,0,0,0,0,0,1,0},
                               {0,0,0,0,0,0,0,1}};
int E[entre][entre_test]={{1,0,0,0,0,0,0,0},
                          {0,1,0,0,0,0,0,0},
                          {0,0,1,0,0,0,0,0},
                          {0,0,0,1,0,0,0,0},
                          {0,0,0,0,1,0,0,0},
                          {0,0,0,0,0,1,0,0},
                          {0,0,0,0,0,0,1,0},
                          {0,0,0,0,0,0,0,1}};
```

```
float er;
typedef struct noreune{
float *w,o,delta;
}noreune;

noreune couche_cache[cache],couche_sortie[sortie];

void intialise(){
int i,j;
for(i=0;i<cache;i++){
     couche_cache[i].w=(float*)malloc(entre*sizeof(float));
     couche_cache[i].delta=0.;
     for(j=0;j<entre;j++){
      couche_cache[i].w[j]=-0.5 + ((float)rand() / RAND_MAX);
     }
}
for(i=0;i<sortie;i++){
     couche_sortie[i].w=(float*)malloc(cache*sizeof(float));
     couche_sortie[i].delta=0.;
     for(j=0;j<cache;j++){
      couche_sortie[i].w[j]=-0.5 + ((float)rand() / RAND_MAX);
     }
}
}
```

```
void forward(float x[]){
int i,j;
float som;

for(i=0;i<cache;i++){
    som=0.;
    for(j=0;j<entre;j++){
       som+=x[j]*couche_cache[i].w[j];
    }
   couche_cache[i].o=sigm(som);

}

for(i=0;i<sortie;i++){
    som=0.;
    for(j=0;j<cache;j++){
      som+=couche_cache[j].o*couche_sortie[i].w[j];
    }
   couche_sortie[i].o=sigm(som);
}

}
float backward(int k){
int i,j;
float som;
for(i=0;i<sortie;i++){
    couche_sortie[i].delta=(desire[k][i]-couche_sortie[i].o)*couche_sortie[i].o*(1-couche_sortie[i].o);
    er+=0.5*(desire[k][i]-couche_sortie[i].o)*(desire[k][i]-couche_sortie[i].o);
}
for(i=0;i<cache;i++){
    som=0;
   for(j=0;j<sortie;j++){
    som+=couche_sortie[j].delta*couche_sortie[j].w[i];
   }
```

```
 couche_cache[i].delta=couche_cache[i].o*(1-couche_cache[i].o)*som;
}
return er;
}
void correction(float x[]){
int i,j;
for(i=0;i<cache;i++){
    for(j=0;j<entre;j++){
        couche_cache[i].w[j]+=u*couche_cache[i].delta*x[j];
    }
}

for(i=0;i<sortie;i++){
    for(j=0;j<cache;j++){
        couche_sortie[i].w[j]+=u*couche_sortie[i].delta*couche_cache[j].o;
    }
}

}

void aprentissage(){
int i,j,periode=1,t;
float x[entre];
intialise();
do{
er=0.;
for(j=0;j<entre_test;j++){
for(i=0;i<entre;i++)x[i]=(float)E[j][i];
forward(x);
er=backward(j);
if(er<EPSILON){
     t=0;
     break;}
else
     t=1;
correction(x);
}
printf("%f ",er);
puts("");
periode++;
}while(t&&periode<10000);}

]void test(){
 float y[entre];
 int i;
 printf("entre le vecteur \n");
]for(i=0;i<entre;i++){
     scanf("%f",&y[i]);
}
 puts("");
]for(i=0;i<entre;i++){
     printf("%0.1f ",y[i]);
}
 puts("");
 forward(y);
]for(i=0;i<sortie;i++){
     printf("%0.1f ",couche_sortie[i].o);
}
}
```

```
int main()
{int i,j;
     aprentissage();
     printf("%f ",er);
     do{
     test();
     puts("");
    getchar();
     puts("");
}while(1);

     return 0;
}
```