Réseaux de neurones

Bruno Bouzy

7 mars 2017

Introduction

Ce document est le chapitre « réseau de neurones » du cours d'apprentissage automatique donné en Master Informatique. Dans la première partie, il montre l'exemple d'un réseau avec 2 neurones résolvant le problème du XOR avec l'algorithme de back-propagation (Rumelhart & al 1986). Cet exemple permet d'introduire cet algorithme et d'en montrer les avantages et limites. Ensuite, ce chapitre présente les fonctions discriminantes linéaires, le perceptron ou réseau de neurones à une couche, le MPL, (Multi-Layer Perceptron) ou réseau de neurones multi-couches.

« Backprop » sur l'exemple du XOR

Cet exemple montre le fonctionnement d'un petit réseau de neurones apprenant avec l'algorithme « backprop » (Rumelhart & al 1986). L'exemple est repris de (Tveter, chapitre 2). Le réseau est très simple. Son but est de donner la valeur de XOR(x, y) en fonction de x et y. La fonction z = XOR(x, y) est définie par la table 1.

x	0	0	1	1
y	0	1	0	1
z	0	1	1	0

Table 1 : la fonction XOR.

Calcul en avant :

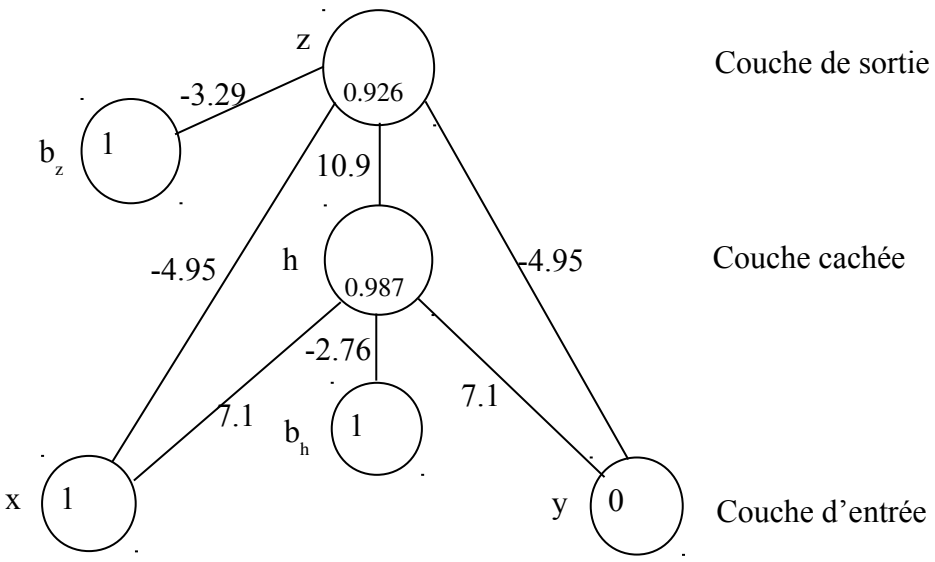

Figure 1 : Un réseau à 3 couches pour résoudre le problème du XOR.

La figure 1 montre un réseau avec 2 neurones permettant de calculer z, le XOR de x et y, deux variables données en entrée du réseau. Les ronds représentent des « unités ». Il y a 2 unités d'entrée : x et y. Il y a une unité cachée : h. z est une unité de sortie. b_h et b_z sont des « unités de biais ». Les nombres dans les ronds indique les valeurs d'activation des unités. Les lignes reliant les ronds sont les connexions entre unités. Chaque connexion a un « poids » indiqué à côté de la ligne correspondante. Un « neurone » est une unité ayant des connexions entrantes. Ici, il y a 2 neurones : h et z.

La plupart du temps, les réseaux sont organisés en couches. Les neurones d'une couche ont des connexions venant uniquement de la couche située en dessous. Ici, ce n'est pas le cas.

Pour calculer z, supposons que x = 1.0 et y = 0.0. D'après le table 1, on doit trouver 1.

Pour calculer la valeur d'activation d'une unité x en fonction de ses connexions entrantes, on commence par effectuer la somme, net_x, sur toutes les connexions entrantes dans l'unité x, des produits de l'unité associée à la connexion et du poids de la connexion. On a donc :

$$net_h = 1x7.1 + 1x(-2.76) + 0x7.1 = 4.34.$$

Dans les réseaux « linéaires », cette somme est égale à la valeur d'activation. Dans notre exemple, le réseau est « non linéaire », il utilise une fonction non linéaire appelée la fonction « sigmoïde » f.

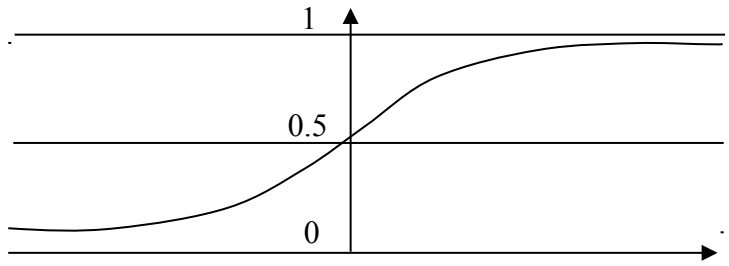

Figure 2 : la fonction sigmoïde f.

La figure 2 montre la fonction sigmoïde dont la formule est :

$$f(x) = 1/(1+e^{-x}) \qquad (1a)$$

On vérifie que f(0) = 0.5 que $f(-\infty) = 0$ et que $f(+\infty) = 1$. La valeur de l'activation o_x d'une unité x est $o_x = f(net_x)$. Dans l'exemple, on a $o_h = f(4.34) = 0.987$. la fonction f s'appelle la fonction sigmoïde standard ou la fonction logistique. De manière générale, la fonction f s'appelle la fonction de transfert, ou fonction d'activation, ou encore fonction « écrasante » ou « aplatissante ».

Pour calculer z, on a :

$$net_z = 1.0x(-4.95) + 0.0x(-4.95) + 0.987x10.9 + 1.0x(-3.29) = 2.52.$$

Donc $o_z = f(2.52) = 0.926$.

0.926 est différent de 1, mais pour notre exemple, c'est une bonne approximation. Avec le réseau de la figure 1, on obtient la table 2, proche de la table 1 :

x	0	0	1	1
y	0	1	0	1
z	0.067	0.926	0.926	0.092

Table 2 : la sortie du réseau de neurones.

On peut écrire plus rapidement le calcul de la valeur d'activation o_j d'un neurone j en fonction de ses entrées i par la formule suivante :

$$o_j = f(net_j) \text{ avec } net_j = \Sigma_i \, w_{ij} \, o_i \qquad (1b)$$

w_{ij} est le poids de la connexion reliant le neurone i avec le neurone j. La somme s'applique pour toutes les connexions partant d'un neurone i et arrivant sur le neurone j.

Apprentissage des poids du réseau avec « backprop » :

Pour obtenir le réseau avec les poids de la figure 1, on a utilisé l'algorithme backprop. Le but de cette partie est de montrer le fonctionnement de backprop sur l'exemple du XOR.

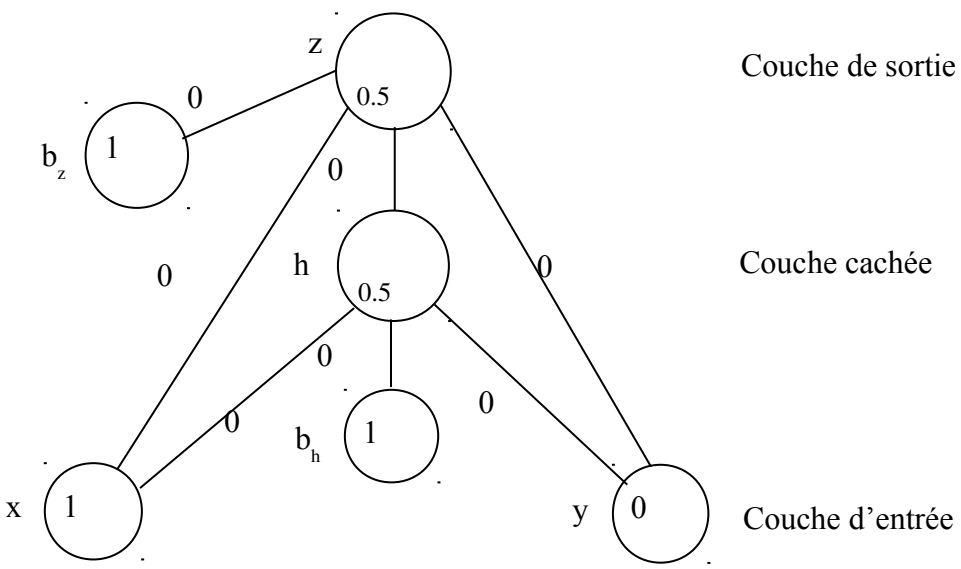

Figure 3 : Le réseau avant le démarrage de l'apprentissage.

Le but de l'apprentissage est de trouver les poids corrects des connexions, c'est-à-dire donnant au réseau un comportement se rapprochant le plus possible de celui de la table 1. La figure 3 montre le réseau supposé avant le démarrage de l'apprentissage. Les poids des connexions sont nuls pour raison de simplicité pédagogique. En pratique, les poids sont initialisés aléatoirement.

Ici nous considérons la présentation de l'exemple (x, y) tel que x=1, y=0, on trouve : o_h=0.5, o_z=0.5. La cible (target) pour z est t_z=1 et l'erreur sur cette unité est donc 0.5.

Le cœur de « backprop » est le suivant :

1° Mettre un exemple à apprendre en entrée du réseau.
2° Calculer les valeurs d'activation des neurones cachés et de sortie avec (1a) et (1b).
3° Calculer l'erreur entre la valeur de l'exemple et celle du réseau avec (2) et (3).
4° Mettre à jour les poids des connexions allant sur l'unité de sortie avec (4).
5° Calculer l'erreur dans les unités de la couche cachée avec (5).
6° Mettre à jour les poids des connexions allant sur la couche cachée avec (6).

Répéter les pas 1 à 6 pour les autres exemples à apprendre. L'ensemble de ces répétitions se nomme une *itération*. Après une itération, la sortie du réseau sera un peu plus proche de la bonne solution. L'algorithme « backprop » consiste à effectuer plusieurs itérations jusqu'à ce que l'erreur soit suffisamment petite.

Les formules utilisées pour les pas 4, 5, et 6 sont les suivantes. Soit t_k la valeur souhaitée de l'unité k et o_k sa valeur d'activation. T pour « target » et O pour « output ». On appelle d_k le « signal d'erreur » défini par la formule :

$$d_k = (t_k - o_k)\, f'(net_k) \qquad (2)$$

où f' est la dérivée de f. Si on utilise pour f la fonction sigmoïde, alors on peut montrer facilement que :

$$f'(net_k) = o_k\,(1 - o_k) \qquad (3)$$

La formule pour changer le poids w_{jk} entre l'unité de sortie k et l'unité j est:

$$\Delta w_{jk} = \nu\, d_k\, o_j \qquad (4)$$

ν se nomme le pas d'apprentissage de l'algorithme. La formule 4 se comprend intuitivement de la manière suivante : on modifie un w poids proportionnellement à la sortie de l'unité associée à la connexion, à l'erreur calculée par 2 et le pas de l'apprentissage ν. Dans l'exemple on a ν=0.1. Avec le réseau de la figure 3, on a successivement :

$$d_z = (1 - 0.5)\, 0.5\, (1 - 0.5) = 0.125$$

$$w_{xz} = 0 + 0.1 \times 0.125 \times 1 = 0.0125$$

$$w_{yz} = 0 + 0.1 \times 0.125 \times 0 = 0$$

$$w_{hz} = 0 + 0.1 \times 0.125 \times 0.5 = 0.00625$$

$$w_{bz} = 0 + 0.1 \times 0.125 \times 1 = 0.0125$$

La formule pour calculer le signal d'erreur d_j d'une unité cachée j est:

$$d_j = f'(net_j) \sum_k d_k\, w_{jk} \qquad (5)$$

La somme s'applique pour toutes les connexions arrivant sur un neurone k et partant du neurone j. Dans notre exemple, l'unité h ne possède qu'une seule connexion sortante (vers z). On a :

$$d_h = 0.5 \text{ x } (1 - 0.5) \text{ x } 0.125 \text{ x } 0.00625 = 0.000195$$

La formule pour changer le w_{ij} entre l'unité cachée j et l'unité d'entrée i est :

$$\Delta w_{ij} = \nu \, d_j \, o_i \qquad (6)$$

Les nouveaux poids des connexions arrivant sur l'unité h seront:

$$w_{xh} = 0 + 0.1\text{x}0.000195\text{x}1 = 0.0000195$$

$$w_{yh} = 0 + 0.1\text{x}0.000195\text{x}0 = 0$$

$$w_{bh} = 0 + 0.1\text{x}0.000195\text{x}1 = 0.0000195$$

Avec ces nouveaux poids, la valeur d'activation de z est 0.507. Si on répète la même procédure pour les trois autres exemples, on obtient les poids de la première itération. Et on obtient la table 3 :

x	0	0	1	1
y	0	1	0	1
z	0.49989	0.49983	0.49983	0.49977

Table 3 : la sortie du réseau de neurones après une itération.

On observe que les valeurs d'activation ont très légèrement changé. Pour aboutir à des sorties égales à la vraie fonction XOR de la table 1, avec une erreur de 0.1, il faut 20,682 itérations, ce qui est très grand pour un problème aussi simple. On peut augmenter le pas de l'apprentissage ν. La table 4 montre que l'on peut réduire le nombre d'itérations à 480 avec ν=2.0.

ν	0.1	0.5	1.0	2.0	3.0
N	20,682	2,455	1,060	480	-

Table 4 : le nombre d'itérations N nécessaires pour obtenir une erreur de 0.1,
pour différentes valeurs du pas d'apprentissage ν.

Lorsque le pas d'apprentissage est trop grand, la méthode ne converge pas. La table 5 montre la valeur de z après 10,000 itérations.

x	0	0	1	1
y	0	1	0	1
z	0.009	0.994	0.994	0.999

Table 5 : la sortie incorrecte du réseau avec un pas d'apprentissage trop grand.

On observe que z vaut 1 au lieu de 0 si x=1 et y=1. La méthode d'apprentissage a convergé vers une mauvaise valeur.

Lors de la mise à jour avec un exemple, on a calculé et mis à jour les poids des connexions de la couche de sortie pour calculer l'erreur de la couche cachée. On aurait pu calculer les poids de la couche de sortie, ne pas les mettre à jour tout de suite, calculer l'erreur de la couche cachée avec les anciens poids, puis mettre à jour les poids de la couche de sortie. De même, on peut faire les calculs d'erreurs sans mettre à jour les poids. Puis lorsque tous les exemples sont présentés, mettre à jour tous les poids. Une controverse existe entre les deux approches.

Reproduction de l'expérience:

La convergence dépend des valeurs initiales des poids des connexions du réseau et de la valeur de ν. Le nombre d'itérations dépend du critère d'arrêt que nous avons fixé à $\varepsilon = 0.01$. Avec des valeurs initiales du réseau *nulles*, les résultats suivants ont été obtenus:

ν	***convergence ?***	***nombre d'itérations***
0,1	oui	459000
0,25	oui	174000
0,5	oui	85000
1	oui	42000
2	oui	21000
4	non	-

Table 6 : convergence et nombre d'itérations observés en fonction de ν.

Dans les cas de convergence, le réseau suivant a été produit:

w_{xh}	w_{yh}	w_{bh}	w_{xz}	w_{yz}	w_{hz}	w_{bz}
8,9	8,9	-3,88	-9,46	-9,46	19,56	-5,23

Table 7 : poids obtenus après convergence et valeurs initiales nulles.

Avec les valeurs initiales *aléatoires* suivantes:

w_{xh}	w_{yh}	w_{bh}	w_{xz}	w_{yz}	w_{hz}	w_{bz}
0,34	-0,1	0,28	0,29	0,41	-0,3	-0,16

Table 8 : valeurs initiales aléatoires.

des résultats analogues ont été obtenus. Les poids sont différents de ceux obtenus avec les valeurs initiales nulles:

w_{xh}	w_{yh}	w_{bh}	w_{xz}	w_{yz}	w_{hz}	w_{bz}
9,88	-9,09	4,62	9,68	-9,37	-19,5	14,42

Table 9 : poids obtenus après convergence et valeurs initiales de la table 8.

« Backprop » sur l'exemple du OR avec un unique neurone

Cet exemple est montre le fonctionnement du cœur de « Backprop » avec un *unique neurone* sur le problème du OR(x, y) en fonction de x et y. La fonction z = OR(x, y) est définie par la table 10. On ne peut pas donner un exemple plus simple.

x	0	0	1	1
y	0	1	0	1
z	0	1	1	1

Table 10 : la fonction OR.

La figure 4 représente un neurone en cours d'apprentissage pour problème du OR. On suppose que l'on a w_{xz}=5 et w_{yz}=-3 et w_{bz}=1.

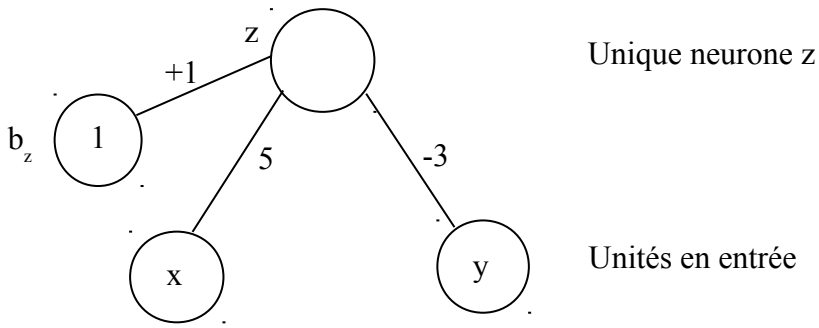

Figure 4 : Un neurone en cours d'apprentissage.

L'équation caractéristique du neurone est :

$$net_z = 5\ x - 3\ y + 1 = 0 \qquad \text{(delta)}$$

Cette équation correspond à une droite delta. Lorsque net_z est positif alors la sortie du neurone est supérieure à ½ et on interprète la sortie du neurone comme un 1, ou un '+'. Lorsque net_z est négatif alors la sortie du neurone est inférieure à ½ et on interprète la sortie du neurone comme un 0, ou un '-'. La figure 5 représente graphiquement la sortie du neurone.

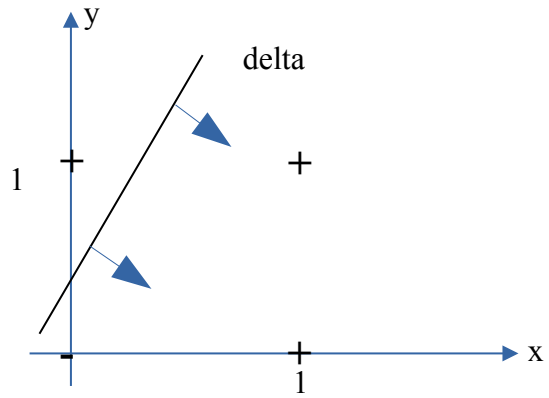

Figure 5 : La droite delta sépare le plan en deux demi-plans.

Le demi-plan en bas à droite de delta correspond à une sortie 1 ou '+'. Le demi-plan en haut à gauche de delta à une sortie 0 ou '-'. Autrement dit, le neurone de la figure 4 classifie correctement les exemples (1, 0) et (1, 1) en '+'. Il classifie incorrectement les exemples (0, 0) et (0, 1).

Dans la suite, on va présenter *successivement* les 4 exemples au cœur de Backprop et voir comment les poids évoluent. On suppose que $\nu = 0.1$.

On présente **(0, 0) avec t_z=0.**
On a $net_z = 1$ et $o_z = 0.7$. L'exemple est *mal* classé.
Ensuite $d_z = (0-0.7)0.7(1-0.7)= -0.7\ 0.7\ 0.3 = -0.15$
Puis $w_{xz} = 5$ $w_{yz} = -3$ $w_{bz} = 1 + 0.1(-0.15)\ 1 = 0.985$
Interpétation graphique : la droite delta représentant le neurone a descendu légèrement.
Il faut comprendre ce changement comme un changement important car (0,0) est mal classé par le neurone. Backprop effectue des « grosses » modifications de poids lorsque l'exemple est mal classé.

On présente **(0, 1) avec t_z=1.**
On a $net_z = -3 + 0.99 = -2.01$ et $o_z = 0.125$. L'exemple est *mal* classé.
Ensuite $d_z = (1-0.125)\ 0.125\ (1-0.125)= 0.08 = 0.1$
Puis $w_{xz} = 5$ $w_{yz} = -3 + 0.01 = -2.99$ $w_{bz} = 0.985 + 0.01 = 0.995$
Interpétation graphique : la droite delta légèrement tourné dans le sens trigonométrique. Même remarque que précédemment.

On présente **(1, 0) avec t_z=1.**
On a $net_z = 5+1 = 6$ et $o_z = 0.99$. L'exemple est *bien* classé.
Ensuite $d_z = 0.0001$
Puis $w_{xz} = 5.00001 = 5$ $w_{yz} = -2.99$ $w_{bz} = 0.995 + 0.00001 = 0.995$
Interpétation graphique : la droite delta ne bouge quasiment pas.

On présente **(1, 1) avec t_z=1.**
On a a $net_z = 5-3+1=3$ et $o_z = 0.97$. L'exemple est *bien* classé.
Ensuite $d_z = (1-0.97)0.97(1-0.97) = 0.001$
Puis $w_{xz} = 5.0001 = 5$ $w_{yz} = -2.99 +0.0001 = -2.99$ $w_{bz} = 0.995 + 0.0001 = 0.995$
Interpétation graphique : la droite delta ne bouge quasiment pas.

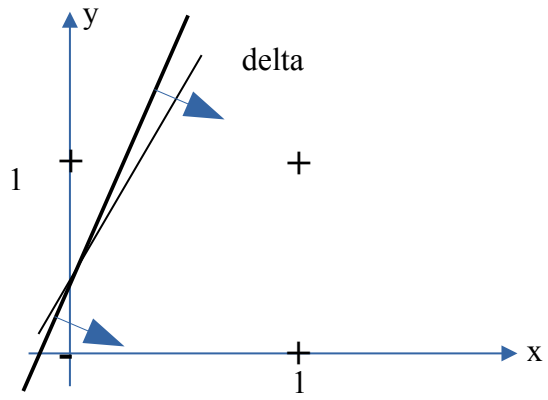

Figure 6 : La droite delta s'est déplacée.

Après la présentation des 4 exemples, la droite delta s'est déplacée depuis sa position de la figure 5 vers la position en gras de la figure 6.

Exemple de « Backprop » avec un neurone du pire cas

Cet exemple est montre le fonctionnement de « Backprop » avec un neurone de départ mal initialisé. La fonction z = Y(x,y) = y est celle de la table 11.

x	0	0	1	1
y	0	1	0	1
z	0	1	0	1

Table 11 : la fonction Y.

On suppose que les valeurs initiales du réseau correspondent au *pire cas* : les exemples sont mal classés et la droite delta du neurone est située entre les exemples + et -, et *tournée dans le mauvais sens*. La figure 7 représente le neurone du pire cas pour le problème du Y : on suppose que l'on a w_{xz}=0 et w_{yz}=-1 et w_{bz}=1/2.

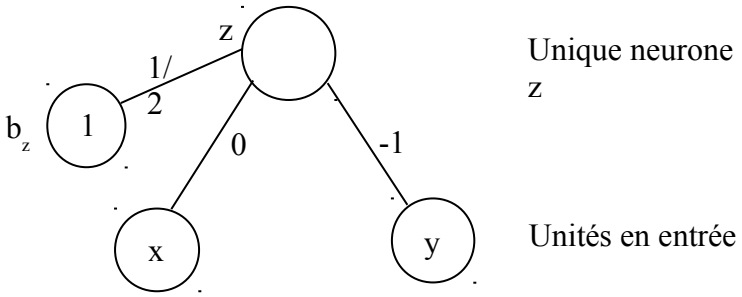

Figure 7 : neurone du *pire cas* pour le problème du Y.

L'équation caractéristique de la droite delta du neurone est :

$$net_z = - y + 1/2 = 0 \qquad \text{(delta)}$$

La figure 8 représente graphiquement la sortie du neurone.

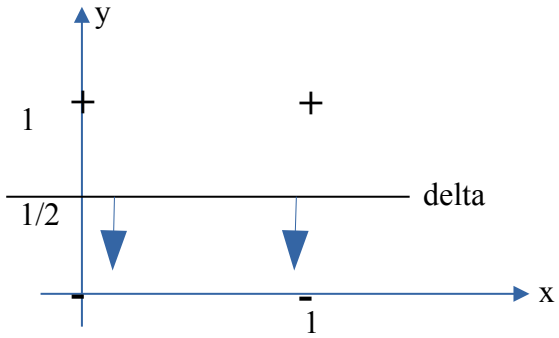

Figure 8 : La droite delta est tournée dans le *mauvais sens* : les – sont classés + et réciproquement.

Le demi-plan en bas correspond à une sortie 1 ou '+'. Le demi-plan en haut à une sortie 0 ou '-'. Autrement dit, le neurone de la figure 8 classifie *incorrectement* les 4 exemples.

Pour être pédagogique, on suppose que l'on présente un des 4 exemples sur le réseau de la figure 7 (pas de présentations successives). Le but est de voir comment les poids du réseau évoluent pour chacun des 4 cas sur ce neurone de la figure 7. On suppose que $\nu = 0.1$.

Si on présente **(0, 0) avec t_z=0.**
On a net_z = 1/2 et o_z = f(1/2) = 0.62. L'exemple est *mal* classé.
Ensuite d_z = (0-0.62)0.62(1-0.62) = -0.15
Puis w_{xz} = 0 et w_{yz} = -1 sont inchangés et w_{bz} = 1/2 + 0.1(-0.15) 1 = 0.485
Interpétation graphique : la droite delta a descendu légèrement.

Si on présente **(1, 0) avec t_z=0.**
On a net_z = 1/2 et o_z = f(1/2) = 0.62. L'exemple est *mal* classé.
Ensuite d_z = -0.15
Puis w_{xz} = -0.015 et w_{yz} = -1 est inchangé et w_{bz} = 0.485.
Interpétation graphique : la droite delta a légèrement descendu et tourné dans le sens des aiguilles d'une montre.

Si on présente **(0, 1) avec t_z=1.**
On a net_z = -1/2 et o_z = 0.38. L'exemple est *mal* classé.
Ensuite d_z = 0.15
Puis w_{xz} = 0 est inchangé et w_{yz} = -1 + 0.015 = -0.985 et w_{bz} = 0.5 + 0.015 = 0.515
Interpétation graphique : la droite delta monte légèrement.

Si on présente **(1, 1) avec t_z=1.**
On a a net_z = -1/2 et o_z = 0.38. L'exemple est *mal* classé.
Ensuite d_z = 0.15
Puis w_{xz} = 0.015 et w_{yz} = -1 +0.015 = -0.985 et w_{bz} = 0.515
Interpétation graphique : la droite delta monte et pivote légèrement dans le sens trigonométrique.

Sur la base qualitative de ce qui précède, il est difficile de prévoir le déplacement de la droite delta après la présentation *successive* des 4 exemples.

En effet, (0, 0) fait descendre la droite. (0, 1) fait monter la droite. La présentation de ces deux exemples ont des effets opposés dont on peut estimer que la somme a un effet proche de nul. De manière analogue, la présentation des exemples (1, 0) et (1, 1) ont des effets opposés : faire pivoter delta dans le sens des aiguilles du montre pour le premier et dans le sens trigonométrique pour le second.

Le déplacement de la droite delta après présentations successives des 4 exemples dépend de l'ordre de présentation des exemples. Si on présente (0,0) puis (0,1), l'effet de la dernière présentation sera légèrement plus fort que celui de la première, donc la droite montera très très légèrement.

Dans ce « pire cas » illustré par cet exemple, on peut dire que la droite delta est en quelque sorte « coincée » : elle ne sait pas si elle doit monter ou descendre, pivoter dans le sens trigonométrique ou des aiguilles d'une montre. Suite à l'effet de présentation successive des premiers exemples, la droite va bouger dans un sens ou un autre, tourner dans un sens ou un autre, et le mouvement de la droite sera amorcé.

Ensuite, au moyen de nombreuses itérations de Backprop, la droite delta exécutera son « demi-tour » en poursuivant le mouvement de départ. Après un nombre suffisant d'itérations, delta aura terminé le demi-tour et le neurone classera correctement les 4 exemples.