Note : durant tout le TD, le coefficient d'apprentissage α sera égal à 0.1.

1 Mise en place d'un perceptron simple

– *Créer une fonction*

```
[y]=perceptron_simple(x,w,active);
```

Ce programme doit évaluer la sortie d'un perceptron simple (1 neurone) pour une entrée élément de R^2.
Les paramètres :
- La variable `w` contient les poids synaptiques du neurone. C'est un vecteur à 3 lignes. La première ligne correspond au seuil.
- La variable `x` contient l'entrée du réseau de neurones. C'est un vecteur à 2 lignes.
- La variable `active` indique la fonction d'activation utilisée. Si `active==0` $\varphi(x) = \text{sign}(x)$ si `active==1` $\varphi(x) = \tanh(x)$ si `active==2` $\varphi(x) = \frac{1}{1+e^{-x}}$

Le résultat :
- La variable `y` est un scalaire correspondant à la sortie du neurone.

– *Tester votre perceptron avec l'exemple du OU logique vu en cours (en utilisant* $\varphi(x) = \text{sign}(x)$*).*
– *Afficher dans le cadre de l'exemple du OU logique sur la même figure les différents éléments de l'ensemble d'apprentissage et la droite séparatrice associée aux poids du neurone sur la même figure.* Par exemple sous un langage comme Matlab/Octave, pour afficher une droite de type $y = ax + b$, par exemple pour $x \in [0, 1]$ on peut faire :

```
x=0:1; y=a*x+b; plot(x,y);
```

2 Etude de l'apprentissage

2.1 Programmation apprentissage simple

– *A partir de la fonction* `perceptron_simple`, *créer une fonction*

```
[w]=apprentissage_simple(x,yd);
```

Ce programme retourne le vecteur de poids w obtenu par apprentissage selon la règle d'apprentissage simple du perceptron.
Les paramètres :
- La variable `x` contient l'ensemble d'apprentissage. C'est une matrice à 2 lignes et n colonnes.
- La variable `yd(i)` indique la réponse désirée pour chaque élément `x(:,i)`. `yd` est un vecteur de 1 ligne et n colonnes de valeurs +1 ou -1 (classification à 2 classes).

Le résultat :
- La variable `w` contient les poids synaptiques du neurone après apprentissage. C'est un vecteur à 3 lignes. La première ligne correspond au seuil.

La droite séparatrice et les points d'apprentissage seront affichés à chaque itération (une itération correspond à la présentation de tous les individus de l'ensemble d'apprentissage), ainsi que le nombre de mal classé. L'algorithme s'arrêtera dès que tous les points auront été bien classés, ou au bout de 100 itérations. Les poids initiaux seront générés de manière aléatoire. La fonction d'activation est $\varphi(x) = \text{sign}(x)$.

2.2 Test 1 simple

– *Charger des données td4_d1.mat.*
Ce fichier comprend une variable :
- La matrice `x` qui contient l'ensemble d'apprentissage constitué de 2 classes de 25 individus chacune en dimension 2. Un "professeur" nous a indiqué que les 25 premiers individus sont issus de la classe 1 et les 25 derniers de la classe 2.

– *Appliquer votre algorithme d'apprentissage simple.*

2.3 Programmation apprentissage Widrow-hoff

– *A partir de la fonction* `perceptron_simple` *créer une fonction*

```
[w]=apprentissage_widrow(x,yd,active);
```

Ce programme retourne le vecteur de poids w obtenu par apprentissage selon la règle d'apprentissage utilisant la descente du gradient.

Les paramètres :

- La variable `x` contient l'ensemble d'apprentissage. C'est une matrice à 2 lignes et n colonnes.
- La variable `yd(i)` indique la réponse désirée pour chaque élément `x(:,i)`. `yd` est un vecteur de 1 ligne et n colonnes de valeurs +1 ou -1 (classification à 2 classes).
- La variable `active` indique la fonction d'activation utilisée. Si `active==1` $\varphi(x) = \tanh(x)$ si `active==2` $\varphi(x) = \frac{1}{1+e^{-x}}$

Le résultat :

- La variable `w` contient les poids synaptiques du neurone après apprentissage. C'est un vecteur à 3 lignes. La première ligne correspond au seuil.

La droite séparatrice et les points d'apprentissage seront affichés à chaque itération (une itération correspond à la présentation de tous les individus de l'ensemble d'apprentissage), ainsi que l'erreur de classification. L'algorithme s'arrêtera dès que l'erreur est nulle, ou au bout de 100 itérations. Les poids initiaux seront générés de manière aléatoire.

2.4 Test 1 simple (suite)

– *Appliquer votre algorithme d'apprentissage de Widrow-Hoff à partir des données du fichier td4_d1.*
– *Comparer la droite séparatrice obtenue avec le résultat précédent (issu de l'apprentissage simple).*

2.5 Test 2

– *Charger des données td4_d2.mat.*

Ce fichier comprend une variable :

- La matrice `x` qui contient l'ensemble d'apprentissage constitué de 2 classes de 25 individus chacune en dimension 2. Un "professeur" nous a indiqué que les 25 premiers individus sont issus de la classe 1 et les 25 derniers de la classe 2.

– *Appliquer votre algorithme d'apprentissage simple et votre algorithme d'apprentissage de Widrow-Hoff. Comparer et conclure.*

3 Perceptron multicouches

3.1 Mise en place d'un perceptron multicouche

– *Créer une fonction*

```
[y]=multiperceptron(x,w1,w2);
```

Ce programme calcul la sortie d'un perceptron multicouches à 1 neurone sur la couche de sortie et deux neurones sur la couche cachée pour une entrée élément de R^2. La fonction d'activation est $\varphi(x) = \frac{1}{1+e^{-x}}$.

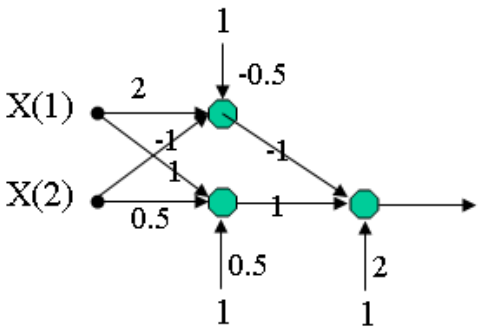
Les paramètres :

- La variable `w1` contient les poids synaptiques des 2 neurones de la couche cachée. C'est une matrice à 3 lignes et 2 colonnes. La première colonne `w1(:,1)` correspond au 1^{er} neurone de la couche cachée et `w1(:,2)` correspond au $2^{ème}$ neurone de la couche cachée.
- La variable `w2` contient les poids synaptiques du neurone de la couche de sortie. C'est un vecteur à 3 lignes.
- La variable `x` contient l'entrée du réseau de neurones. C'est un vecteur à 2 lignes.

Le résultat :

- La variable `y` est un scalaire correspondant à la sortie du neurone.

– *Tester votre perceptron multicouches avec l'exemple ci-dessous pour une entrée $x = [1\ 1]'$:*



Comparer le résultat "informatique" avec la sortie attendue calculée sur "le papier".

3.2 Programmation apprentissage multicouches

– *Créer une fonction*

```
[w1,w2]=multiperceptron_widrow(x,yd);
```

Ce programme retourne les poids $w1$ et $w2$ obtenus par apprentissage selon la règle d'apprentissage utilisant la descente du gradient. Nous étudions le cas d'un perceptron multicouches à 1 neurone sur la couche de sortie et deux neurones sur la couche cachée pour une entrée élément de R^2. La fonction d'activation est $\varphi(x) = \frac{1}{1+e^{-x}}$.

Les paramètres :

- La variable `x` contient l'ensemble d'apprentissage. C'est une matrice à 2 lignes et n colonnes.
- La variable `yd(i)` indique la réponse désirée pour chaque élément `x(:,i)`. `yd` est un vecteur de 1 ligne et n colonnes de valeurs $+1$ ou 0 (classification à 2 classes).

Le résultat :

- La variable `w1` contient les poids synaptiques des 2 neurones de la couche cachée. C'est une matrice à 3 lignes et 2 colonnes. La première colonne `w1(:,1)` correspond au 1^{er} neurone de la couche cachée et `w1(:,2)` correspond au $2^{ème}$ neurone.
- La variable `w2` contient les poids synaptiques du neurone de la couche de sortie. C'est un vecteur à 3 lignes.

L'erreur de classification sera affichée à chaque itération (une itération correspond à la présentation de tous les individus de l'ensemble d'apprentissage). L'algorithme s'arrêtera dès que l'erreur est nulle, ou au bout de 5000 itérations. Les poids initiaux seront générés de manière aléatoire. Nous utiliserons comme coefficient d'apprentissage $\alpha = 0.5$.

Vous allez tester votre algorithme d'apprentissage pour le cas du XOR. La table de cette fonction est

$x(1)$	0	1	0	1
$x(2)$	0	0	1	1
$ydesire$	0	1	1	0

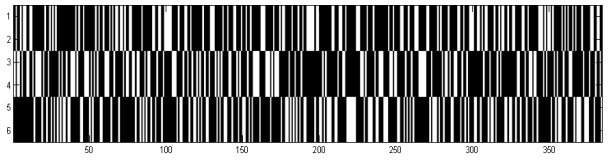
– *Créer l'ensemble d'apprentissage. Afficher cet ensemble avec la fonction affiche_classe. Pensez-vous que ce problème puisse être traité par un perceptron simple ?*
– *Appliquer votre algorithme d'apprentissage.*
– *Tester, à partir de votre fonction* ***`multiperceptron`****, le réseau de neurones ainsi obtenu sur l'ensemble d'apprentissage.*
– *Afficher les droites séparatrices associées aux différents neurones et les points de l'ensemble d'apprentissage.*

4 Lecteur de Code-barres

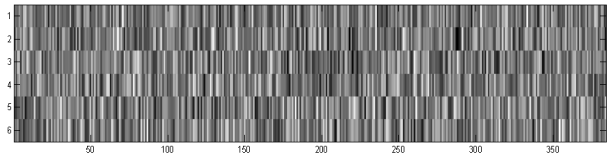
L'objectif de cette dernière partie est de mettre en place un réseau de neurones permettant de décoder une séquence code-barres. Un lecteur optique va lire successivement des colonnes composées de patch noir ou blanc et les numériser. Ces colonnes sont d'une largeur de 1 pixels et d'une hauteur de 6 pixels. Trois valeurs différentes.

Pour le symbole 1 correspond $\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$, pour le symbole 2 correspond $\begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$, pour le symbole 3 correspond $\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$

Toutefois, les codes barres sont dégradés et les valeurs numérisées ne sont pas exactement celles attendues. Nous illustrons ce problème sur la figure ci-dessous, où nous présentons le code-barres d'origine et celui finalement numérisé.



Code-Barres du produit



Code-Barres numérisé

– *Proposer et programmer un perceptron multicouches permettant de détecter la séquence associée à un code-barres.*

Nous proposons quelques éléments :

1. Le réseau aura donc 6 entrées, puisque chaque symbole est représenté par un vecteur de 6 valeurs.
2. Une structure en deux couches est suffisantes.

– *Appliquer votre perceptron pour détecter la séquence associée au code-barres contenu dans le fichier donnee.mat*

Le fichier contient la variable x correspondant au code-barres (de taille 6 lignes par 384 colonnes) et la variable oracle qui indique les 384 symboles qu'il faut trouver.