CY TECH	ING2-GI : GENIE INFORMATIQUE IA : RESEAUX DE NEURONES
Rédigé par : Astrid Jourdan & Yannick Le Nir	A l'intention de : Elèves ING2-GI
Durée : 6h00	

Exercice 1 : OR *vs* XOR

1) Considérons le problème du « OR » : les points (1,0), (1,1), (0,1) sont positifs et le point (0,0) est négatif. Soit le réseau,

$$\hat{y} = f(x_1 + x_2 - 0.5)$$

avec comme activation la fonction de Heaviside, $f(x) = 1$ si $x \geq 0$ et 0 sinon.

(a) Dessiner le réseau de neurones.

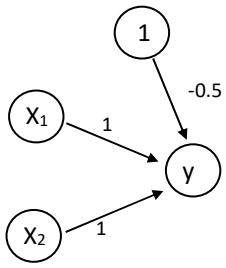
(b) Vérifier qu'il correspond bien au OR

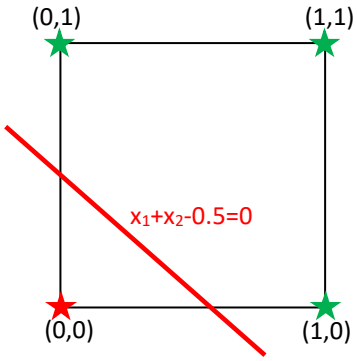
(c) Faire une représentation graphique des points de la base d'apprentissage et de la frontière obtenue.

(d) La solution est-elle unique ?

(e) Utiliser la fonction d'activation sigmoïde. Comparer les probabilités des différents points de la classe positive.

Correction





On a $\hat{y} = f(x_1 + x_2 - 0.5)$, d'où

X_1	X_2	$\hat{y}$
0	0	0+0-0.5=-0.5<0 $\Rightarrow$ 0
0	1	0+1-0.5=0.5>0 $\Rightarrow$ 1
1	0	0+1-0.5=0.5>0 $\Rightarrow$ 1
1	1	1+1-0.5=1.5>0 $\Rightarrow$ 1

La solution n'est pas unique car plusieurs droites sont possibles pour séparer les positifs des négatifs.

On peut utiliser une autre fonction d'activation car la sortie de réseau de neurones ne représente pas la classe mais la probabilité d'appartenir à la classe positive sachant x. Si cette probabilité est supérieure à 0,5, on prédit la classe positive et vice-versa. Par exemple avec la fonction sigmoïde, on a

$\hat{y} = f(0,0) = f(-0,5) = \frac{1}{1+e^{-(-0.5)}} = 0.38 = P(Y = 1|(x1 = 0, x2 = 0)) < 0.5 \Rightarrow \hat{y} = 0$

$\hat{y} = f(1,0) = f(0,5) = \frac{1}{1+e^{(-0.5)}} = 0.62 = P(Y = 1|(x1 = 1, x2 = 0)) > 0.5 \Rightarrow \hat{y} = 1$ (idem pour (0,1))

$\hat{y} = f(1,1) = f(1,5) = \frac{1}{1+e^{(-1.5)}} = 0.82 = P(Y = 1|(x1 = 1, x2 = 1)) > 0.5 \Rightarrow \hat{y} = 1$

On note que plus le point est éloigné de la frontière et plus la probabilité augmente.

2) Considérons le problème du « XOR » (ou exclusif) : les points (1,0) et (0,1) sont positifs et les points (0,0) et (1,1) sont négatifs.

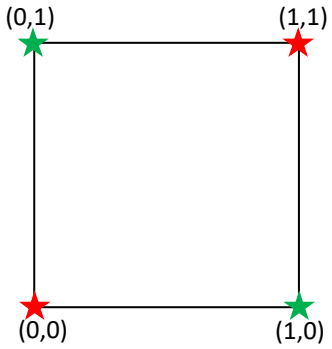
(a) Faire une représentation graphique des points de la base d'apprentissage. Pensez-vous qu'un réseau sans couche cachée soit suffisant ici ?

Considérons maintenant un réseau avec 2 neurones cachés. Notons (x_1,x_2) la couche d'entrée (x_3,x_4) la couche cachée et $\hat{y}=x_5$ la couche de sortie. On choisit les poids suivants : $w_{13}=1$, $w_{14}=-1$, $w_{23}=-1$, $w_{24}=1$, $w'_{35}=-1$, $w'_{45}=-1$, un biais de poids 1 entre la couche cachée et la couche de sortie, et toujours la fonction de Heaviside.

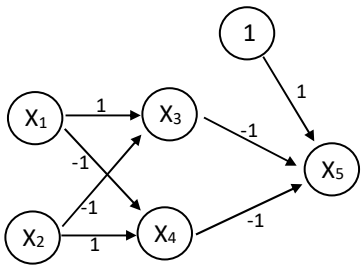
(b) Dessiner le réseau de neurone.

(c) Vérifier qu'il correspond bien au XOR.

Correction



Avec un seul neurone, on obtient une séparation linéaire (ou sigmoïde) or on constate qu'une simple frontière linéaire ne peut pas séparée les points.



On a $x_3=f(x_1-x_2)$, $x_4=f(-x_1+x_2)$ et $x_5=f(-x_3-x_4+1)$, d'où

X_1	X_2	X_3	X_4	X_5
0	0	1x0-1x0=0 $\Rightarrow$ 1	-1x0+1x0=0 $\Rightarrow$ 1	-1x1-1x1+1=-1 $\Rightarrow$ 0
0	1	1x0-1x1=-1 $\Rightarrow$ 0	-1x0+1x1=1 $\Rightarrow$ 1	-1x0-1x1+1=0 $\Rightarrow$ 1
1	0	1x1-1x0=1 $\Rightarrow$ 1	-1x1+1x0=-1 $\Rightarrow$ 0	-1x1-1x0+1=0 $\Rightarrow$ 1
1	1	1x1-1x1=0 $\Rightarrow$ 1	-1x1+1x1=0 $\Rightarrow$ 1	-1x1-1x1+1=-1 $\Rightarrow$ 0

Exercice 2

L'objectif est de comprendre l'algorithme de rétro-propagation du gradient sur un exemple simple en dimension 2. Il s'agit d'ajuster un réseau à un seul neurone,

$$\hat{y}=f(w_1x_1+w_2x_2+w_0)$$

avec comme activation la fonction de Heaviside, $f(x)=1$ si $x\geq 0$ et 0 sinon.

La base d'apprentissage est réduite aux deux points,

	x_1	x_2	y
a	1	2	1
b	2	1	0

1) Expliquez que le séparateur est une droite. Supposons que le biais est nul. Qu'est-ce que cela signifie ? Que va optimiser l'algorithme ?

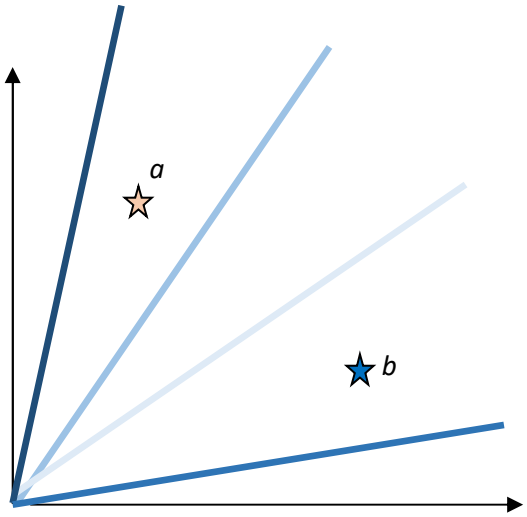
Correction

On remarque que

$$\hat{y}=1 \quad \Leftrightarrow \quad w_1x_1+w_2x_2+w_0\geq 0$$
$$\hat{y}=0 \quad \Leftrightarrow \quad w_1x_1+w_2x_2+w_0<0$$

Donc la droite d'équation $w_1x_1+w_2x_2+w_0=0$ sépare le plan en deux. Pour plus de simplicité, supposons que la constante w_0 (biais) est nulle, c'est-à-dire que la droite passe par l'origine.

Le coefficient directeur de la droite est donné par le rapport des poids,



$$w_1 x_1 + w_2 x_2 = 0 \quad \Leftrightarrow \quad x_2 = \boxed{-\frac{w_1}{w_2}} x_1$$

L'ajustement de la pente de la droite se fait de façon itérative grâce à l'algorithme de rétro-propagation.

2) Ecrire la fonction de mise à jour des poids pour un taux d'apprentissage α.

Correction

$$\Delta w_i = \alpha(y - \hat{y}) x_i \qquad \text{i=1,2}$$

3) On initialise les poids w_1=w_2=1. Quelle est l'initialisation de la pente de la droite ?

Correction

$$\delta = -\frac{w_1}{w_2} = -1$$

4) Exprimez la pente de la droite en fonction de α à l'issue de la 1ère itération. Quelle valeur de α faudrait-il poser pour que l'algorithme s'arrête à la première itération ?

Correction

- Pour le point *a* : $w_1 x_1 + w_2 x_2 = x_1 + x_2 = 3 \geq 0 \quad \Rightarrow \quad \hat{y} = 1$

  $\Rightarrow \quad \Delta w_1 = \alpha(1-1)x_1 = 0$

  $\Rightarrow \quad \Delta w_2 = \alpha(1-1)x_2 = 0$

  Donc pas de changement de poids.

- Pour le point *b* : $w_1 x_1 + w_2 x_2 = x_1 + x_2 = 3 \geq 0 \quad \Rightarrow \quad \hat{y} = 1$

  $\Rightarrow \quad \Delta w_1 = \alpha(0-1)x_1 = -2\alpha \qquad \Rightarrow \quad w_1 = 1 - 2\alpha$

  $\Rightarrow \quad \Delta w_2 = \alpha(0-1)x_2 = -\alpha \qquad \Rightarrow \quad w_2 = 1 - \alpha$

$$\Rightarrow \quad \delta = -\frac{w_1}{w_2} = -\frac{1-2\alpha}{1-\alpha}$$

Si on veut que l'algorithme classe bien les deux exemples dès la 1ère itération, il faut que la droite passe entre *a* et *b*, par exemple la 1ère bissectrice, *i.e.* δ=1.

$$\delta = 1 = -\frac{1-2\alpha}{1-\alpha} \quad \Rightarrow \quad \alpha = \frac{2}{3}$$

5) On pose α=0,2. Déroulez l'algorithme jusqu'à la quatrième itération. Tracez les droites séparatrices au fur et à mesure de l'algorithme. Que se passe-t-il à la quatrième itération ?

Correction

1ère itération : $w_1 = 1 - 2\alpha = 1 - 2 * 0{,}2 = 0{,}6$ et $w_2 = 1 - \alpha = 1 - 0{,}2 = 0{,}8$ et $\delta = -\frac{0{,}6}{0{,}8} = -0{,}75$

2ème itération :

- Pour le point *a* : $w_1 x_1 + w_2 x_2 = 0{,}6 x_1 + 0{,}8 x_2 = 2{,}2 \geq 0 \quad \Rightarrow \quad \hat{y} = 1$

  $\Rightarrow \quad \Delta w_1 = \alpha(1-1)x_1 = 0$

  $\Rightarrow \quad \Delta w_2 = \alpha(1-1)x_2 = 0$

  Donc pas de changement de poids.

- Pour le point *b* : $w_1 x_1 + w_2 x_2 = 0{,}6 x_1 + 0{,}8 x_2 = 2 \geq 0 \quad \Rightarrow \quad \hat{y} = 1$

  $\Rightarrow \quad \Delta w_1 = \alpha(0-1)x_1 = -2\alpha \qquad \Rightarrow \quad w_1 = 0{,}6 - 2\alpha = 0{,}2$

  $\Rightarrow \quad \Delta w_2 = \alpha(0-1)x_2 = -\alpha \qquad \Rightarrow \quad w_2 = 0{,}8 - \alpha = 0{,}6$

$\Rightarrow \quad \delta = -\frac{w_1}{w_2} = -0{,}33$

a et b sont toujours du même côté de la droite

3ème itération :

- Pour le point *a* : $w_1x_1 + w_2x_2 = 0{,}2x_1 + 0{,}6x_2 = 1{,}4 \geq 0 \quad \Rightarrow \quad \hat{y} = 1$

  $\Rightarrow \quad \Delta w_1 = \alpha(1-1)x_1 = 0$

  $\Rightarrow \quad \Delta w_2 = \alpha(1-1)x_2 = 0$

  Donc pas de changement de poids.

- Pour le point *b* : $w_1x_1 + w_2x_2 = 0{,}2x_1 + 0{,}6x_2 = 1 \geq 0 \quad \Rightarrow \quad \hat{y} = 1$

  $\Rightarrow \quad \Delta w_1 = \alpha(0-1)x_1 = -2\alpha \qquad \Rightarrow \quad w_1 = 0{,}2 - 2\alpha = -0{,}2$

  $\Rightarrow \quad \Delta w_2 = \alpha(0-1)x_2 = -\alpha \qquad \Rightarrow \quad w_2 = 0{,}6 - \alpha = 0{,}4$

$\Rightarrow \quad \delta = -\frac{w_1}{w_2} = 0{,}5$

a et b sont toujours du même côté de la droite

4ème itération :

- Pour le point *a* : $w_1x_1 + w_2x_2 = -0{,}2x_1 + 0{,}4x_2 = 0{,}2 \geq 0 \quad \Rightarrow \quad \hat{y} = 1$

  $\Rightarrow \quad \Delta w_1 = \alpha(1-1)x_1 = 0$

  $\Rightarrow \quad \Delta w_2 = \alpha(1-1)x_2 = 0$

  Donc pas de changement de poids.

- Pour le point *b* : $w_1x_1 + w_2x_2 = -0{,}2x_1 + 0{,}4x_2 = 0 \geq 0 \quad \Rightarrow \quad \hat{y} = 1$

  $\Rightarrow \quad \Delta w_1 = \alpha(0-1)x_1 = -2\alpha \qquad \Rightarrow \quad w_1 = -0{,}2 - 2\alpha = -0{,}6$

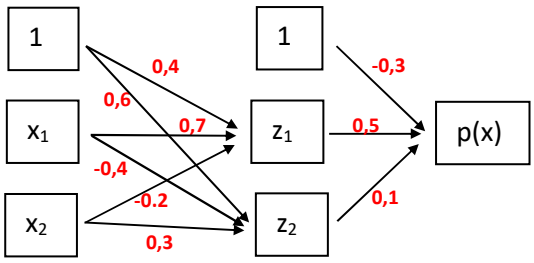
  $\Rightarrow \quad \Delta w_2 = \alpha(0-1)x_2 = -\alpha \qquad \Rightarrow \quad w_2 = 0{,}4 - \alpha = 0{,}2$

$\Rightarrow \quad \delta = -\frac{w_1}{w_2} = 3$

La droite a basculé de l'autre côté des points. Maintenant b est bien classé et a est mal classé. Le taux d'apprentissage est trop grand et engendre un phénomène d'oscillation.

Exercice 3

Soit le réseau de neurones suivant pour modéliser une sortie binaire, p($\boldsymbol{x}$)=P(Y=1|$\boldsymbol{x}$), où $\boldsymbol{x}$=(x_1,x_2)



1) On suppose une fonction d'activation relu sur la couche cachée. Explicitez les deux neurones de la couche cachée

Correction

$$\Sigma_1 = 0{,}4 + 0{,}6x_1 - 0{,}2x_2 \quad \Rightarrow \quad Z_1 = f(\Sigma_1) = max(0, \Sigma_1)$$

$$\Sigma_2 = 0{,}6 - 0{,}4x_1 + 0{,}3x_2 \quad \Rightarrow \quad Z_2 = f(\Sigma_2) = max(0, \Sigma_2)$$

2) Quelle est la fonction d'activation sur la couche de sortie ? Explicitez p($\boldsymbol{x}$) en fonction des deux neurones

Correction

La couche de sortie retourne la probabilité pour que Y=1 sachant x, la seule fonction d'activation qui donne une probabilité est la fonction sigmoide (logit inverse)

$$T = -0,3 + 0,5Z_1 + 0,1Z_2 \quad \Rightarrow \quad p(\boldsymbol{x}) = g(T) = \frac{1}{1+e^{-T}}$$

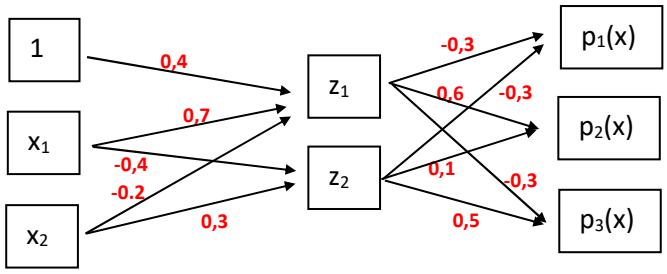
3) On considère l'exemple $\boldsymbol{x}$=(0,1). Calculez la sortie du réseau.

Correction

$$\Sigma_1 = 0,2 \quad \Rightarrow \quad Z_1 = max(0,0.2) = 0,2$$
$$\Sigma_2 = 0,9 \quad \Rightarrow \quad Z_2 = max(0,0.9) = 0,9$$
$$T = -0,3 + 0,2 \times 0,2 + 0,1 \times 0,9 = -1.06 \quad \Rightarrow \quad p(x) = g(T) = \frac{1}{1+e^{+1.06}} = 0,26$$

Exercice 3

Soit le réseau de neurones suivant pour modéliser une sortie ayant trois classes, Y∈{1,2,3}. On note, $p_k(\boldsymbol{x})=P(Y=k|\boldsymbol{x})$, où $\boldsymbol{x}=(x_1,x_2)$ et k=1,2,3.



1) On suppose une fonction d'activation tangente hyperbolique sur la couche cachée. Explicitez les deux neurones de la couche cachée en fonction de x_1, x_2 et le biais.

Correction

$$\Sigma_1 = 0,4 + 0,7x_1 - 0,2x_2 \quad \Rightarrow \quad Z_1 = f(\Sigma_1) = \tanh(\Sigma_1)$$
$$\Sigma_2 = -0,4x_1 + 0,3x_2 \Rightarrow \quad Z_2 = f(\Sigma_2) = \tanh(\Sigma_2)$$

2) Quelle est la fonction d'activation de la couche de sortie ? Explicitez $p_k(\boldsymbol{x})$ en fonction des deux neurones de la couche cachée.

Correction

Etant donné qu'il y a plus de deux classes en sortie, il faut utiliser la fonction softmax pour s'assurer que la somme des probabilités en sortie fasse 1.

$$T_1 = -0,3Z_1 - 0,3Z_2 \quad \Rightarrow \quad p_1(x) = g(T_1) = \frac{e^{T_1}}{e^{T_1}+e^{T_2}+e^{T_3}}$$

$$T_2 = 0,6Z_1 + 0,1Z_2 \quad \Rightarrow \quad p_2(x) = g(T_2) = \frac{e^{T_2}}{e^{T_1}+e^{T_2}+e^{T_3}}$$

$$T_3 = -0,3Z_1 + 0,5Z_2 \quad \Rightarrow \quad p_3(x) = g(T_3) = \frac{e^{T_3}}{e^{T_1}+e^{T_2}+e^{T_3}}$$

3) On considère l'exemple $\boldsymbol{x}$=(0,1). Calculez la sortie du réseau.

Correction

$$\Sigma_1 = 0,4 + 0,7x0 - 0,2x1 = 0,2 \Rightarrow \quad Z_1 = \tanh(0,2) = 0,197$$
$$\Sigma_2 = -0,4x0 + 0,3x1 = 0,3 \Rightarrow \quad Z_2 = \tanh(0,3) = 0,291$$

$$T_1 = -0,3x0,197 - 0,3x0,291 = -0,147 \quad \Rightarrow \quad p_1(x) = \frac{e^{-0,147}}{e^{-0,147} + e^{0,218} + e^{0,086}} = 0,27$$

$$T_2 = 0,6x0,197 + 0,1x0,291 = 0,218 \quad \Rightarrow \quad p_2(x) = \frac{e^{0,218}}{e^{-0,147} + e^{0,218} + e^{0,086}} = 0,39$$

$$T_3 = -0,3x0,197 + 0,5x0,291 \quad \Rightarrow \quad p_3(x) = \frac{e^{0,086}}{e^{-0,147} + e^{0,218} + e^{0,086}} = 0,34$$

La classe la plus probable est donc la 2.

NB. On note que la somme des probabilités = 1

Exercice 4

Dans cet exercice il s'agit de mettre en œuvre un réseau de neurones avec la fonction

`NN=nnet(Y~X1+X2+…,data=datatrain,size=2,decay=0.1,…)`

du package `nnet` de R. L'affichage du réseau de neurones se fait à l'aide de la fonction

`plotnet(NN)`

du package `NeuralNetTools`.

L'optimisation des hyperparamètres du réseau se fait avec la fonction

`tune.nnet(Y~X1+X2+…,data=datatrain,size=2:10,decay=c(0,0.1,1,2,3),maxit=100…)`

du package `e1071`

1) A partir de l'aide de la fonction `nnet`, répondez aux questions suivantes.
   a) Comment sont initialisés les poids par défaut. Qu'est-ce que cela implique pour les variables d'entrée ? Préparez le jeu de données « iris » en conséquence.

`rang`	Initial random weights on [-`rang`, `rang`]. Value about 0.5 unless the inputs are large, in which case it should be chosen so that `rang` * max(|x|) is about 1.

L'initialisation des poids se fait de façon aléatoire entre [-0.5,0.5] par défaut quelles que soient les valeurs prises par la variable x. Afin d'éviter que des variables d'entrée prenant de grandes valeurs aient un rôle trop prédominant dans la combinaison linéaire de la couche cachée, on normalise les variables de façon à ramener leur plage de variation entre [0,1] :

$$\frac{x_i - \min}{\max - \min}$$

```
data(iris)
Mydata=iris

# Find max and min of each column (except the target)
help(apply)
max = apply(Mydata[,1:4] , 2 , max)
print(max)
min = apply(Mydata[,1:4], 2 , min)
print(min)

# Scale the data
data.scaled = scale(Mydata [,1:4], center = min, scale = max-min)
print(apply(data.scaled,2,range)) # the range of variation for each column is [0,1]
class(data.scaled) # data.scaled is a matrix and not a data.frame
data.scaled=as.data.frame(data.scaled)

# Add the target
data.scaled=cbind(data.scaled,Mydata$Species)
```

```
print(names(data.scaled)) # names are not ok
names(data.scaled)=names(Mydata)
View(data.scaled)
```

b) Construisez un réseau avec 5 neurones pour les données « iris » (on utilisera toute la base pour apprendre le réseau). Combien y-a-t-il de poids à ajuster dans ce réseau (combien sur la première couche et combien sur la deuxième couche)? A quoi correspondent les informations affichées ? Est-ce que l'algorithme de rétro-propagation a convergé ? Quel autre indicateur peut-on récupérer pour savoir si l'algorithme a convergé ?

```
NN=nnet(Species~.,data=data.scaled,size=5)
# weights:  43
initial  value 180.530689
iter  10 value 65.807375
iter  20 value 6.517717
iter  30 value 5.646148
iter  40 value 5.128868
iter  50 value 3.417229
iter  60 value 2.782739
iter  70 value 2.707157
iter  80 value 2.684976
iter  90 value 2.259868
iter 100 value 2.251662
final  value 2.251662
stopped after 100 iterations
```

weights: 43 : Il y a 43 poids à ajuster (4 variables + constante)* (5 neurones)=25 entre la couche d'entrée et la couche cachée et (5 neurones + constante)*(3 classes)=18 entre la couche cachée et la couche de sortie.

Ensuite il y a l'évolution de la fonction de coût au cours des itérations (ici de 180.53 à 2.25)
L'algorithme n'a pas convergé car il stoppe après 100 itérations.
On peut aussi utiliser l'attribut convergence :

`convergence` `1` if the maximum number of iterations was reached, otherwise `0`.

```
NN$convergence
[1] 1
```

c) Quels sont les critères d'arrêt possibles pour stopper l'algorithme de rétro-propagation ? Faites varier ces critères de façon à ce que l'algorithme converge.

L'algorithme s'arrête après 100 itérations. C'est la valeur fixée par défaut. Si on la fixe à 500 par exemple, l'algorithme converge. En répétant plusieurs fois l'algorithme, on note que certaines fois, il converge vers un optimum local car l'erreur finale reste assez grande.

```
NN=nnet(Species~.,data=data.scaled,size=5,maxit=500)
```

```
# weights:  43
initial  value 163.954829
iter  10 value 7.014039
iter  20 value 5.999940
iter  30 value 5.843863
iter  40 value 5.139650
iter  50 value 4.596485
iter  60 value 4.314287
iter  70 value 3.538995
iter  80 value 0.477191
iter  90 value 0.008344
iter 100 value 0.001919
iter 110 value 0.000432
iter 120 value 0.000396
iter 130 value 0.000378
iter 140 value 0.000198
iter 150 value 0.000190
iter 160 value 0.000146
iter 170 value 0.000142
final  value 0.000091
converged
```

```
# weights:  43
initial  value 192.607179
iter  10 value 16.069879
iter  20 value 6.019744
iter  30 value 5.928174
iter  40 value 5.749375
iter  50 value 5.466930
iter  60 value 5.071446
iter  70 value 5.022896
iter  80 value 5.012093
iter  90 value 5.011265
iter 100 value 5.007869
iter 110 value 4.999005
iter 120 value 4.989568
iter 130 value 4.978394
iter 140 value 4.975419
iter 150 value 4.975361
iter 160 value 4.971277
iter 170 value 4.968670
iter 180 value 4.967285
iter 190 value 4.966859
iter 200 value 4.966761
final  value 4.966611 minimum local
```

```
converged
```

L'algorithme considère qu'il y a convergence quand l'erreur descend en dessous d'un certain seuil. On peut augmenter ce seuil, par exemple le fixer à 10.

```
NN=nnet(Species~.,data=data.scaled,size=5,maxit=100,abstol=10)
# weights:  43
initial  value 211.150076
iter  10 value 14.039224
iter  10 value 9.241374
iter  10 value 9.241374
final  value 9.241374
converged
```

d) Quelle est la fonction d'activation de la couche de sortie ? de la couche cachée ? Quelle est la fonction de coût ?

`linout`	switch for linear output units. Default logistic output units.
`entropy`	switch for entropy (= maximum conditional likelihood) fitting. Default by least-squares.
`softmax`	switch for softmax (log-linear model) and maximum conditional likelihood fitting. `linout`, `entropy`, `softmax` and `censored` are mutually exclusive.

S'il y a une seule sortie (régression ou classification binaire) alors la fonction d'activation de la couche de sortie est par défaut la fonction logistique.
S'il y a plusieurs sorties (classification à k>2 classes) alors il bascule tout seul sur la fonction softmax. Pas besoin de la passer en argument.

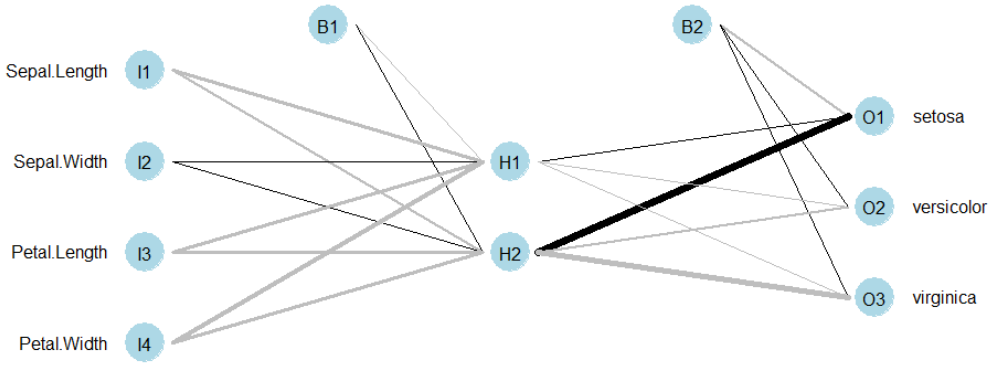
```
NN$softmax
[1] TRUE
```

La fonction de coût est par défaut l'erreur quadratique moyenne.

```
NN$entropy
[1] FALSE
```

2) Construisez un réseau avec deux neurones et utilisez la fonction `plotnet` pour afficher le réseau de neurones. Déterminez le poids de chaque branche.

```
summary(NN)
a 4-2-3 network with 19 weights
options were - softmax modelling
  b->h1  i1->h1  i2->h1  i3->h1  i4->h1
-317.26    1.42 -186.25   55.72   72.57
  b->h2  i1->h2  i2->h2  i3->h2  i4->h2
   5.43   -0.90    2.83   -4.80   -6.62
  b->o1  h1->o1  h2->o1
-113.23 -138.84  266.68
  b->o2  h1->o2  h2->o2
  29.89   35.39   95.74
  b->o3  h1->o3  h2->o3
  81.78  102.93 -363.15
```



3) Etudiez les valeurs de la fonction coût en fonction des valeurs du paramètre de régularisation decay∈ [0,0.1,0.2,0.5].

decay=0	decay=0.1	decay=0.2	decay=0.5
# weights: 43 initial value 186.142254 iter 10 value 10.375282 iter 20 value 5.921430 iter 30 value 5.598421 iter 40 value 5.284298 iter 50 value 5.160997 iter 60 value 4.521340 iter 70 value 1.318902 iter 80 value 0.008205 iter 90 value 0.000385 **final value 0.000058** converged	# weights: 43 initial value 170.577866 iter 10 value 67.482703 iter 20 value 41.974713 iter 30 value 39.364782 iter 40 value 39.219056 iter 50 value 39.210198 iter 60 value 39.198179 iter 70 value 39.139230 iter 80 value 39.025984 iter 90 value 38.968117 iter 100 value 38.963380 **final value 38.963105** converged	# weights: 43 initial value 173.557554 iter 10 value 82.556971 iter 20 value 61.476215 iter 30 value 58.516273 iter 40 value 57.607798 iter 50 value 57.469438 iter 60 value 57.402326 iter 70 value 57.359406 iter 80 value 57.326686 iter 90 value 57.297556 iter 100 value 57.293756 **final value 57.293664** converged	# weights: 43 initial value 181.270182 iter 10 value 100.869840 iter 20 value 88.646150 iter 30 value 88.387765 iter 40 value 88.386828 **final value 88.386825** converged

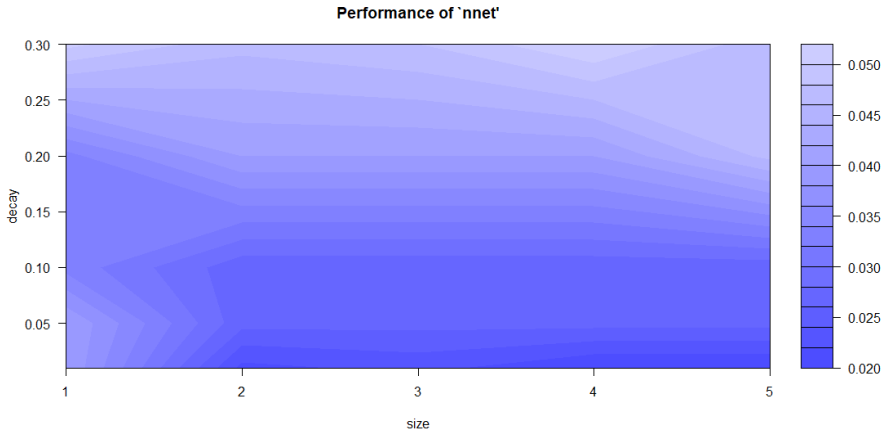
4) Utiliser la fonction `tune.nnet` pour trouver les hyperparametres optimaux.

```
opt=tune.nnet(Species~.,data=data.scaled,size=1:5,decay=c(0.01,0.05,0.1,0.2,0.3),maxit=100)
attributes(opt)
$names
[1] "best.parameters"  "best.performance" "method"
[4] "nparcomb"         "train.ind"        "sampling"
[7] "performances"     "best.model"

$class
[1] "tune"
```

On peut récupérer les paramètres optimaux, le meilleur score, la méthode de validation croisée,…

```
opt$best.parameters
  size decay
4    4  0.01
plot(opt)
```




```
opt$sampling
```

```
[1] "10-fold cross validation"
```

On peut modifier la méthode de cross-validation avec la fonction `tune.control`

5) Utilisez la fonction `predict` pour prédire la probabilité des chacune des classes des exemples de la base d'apprentissage. Calculez la matrice de confusion.

```
predict(NN,data.scaled[,1:4],type="raw")
          setosa   versicolor    virginica
1   1.000000e+00 1.441660e-21 2.500664e-56
2   1.000000e+00 3.536638e-20 8.766913e-54
3   1.000000e+00 7.478261e-21 5.095763e-55
4   1.000000e+00 9.808499e-20 5.676299e-53
```

…

```
prev=predict(NN,data.scaled[,1:4],type="class")
table(data.scaled$Species,prev)
            prev
             setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         49         1
  virginica       0          1        49
```

Exercice 4

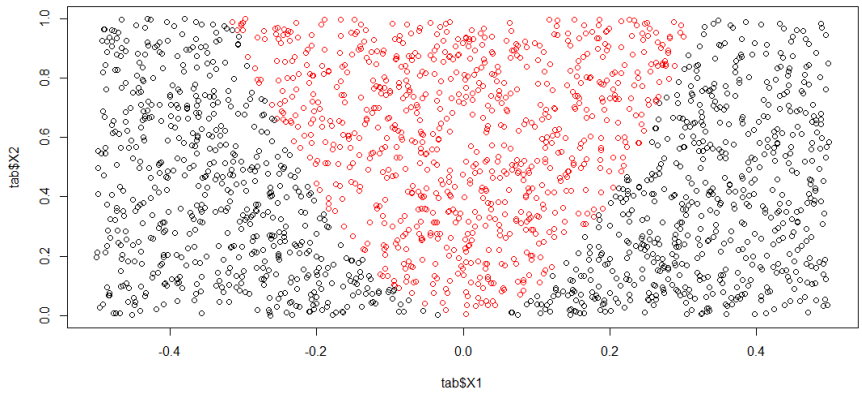
L'objectif de cet exercice est d'illustrer les frontières de séparation créées par le réseau de neurones.

1) Simulez un jeu de données de 2000 exemples tel que
   X1~U[-0.5,0.5]
   X2~U[0,1]
   Y=1 si $0.1X2>X1^2$ et Y=0 sinon
   Faites une représentation graphique du nuage de points avec les classes. Est-ce qu'il est linéairement séparable ?
2) Ajustez un réseau avec deux neurones et avec 1000 itérations maxi.
3) La frontière séparatrice pour un neurone Σ est la droite d'équation : $w_0+w_1x_1+w_2x_2=0$. Sur le nuage de points, ajoutez les droites engendrées par les deux neurones.
4) On considère le nouveau repère défini par les deux neurones

$$Z_1 = \frac{1}{1+e^{-\Sigma_1}} \quad \text{et} \quad Z_2 = \frac{1}{1+e^{-\Sigma_2}}$$

   Tracez le nuage de points dans ce nouveau repère. Est-il linéairement séparable ?
5) Dans ce nouveau repère, la frontière est définie par la droite d'équation : $w'_0+w'_1z_1+w'_2z_2=0$. Ajoutez cette droite sur le nouveau nuage de points.

```
tab=matrix(0,2000,3)
tab[,1]=runif(2000,min=-0.5,max=0.5) # X1=random entre [-0.5,0.5]
tab[,2]=runif(2000,min=0,max=1) # X2=random entre [0,1]
tab[,3]=(0.1*tab[,2]>(tab[,1]^2)) # Y=1 si 0.1*X2>X1²
tab=as.data.frame(tab)
names(tab)=c("X1","X2","Y")
plot(tab$X1,tab$X2,col=tab$Y+1)
```

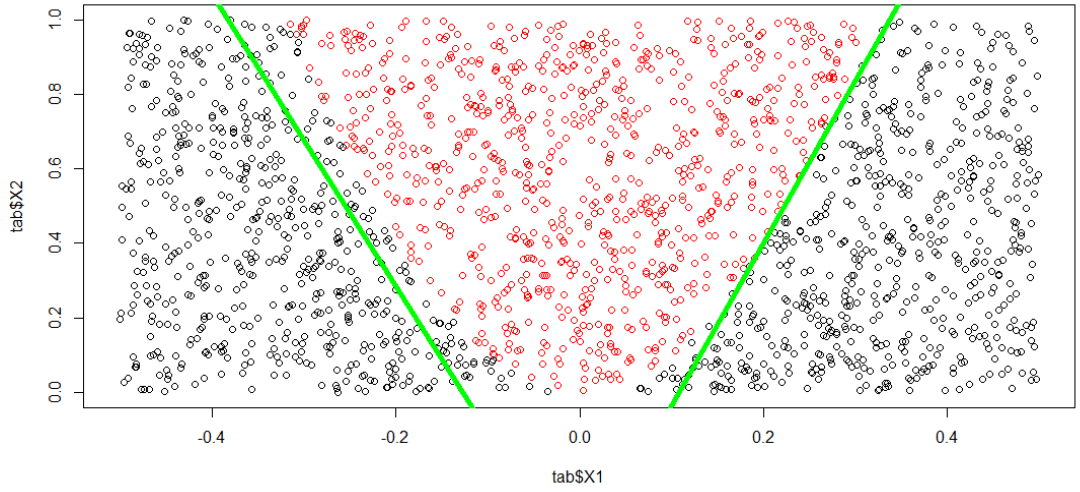


Une droite ne suffira pas à séparer les deux classes

```
NN=nnet(Y~.,data=tab,size=2,maxit=1000)
summary(NN)


poids.H1=NN$wts[1:3]
poids.H2=NN$wts[4:6]


abline(-poids.H1[1]/poids.H1[3],-poids.H1[2]/poids.H1[3],lwd=5,col="green")
abline(-poids.H2[1]/poids.H2[3],-poids.H2[2]/poids.H2[3],lwd=5,col=" green")
```



Les frontières étant légèrement quadratiques, les droites ne séparent pas parfaitement les classes mais elles se rapprochent des frontières.

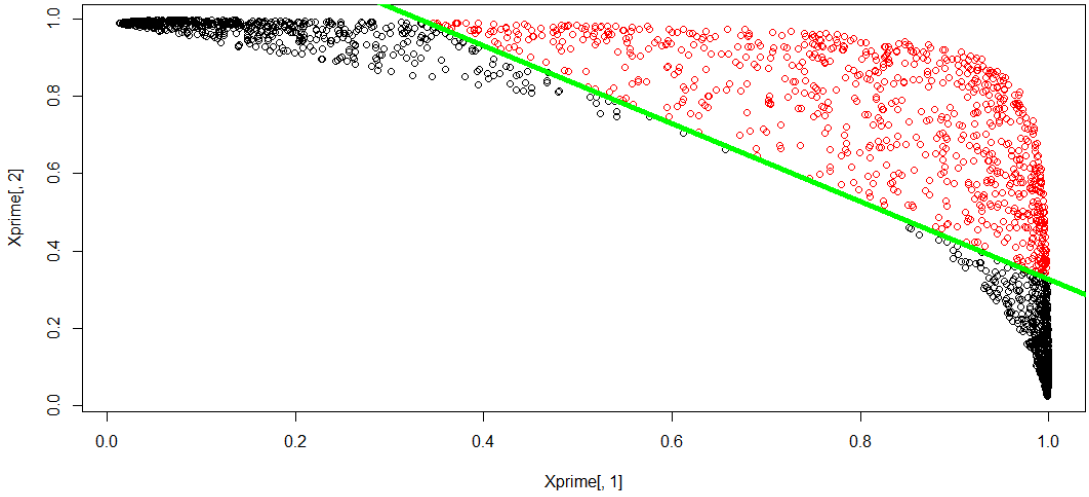
```
Xprime=tab[,1:2] # nouvelles coordonnées

for (i in 1:nrow(tab))
{
  x1=tab[i,1]
  x2=tab[i,2]
  sigma1=poids.H1[1]+poids.H1[2]*x1+poids.H1[3]*x2
  Xprime[i,1]=1/(1+exp(-sigma1))
  sigma2=poids.H2[1]+poids.H2[2]*x1+poids.H2[3]*x2
  Xprime[i,2]=1/(1+exp(-sigma2))

}

plot(Xprime[,1],Xprime[,2],col=tab$Y+1)
```

```
poids.sortie=NN$wts[7:9]
abline(-poids.sortie[1]/poids.sortie[3],-poids.sortie[2]/poids.sortie[3],lwd=5,col="green")
```



La couche cachée « transforme » le problème initial en un problème plus facilement séparable.

Exercice supplémentaire

Dans l'algorithme de rétro-propagation, déterminez les δ_k et δ_j lorsque la fonction coût est l'entropie croisée.

Correction

Pour un exemple (x,y), la fonction coût est de la forme

$$E(w) = \sum_{k=1}^{K} y_k \ln(p_k(x)) = \sum_{k=1}^{K} y_k \ln(g(T_k)) = \sum_{k=1}^{K} E_k(w)$$

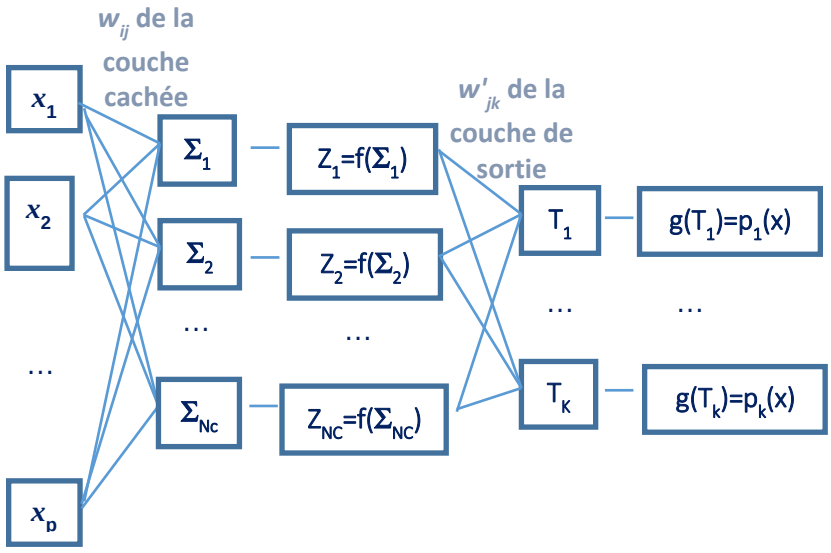
avec $E_k(w) = y_k \ln(g(T_k))$, où $T_k = \sum_{j=1}^{N_c} w'_{jk} Z_j$ et $Z_j = f(\Sigma_j)$ et $\Sigma_j = \sum_{i=1}^{p} w_{ij} x_i$.

- Poids de la deuxième couche :

w'_{jk} entre le neurone **j** et la sortie de la classe **k**

- Poids de la première couche :

w_{ij} entre la variable **i** et le neurone **j**



Dérivée par rapport aux poids de la deuxième couche

$$\frac{\partial E(w)}{\partial w'_{jk}} = \frac{\partial E(w)}{\partial T_k} \times \frac{\partial T_k}{\partial w'_{jk}} = \frac{\partial E(w)}{\partial T_k} Z_j$$

car nous avons montré en cours que $\frac{\partial T_k}{\partial w'_{jk}} = Z_j$. Donc

$$\delta_k = \frac{\partial E(\boldsymbol{w})}{\partial T_k} = \frac{\partial}{\partial T_k}\sum_{k=1}^{K} E_k(\boldsymbol{w}) = \frac{\partial}{\partial T_k} E_k(\boldsymbol{w}) = \frac{\partial}{\partial T_k} y_k \ln\left(g(T_k)\right) = y_k \frac{g'(T_k)}{g(T_k)}.$$

Si g est la fonction logit^{-1} alors g'(x)=g(x)(1-g(x)), d'où

$$\delta_k = \frac{\partial E(\boldsymbol{w})}{\partial T_k} = y_k \frac{g(T_k)\left(1-g(T_k)\right)}{g(T_k)} = y_k\left(1-g(T_k)\right).$$

<u>Dérivée par rapport aux poids de la première couche</u>

$$\frac{\partial E(\boldsymbol{w})}{\partial w_{ij}} = \sum_{k=1}^{K} \frac{\partial E_k(\boldsymbol{w})}{\partial w_{ij}}$$

On décompose

$$\frac{\partial E_k(\boldsymbol{w})}{\partial w_{ij}} = \frac{\partial E_k(\boldsymbol{w})}{\partial T_k} \times \frac{\partial T_k}{\partial Z_j} \times \frac{\partial Z_j}{\partial \Sigma_j} \times \frac{\partial \Sigma_j}{\partial w_{ij}}$$

On a vu en cours que : $\frac{\partial E_k(\boldsymbol{w})}{\partial T_k} = \delta_k$, $\frac{\partial T_k}{\partial Z_j} = w'_{jk}$, $\frac{\partial \Sigma_j}{\partial w_{ij}} = x_i$ et $\frac{\partial Z_j}{\partial \Sigma_j} = f'(\Sigma_j)$, donc

$$\frac{\partial E(\boldsymbol{w})}{\partial w_{ij}} = \delta_j x_i \quad \text{où} \quad \delta_j = \left(\sum_{k=1}^{K} \delta_k w'_{jk}\right) f'(\Sigma_j)$$

En fait, la démonstration pour la première couche est indépendante de la fonction coût.