1. **Introduction**

Nous allons concevoir un réseau de neurones capable d’identifier quelques formes conventionnelles, pour des raisons de simplicité on a choisi les formes suivantes : triangle, rectangle et cercle.

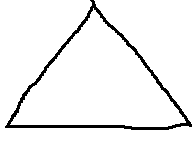
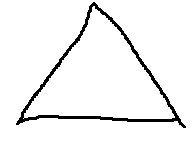
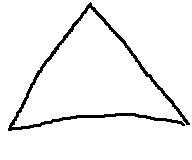
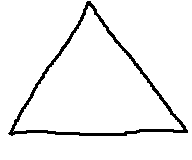
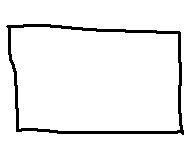
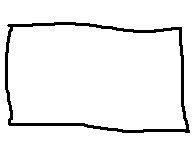
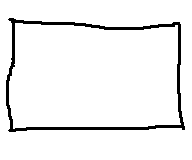
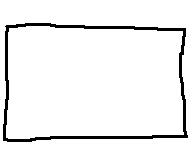
Le réseau sera un réseau de type MLP (MultiLayer Perceptron).

En premier lieu nous allons préparer les échantillons (images) puis on procède à l’apprentissage et enfin la reconnaissance des formes déjà mentionnées.

Le travail sera fait sous Matlab en exploitant les fonctions de la biblo Neural Network Toolbox.

1. **Préparation des données d’apprentissage et de simulation**

Les données utilisées sont 4 images pour chaque type de forme (12 en totales) dont 3 de chaque forme (9 en total) seront utilisées pour l’apprentissage et une de chaque forme (3en total) seront utilisées pour le test.

****

r4.bmp

r3.bmp

r1.bmp

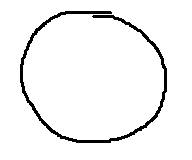
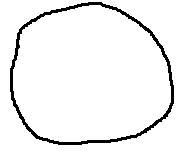
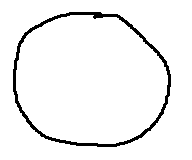
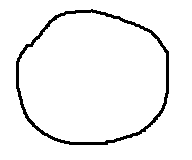
r2.bmp

t1.bmp

t3.bmp

t4.bmp

t2.bmp



c4.bmp

c3.bmp

c2.bmp

c1.bmp

Les images sont des images binaires de type bitmap avec une résolution de 192 x 160.

L’une des étapes indispensables pour la préparation des données est traiter les images afin d’avoir des images unifiées en dimension et qui ne présente pas de parasites ainsi pour des raisons de simplicités tous les images sont déjà de même dimension, ne présente pas de parasites et les formes sont presque de même dimension relativement à leurs résolutions.

Le critère choisi pour remplir la matrice d’entrée est qu’on divise les lignes d’une image par lot de 16 lignes, on parcourt les 16 lignes et si nous avons un pixel noir alors on stocke la valeur de son abscisse (numéro de la colonne) et on l’accumule dans une matrice P.

Comme notre image est de 160 lignes donc nous aurons 10 (160/16) entrées.

Donc la matrice P contient 10 lignes et 9 colonnes (nombres des échantillons des données d’entrées), chaque colonne représente les 10 caractères d’une image.

De même il faut préparer une matrice T qui va indiquer au réseau de neurones lors de l’apprentissage s’il s’agit d’une forme rectangulaire, triangulaire ou circulaire.

La matrice T est une matrice 3 x 9, tel que si on écrit la valeur 1 alors la forme associée est vraie et si on écrit -1 alors la forme associée est fausse.

La première ligne représente l’état de la forme triangulaire, la deuxième ligne représente l’état de la forme rectangulaire et la troisième ligne représente l’état de la forme circulaire.

Le code associé est :

% préparation données d'apprentissage

Num\_Inputs=10 ;% nombre de critères

P=zeros(Num\_Inputs,9);% input matrix

T=zeros(3,9);

for h=1:9 % 9 boucles pour 9 images

switch h

case 1

Img = imread('t1.bmp');

Target=[1;-1;-1] ;

case 2

Img = imread('t2.bmp');

Target=[1;-1;-1] ;

case 3

Img = imread('t3.bmp');

Target=[1;-1;-1] ;

case 4

Img = imread('r1.bmp');

Target=[-1;1;-1] ;

case 5

Img = imread('r2.bmp');

Target=[-1;1;-1];

case 6

Img = imread('r3.bmp');

Target=[-1;1;-1];

case 7

Img = imread('c1.bmp');

Target=[-1;-1;1];

case 8

Img = imread('c2.bmp');

Target=[-1;-1;1];

case 9

Img = imread('c3.bmp');

Target=[-1;-1;1];

end %fin switch

T(:,h)=Target ; % remplissage matrice T

[Num\_Row,Num\_column] = size(Img) ;% determiner la dimension de l'image

% remplissage matrices d'entrée

for i=1:Num\_Inputs

for j=(((Num\_Row/Num\_Inputs)\*(i-1))+1) : ((Num\_Row/Num\_Inputs)\*(i))

for k=1 : Num\_column

if Img(j,k)==0

P(i,h)=P(i,h)+k ;

end

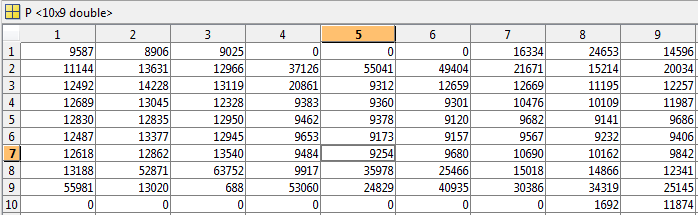
end

end

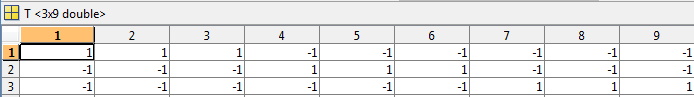
end

end % fin boucle for

après execution, la matrice P est :



et la matrice T est :



La matrice de simulation S est de 10 x 3, qui représente les 10 critères de 3 images nécessaire pour la simulation.

Le code associé est :

% préparation données de simulation

S=zeros(Num\_Inputs,3);% matrice de simulation

for h=1:3 % 3 boucles pour 3 images de simulation

switch h

case 1

Img = imread('t4.bmp');

case 2

Img = imread('r4.bmp');

case 3

Img = imread('c4.bmp');

end %fin switch

[Num\_Row,Num\_column] = size(Img) ;

% initiation for input matrix

for i=1:Num\_Inputs

for j=(((Num\_Row/Num\_Inputs)\*(i-1))+1) : ((Num\_Row/Num\_Inputs)\*(i))

for k=1 : Num\_column

if Img(j,k)==0

S(i,h)=S(i,h)+k ;

end

end

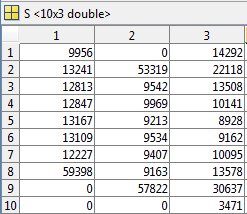
end

end

end % fin boucle for

%fin préparation données de simulation

Après exécution la matrice S est :



A ne pas oublier de normalisation les valeurs des matrices P et S entre 1 et -1 , le code associé et le suivant :

% Normalisation des Matrices

A=[P,S] ;

maxi=max(max(A));

mini=min(min(A));

[a,b]=size(A);

for i=1:a

for j=1:b

AN(i,j)=2\*(A(i,j)/(maxi-mini))-1;

end

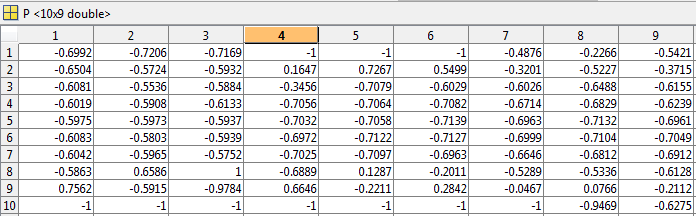
end

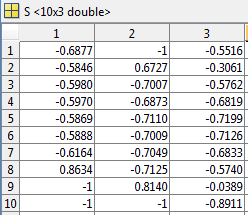
P=AN(:,1:9);

S=AN(:,10:12) ;

%fin Normalisation des matrice

Après exécution les matrices P et S sont :





1. **Apprentissage et simulation**

Le réseau de neurones est un réseau MLP dont la couche caché contient 20 neurones dont la fonction d’activation est la fonction tansig et la couche de sorties contient 3 neurones dont la fonction d’activation est purelin

Le code de l’étape de création et d’apprentissage est le suivant :

% étape de création de réseau de neurones et d'apprentissage

Num\_Neuron\_Hidden=20 ;%nombre de neurones de la couche cachée

net = newff(P,T,Num\_Neuron\_Hidden,{},'traingd');% Num\_Neuron\_Hidden=10

net=init(net); % reintialisation des poids et des bias

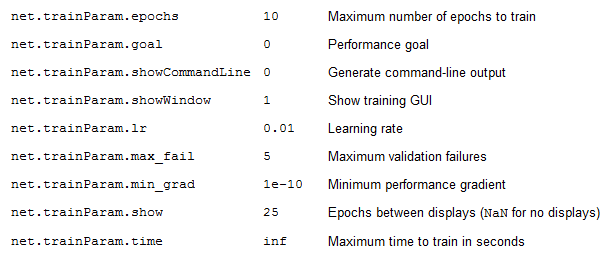
net.trainparam.epochs=250;% nombre maximal 'itération

net.trainparam.goal=0.0001; % erreur à respecter

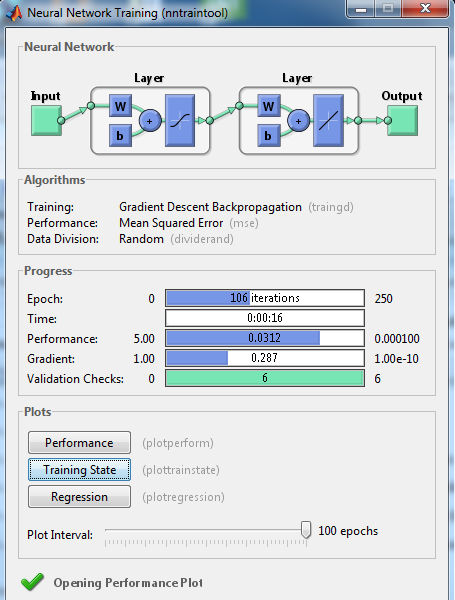
net=train(net,P,T); % lancement de l'apprentissage

% fin apprentissage

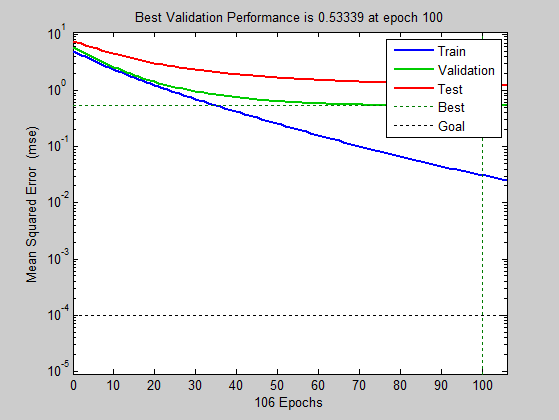
Ce qu’il faut noter c’est que l’algorithme de retro propagation utilisé est l’algorithme du gradient descendant dont les paramètres par défaut sont :



Après exécution l’outil nntraintool nous affiche :



Avec une variation de l’erreur quadratique :



Pour l’étape de simulation on se dispose de 3 échantillons dont on sait d’avant que le premier est un triangle, le deuxième est un rectangle et le troisième est un cercle.

On lance la simulation dont le code est :

% étape de Simulation

y = sim(net,S);

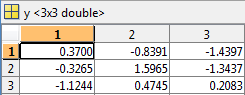
% fin simulation

Et on obtient la matrice de sortie Y :

Verdict pour la forme 1

Verdict pour la forme 3

Verdict pour la forme 2



Ligne1 : Etat de la forme triangulaire

Ligne2 : Etat de la forme rectangulaire

Ligne3 : Etat de la forme circulaire

On remarque que le réseau de neurones à bien identifier les 3 formes.

Pour la première forme on trouve que la forme est à 0.37 triangulaire, à -0.3265 rectangulaire et à -1.1244 circulaire, donc la valeur la plus grande est donnée pour la forme triangulaire ainsi le réseau nous indique que la forme est triangulaire, hors nous savions dès le début que la forme est triangulaire.

Pour la forme 2 le réseau nous indique que la forme est rectangulaire

(1.5965 > 0.4745 >-0.4391).

Pour la forme 3 le réseau nous indique que la forme est circulaire

(0.2083 > -1.3437 >--1.4397).

Donc le réseau de neurones a reconnu parfaitement les formes.

Enfin on donne le code tout entier :

% préparation données d'apprentissage

Num\_Inputs=10 ;% nombre de critères

P=zeros(Num\_Inputs,9);% input matrix

T=zeros(3,9);

for h=1:9 % 9 boucles pour 9 images

switch h

case 1

Img = imread('t1.bmp');

Target=[1;-1;-1] ;

case 2

Img = imread('t2.bmp');

Target=[1;-1;-1] ;

case 3

Img = imread('t3.bmp');

Target=[1;-1;-1] ;

case 4

Img = imread('r1.bmp');

Target=[-1;1;-1] ;

case 5

Img = imread('r2.bmp');

Target=[-1;1;-1];

case 6

Img = imread('r3.bmp');

Target=[-1;1;-1];

case 7

Img = imread('c1.bmp');

Target=[-1;-1;1];

case 8

Img = imread('c2.bmp');

Target=[-1;-1;1];

case 9

Img = imread('c3.bmp');

Target=[-1;-1;1];

end %fin switch

T(:,h)=Target ; % remplissage matrice T

[Num\_Row,Num\_column] = size(Img) ;% determiner la dimension de l'image

% remplissage matrices d'entrée

for i=1:Num\_Inputs

for j=(((Num\_Row/Num\_Inputs)\*(i-1))+1) : ((Num\_Row/Num\_Inputs)\*(i))

for k=1 : Num\_column

if Img(j,k)==0

P(i,h)=P(i,h)+k ;

end

end

end

end

end % fin boucle for

%fin préparation données d'apprentissage

% préparation données de simulation

S=zeros(Num\_Inputs,3);% matrice de simulation

for h=1:3 % 3 boucles pour 3 images de simulation

switch h

case 1

Img = imread('t4.bmp');

case 2

Img = imread('r4.bmp');

case 3

Img = imread('c4.bmp');

end %fin switch

[Num\_Row,Num\_column] = size(Img) ;

% initiation for input matrix

for i=1:Num\_Inputs

for j=(((Num\_Row/Num\_Inputs)\*(i-1))+1) : ((Num\_Row/Num\_Inputs)\*(i))

for k=1 : Num\_column

if Img(j,k)==0

S(i,h)=S(i,h)+k ;

end

end

end

end

end % fin boucle for

%fin préparation données de simulation

% Normalisation des Matrices

A=[P,S] ;

maxi=max(max(A));

mini=min(min(A));

[a,b]=size(A);

for i=1:a

for j=1:b

AN(i,j)=2\*(A(i,j)/(maxi-mini))-1;

end

end

P=AN(:,1:9);

S=AN(:,10:12) ;

%fin Normalisation des matrices

% étape de création de réseau de neurones et d'apprentissage

Num\_Neuron\_Hidden=20 ;%nombre de neurones de la couche cachée

net = newff(P,T,Num\_Neuron\_Hidden,{},'traingd');% Num\_Neuron\_Hidden=10

net=init(net); % reintialisation des poids et des bias

net.trainparam.epochs=250;% nombre maximal 'itération

net.trainparam.goal=0.0001; % erreur à respecter

net=train(net,P,T); % lancement de l'apprentissage

% fin apprentissage

% étape de Simulation

y = sim(net,S);

% fin simulation