Support théorique de « backprop » :

Cette partie présente la théorie qui sous-tend l'algorithme « backprop » dont une exécution a été montrée à la partie précédente. La figure 4 montre un réseau plus général que celui du XOR. Il sert de support aux explications données dans cette partie.

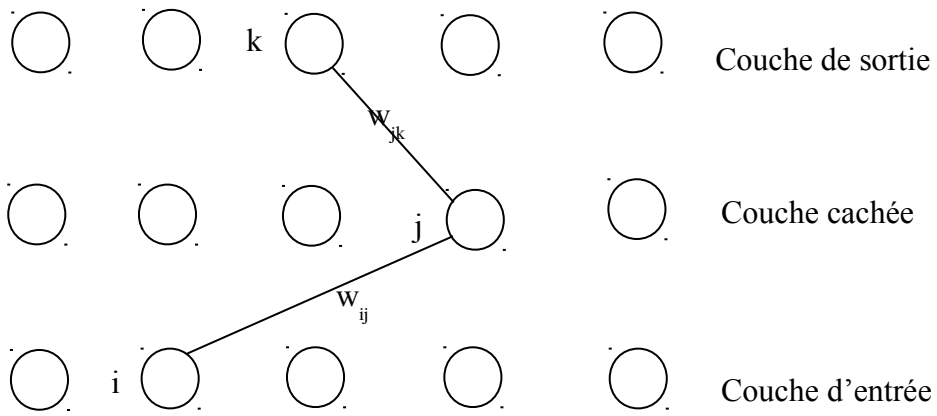

Figure 4 : Un réseau général à trois couches. Si w_{jk} change alors, seul k change. Si w_{ij} change alors toutes les valeurs de la couche de sortie changent.

Dans ce réseau, on a toujours :

$$o_k = 1/(1+e^{-net_k}) \qquad (7)$$

$$net_k = \Sigma_j w_{jk} o_j \qquad (8)$$

L'algorithme "backprop" suppose que l'on cherche à minimiser l'erreur de la sortie sur tous les exemples, en utilisant la formule d'erreur suivante :

$$E = \tfrac{1}{2} \Sigma_p \left(\Sigma_k (t_{pk} - o_{pk})^2 \right) \qquad (9a)$$

p désigne un exemple et k une unité de sortie du réseau. t_{pk} est la valeur souhaitée de l'unité de sortie k du réseau quand l'exemple p est présenté. o_{pk} est la valeur effective de l'unité de sortie k du réseau quand l'exemple p est présenté. La fonction d'erreur de la formule 9 est la plus utilisée, bien que d'autres fonctions d'erreurs soient possibles. Noter que E est une somme sur tous les exemples p. On peut supposer que en minimisant l'erreur pour chaque exemple pris individuellement, on minimise aussi l'erreur E. Donc dans la suite, pour simplifier nous éliminons p et obtenons la formule suivante.

$$E = \tfrac{1}{2} \Sigma_k (t_k - o_k)^2 \qquad (9b)$$

Backprop est donc une descente de gradient pour la fonction E. Il faut étudier d'une part comment E varie en fonction de w_{jk}, un poids d'une connexion arrivant vers la couche de sortie, et d'autre part comment E varie en fonction de w_{ij}, un poids d'une connexion arrivant vers la couche cachée. Pour la première part on écrit :

$$\partial E/\partial w_{jk} = \partial E/\partial o_k \; \partial o_k/\partial net_k \; \partial net_k/\partial w_{jk} = -(t_k - o_k)\, o_k\, (1 - o_k)\, o_j \qquad (10)$$

Donc pour modifier w_{jk} suivant l'algorithme de descente de gradient, on écrit :

$$\Delta w_{jk} = -\nu \partial E/\partial w_{jk} \qquad (11)$$

Usuellement on définit le signal d'erreur d_k par les formules 2 et 3 vues précédemment, d'où :

$$\Delta w_{jk} = \nu\, d_k\, o_j \qquad (12)$$

Pour la seconde part du problème relative à la variation de E en fonction d'un poids w_{ij} arrivant sur une unité j de la couche cachée, on doit tenir compte de toutes les unités de la couche de sortie reliées à l'unité j. La variation de E est donc fonction de la somme des variations de chaque unité de sortie, ce qui donne :

$$\partial E/\partial w_{ij} = \Sigma_k \, \partial E^k \partial w_{ij}$$

$$\partial E/\partial w_{ij} = \Sigma_k \, \partial E^k/\partial o_k \, \partial o_k/\partial net_k \, \partial net_k/\partial o_j \, \partial o_j/\partial net_j \, \partial net_j/\partial w_{ij}$$

$$\partial E/\partial w_{ij} = \Sigma_k - (t_k - o_k)\, o_k\, (1 - o_k)\; w_{jk}\, o_j\, (1 - o_j)\; o_i$$

$$\partial E/\partial w_{ij} = \Sigma_k - d_k\, w_{jk}\, o_j\, (1 - o_j)\; o_i$$

(d'après les formules 2 et 3)

$$\partial E/\partial w_{ij} = -o_j(1 - o_j) o_i \; \Sigma_k \, d_k w_{jk}$$

$$\partial E/\partial w_{ij} = -o_i\, d_j$$

(d'après la formule 5) Donc, le changement de w_{ij} suivant l'algorithme de descente de gradient est :

$$\Delta w_{ij} = \nu\, d_j\, o_i \qquad (14)$$

Ces résultats peuvent être généralisés à des réseaux avec plus de trois couches avec trois formules synthétiques. La première est la formule 14 donnant le changement d'un poids du réseau. La seconde est la formule 15, synthèse des formules 2 et 3, donnant le signal d'erreur pour une unité j de la couche de sortie :

$$d_j = (t_j - o_j)\, o_j\, (1 - o_j) \qquad (15)$$

La troisième formule est la formule 16 qui reprend 5 et donne le signal d'erreur pour une unité j de la couche cachée avec des unités k au dessus d'elle:

$$d_j = o_j\, (1 - o_j)\, \Sigma_k \, d_k\, w_{jk} \qquad (16)$$

Réseaux à une couche

Fonctions linéaires généralisées

la fonction de transfert n'est pas nécessairement une sigmoïde. Dans les « fonctions linéaires généralisées », la fonction de transfert f est la fonction identité donc f'(net) = 1.

Perceptron

perceptrons (Rosenblatt 1962), adalines (Widrow & Hoff 1960). La fonction de transfert est la fonction sigmoïde.

Séparabilité linéaire

Un ensemble d'exemples est séparable linéairement s'il existe un hyperplan séparant l'ensemble des exemples de celui des contre-exemples. La figure 5 montre un ensemble d'exemples et de contre-exemples séparables linéairement (a), et un ensemble non séparable (b).

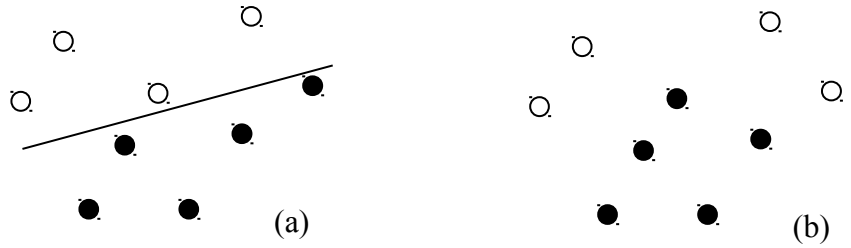

Figure 5 : un ensemble d'exemples (ronds blancs) et de contre-exemples (ronds noirs) séparables linéairement (a) et le même ensemble dans lequel un exemple et un contre-exemple ont changé de classe : il n'est plus séparable linéairement.

Théorème de convergence : pour tout ensemble de données séparable linéairement, le perceptron converge vers une solution en un nombre fini de pas. La figure 6 donne une explication visuelle du fonctionnement du perceptron dans le cas d'un réseau de neurones à une couche.

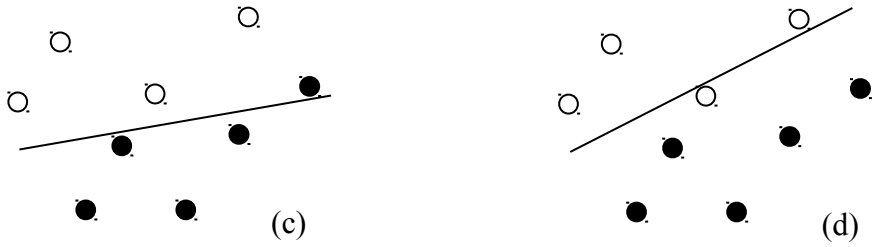

Figure 6 : lorsque l'hyperplan séparateur classe mal un exemple (c), une itération de backprop sur cet exemple change l'hyperplan pour améliorer le classement de cet exemple (d). Si d'autres exemples sont mal classés, l'algorithme continue.

Limitations : pour certains problèmes, le perceptron est incapable de les résoudre [Minsky & Papert 1969]. Par exemple, la fonction XOR ne marche pas. Fonctions d'entrées binaires, la fraction des applications d'appartenance a deux classes qui peuvent être implémentée par un perceptron sont appelées des fonctions logiques a seuil et elles forment un extrêmement petit sous-ensemble de l'ensemble total.

Réseaux à plusieurs couches ou « Multi-Layer Perceptron »

Cette partie résume ce qu'il faut savoir sur les réseaux de neurones multi-couches.

Diagrammes de Hinton

Les diagrammes de Hinton montrent visuellement les poids des connexions d'un réseau. La figure 7 montre le diagramme de Hinton pour un réseau à deux couches résolvant le problème du XOR. La fonction d'activation est la sigmoïde. A gauche, les unités de biais sont indiquées par des ronds noirs petits ; les unités d'entrée sont des petits ronds blancs ; les unités proprement dites sont indiquées par des grands ronds blancs ; les poids des connexions différents de un ne sont pas indiqués. A droite, un poids de connexion est représenté par un carré dont la taille est proportionnelle à sa valeur absolue. Un carré blanc (respectivement noir) indique un poids négatif (respectivement positif).

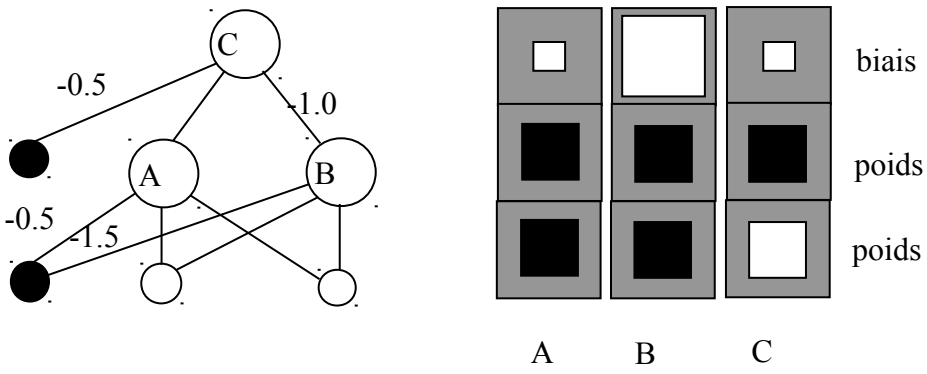

Figure 7 : le diagramme de Hinton (à droite) pour un réseau à deux couches résolvant le problème du XOR (à gauche).

En utilisant une fonction f à seuil, on vérifie que le réseau de la figure 7 donne le bon résultat :

X	***Y***	***A***	***f(A)***	***B***	***f(B)***	***C***	***f(C)***
0	0	-0.5	0	-1.5	0	-0.5	0
0	1	+0.5	1	-0.5	0	+0.5	1
1	0	+0.5	1	-0.5	0	+0.5	1
1	1	+1.5	1	+0.5	1	-0.5	0

D'où viennent les valeurs des connexions de la figure 7 ?
Tracer les hyperplans séparateurs nécessaires pour résoudre ce problème.
Y a-t-il une autre solution simple à ce problème ?

Comment simuler le ET logique de N unités ? Avec w=1 et biais=-N+0.5
Comment simuler le OU logique de N unités ? Avec w=1 et biais=-0.5
Comment simuler le NON logique de 1 unité ? Avec w=-1 et biais=0.5

Résultats théoriques

Il faut bien distinguer résultat théorique et résultat pratique. Résultat théorique signifie que l'on peut démontrer qu'un réseau peut simuler une fonction ou identifier une zone, indépendamment de backprop. La démonstration peut éventuellement préciser comment construire le réseau répondant à la question, on dit que la démonstration est constructive. Un résultat théorique n'entraîne pas qu'il sera facile avec backprop de trouver la bonne architecture de réseau avec les poids adéquats.

Résultat pratique signifie que l'on veut un programme utilisant un réseau de neurones et donnant les bons résultats. Le résultat pratique peut ou non utiliser backprop. En résumé de cette introduction, bien faire attention au fait que, vus au travers de la théorie, les résultats pratiques sont évidemment inférieurs aux résultats théoriques.

On a vu que la frontière de décision d'un réseau à une couche est un hyperplan (cf (a) figure 5).

Fonctions booléennes

On a vu que les fonctions booléennes ne peuvent pas toutes être simulées avec un réseau à une couche. En revanche, un réseau à 2 couches peut implémenter n'importe quelle fonction booléenne pourvu que le nombre d'unités cachées soient assez grand (McCulloch & Pitt 1943). Comment montrer qu'une fonction booléenne à d entrées peut être simulée par un réseau à deux couches ?

Il y a 2^d entrées possibles. On prend autant d'unités dans la couche cachée que d'exemples dont la valeur de la fonction booléenne est 1. Une unité de la couche cachée ne répond que à un seul exemple en entrée. On fait cela en mettant à +1 les poids des connexions reliées aux 1 de l'exemple, et à -1 les poids des connexions reliées aux 0 de l'exemple. Le poids du biais est fixé à 0.5 – b, où b est le nombre d'entrées non nulles de l'exemple. Ainsi, sur son exemple, l'unité répond $b+0.5-b=0.5>0$. Et sur un exemple différent il répond au plus $b-2+0.5-b=-1.5<0$. Avec la fonction sigmoïde, sur son exemple, l'unité répond 1 et sur un autre exemple, elle répond 0. Pour l'unité de sortie, on met des poids égaux à 1 et un biais égal à –0.5. Ainsi, sur un exemple dont la valeur de la fonction booléenne est 1, son unité répond 1 et les autres répondent 0, donc l'unité de sortie répond $1-0.5=0.5>0$. Sur un contre-exemple, toutes les unités cachées répondent 0, donc l'unité de sortie répond $-0.5<0$.

Reconnaissance d'une zone convexe de l'espace avec un réseau à 2 couches

Lorsque les entrées (et sorties) sont des variables numériques, la frontière de décision d'un réseau à 2 couches est un ensemble de morceaux d'hyperplans définissant une zone convexe.

Supposons que l'on veuille classifier les exemples de la figure 8. Avec les droites a, b, c il est possible d'entourer les exemples + sans exemple -. A chaque droite on peut faire correspondre un demi-plan tel que les exemples + soit situés dans ce demi-plan (les flèches bleues de la figure 8). A chaque demi-plan correspond un neurone reconnaissant les points du demi-plan comme des exemples +. En combinant les neurones a, b, c avec un neurone ET logique on obtient le réseau de neurones de la figure 9. Ce réseau à 2 couches classe correctement les exemples + et – dans la figure 8.

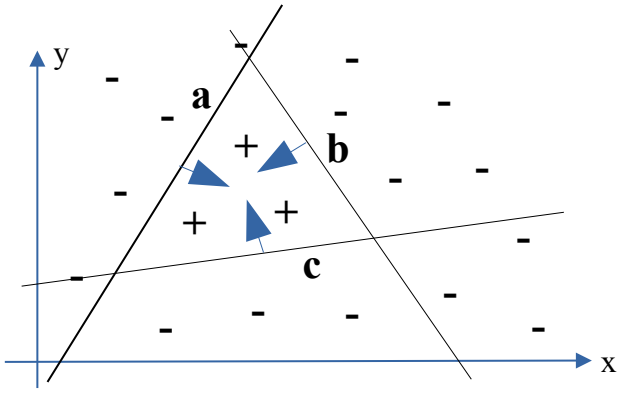

Figure 8 : Les exemples + sont dans une zone convexe et les – à l'extérieur.

Pour effectuer un ET logique sur M unités, il suffit de mettre toutes les connexions avec un poids +1 et le poids de l'unité de biais égal à –M+0.5. Ainsi, le neurone ET a un net_{ET} égal à nombre d'unités à 1 – M+0.5 qui est positif si et seulement si le nombre d'unités à 1 égal M.

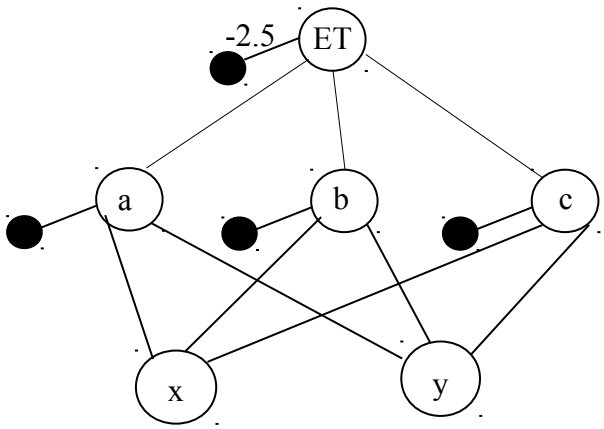

Figure 9 : Le ET logique permet de reconnaître une zone convexe.

Avec un réseau à deux couches, pourvu que le nombre d'unités de la couche cachée soit assez grand, on peut théoriquement engendrer *certaines* régions non convexes ou disjointes (Huang & Lippman 1988). En revanche, on ne peut pas engendrer des zones arbitrairement complexes avec un réseau à 2 couches.

Reconnaissance d'une union de zones convexes de l'espace avec un réseau à 3 couches

Pour un réseau à 3 couches, théoriquement, pourvu que le nombre d'unités des couches cachées soit assez grand, les zones sont arbitraires, non convexes, disjointes, limitées par des hyperplans (Lippman 1987). Dans le cas où la zone à définir est non convexe et disjointe, elle est une réunion de zones convexes élémentaires. Pour chaque zone convexe élémentaire de la zone à définir, on associe une unité de la deuxième couche cachée en faisant comme avec un réseau à deux couches, celle-ci répond 1 lorsque un exemple est dans cette zone convexe élémentaire, 0 sinon. Enfin, l'unité de sortie réalise un OU logique des unités de la deuxième couche en mettant un poids 1 à chaque connexion et un biais –0.5. En pratique, un réseau à 3 couches permet de définir n'importe quelle zone non convexe et disjointe.

Supposons que l'on veuille classifier les exemples de la figure 10. Avec les droites a, b, c il est possible d'entourer les exemples + de la première zone convexe, et avec les droites d, e, f, g ceux de la seconde zone convexe. A chaque droite, on fait correspondre un demi-plan tel que les exemples + soit situés dans ce demi-plan (les flèches bleues de la figure 10). A chaque demi-plan correspond un neurone reconnaissant les points du demi-plan comme des exemples +. On combine les neurones a, b, c avec un neurone ET logique et les neurones b, c, d, e avec un autre neurone ET. On combine les deux neurones ET avec neurone OU pour donner le réseau de la figure 11. Ce réseau à 3 couches classe correctement les exemples + et – dans la figure 10.

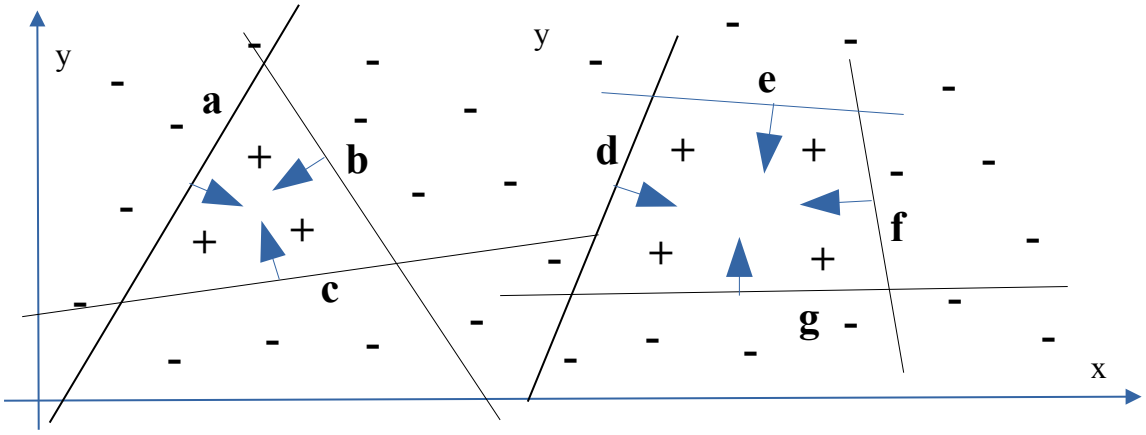

Figure 10 : Les exemples + sont dans une union de zones convexes et les – à l'extérieur.

NB : Pour effectuer un OU logique sur N unités, il suffit de mettre toutes les connexions avec un poids +1 et le poids de l'unité de biais égal à –0.5. Ainsi, le neurone OU a un net_{OU} égal à nombre d'unités à 1 – 0.5 qui est positif si et seulement si au moins une unité est à 1.

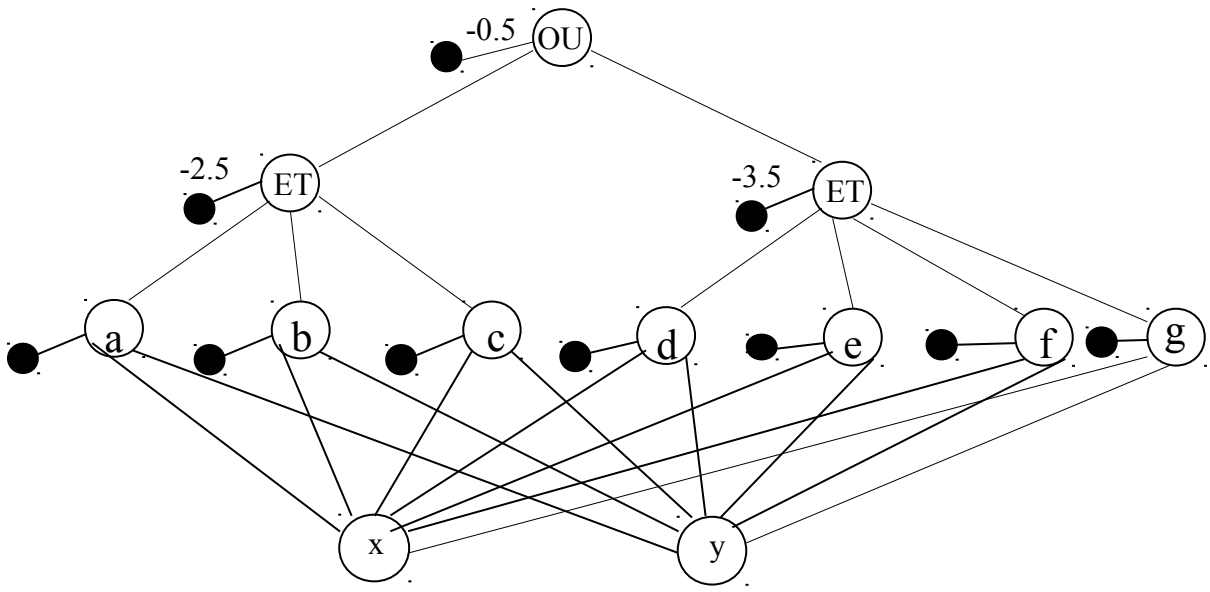

Figure 11 : Un OU et des ET logiques permettent de reconnaître une union de zones convexes.

Résumé du cœur de back-prop

1 : applique un exemple et propage le réseau en avant avec (7) et (8)
2 : évalue δ_k pour chaque unité de sortie avec (15)
3 : propage en arrière les δ pour obtenir les δ_j pour les unité cachées avec (16)
4 : mettre a jour les poids avec la formule de descente de gradient avec (14)

Efficacité computationnelle

La caractéristique essentielle de MLP back-prop est son *efficacité computationnelle*. c'est un algorithme en O(W) où W est le nombre de poids. A comparer avec la résolution de l'équation d'annulation de la fonction d'erreur qui suppose une inversion de matrice et qui est en $O(W^2)$.

Exercice 1

On lance BackProp sur un unique neurone avec $\nu = 1$. On appelle Δ la droite séparatrice du neurone. Soient les 4 exemples suivants :

(x=0, y=1), + (x=1, y=1), + (x=0, y=0), - (x=1, y=0), -

1) On suppose que, initialement, Δ est **horizontale**, **orientée dans le mauvais sens** : on a Wxz = 0, Wyz = -1, Bz = 0.5. Pour chacun des exemples, quelles sont les nouvelles valeurs de Wxz, Wyz et Bz après une application des règles (14) et (15) de backprop ? Donner l'interprétation géométrique.

(x=0, y=0) : T=0 Oz = f(0.5) ~= 0.7 Dz = -0.7 0.7 0.3 ~= -0.15
Wxz = 0 + 0 = 0 Wyz = -1 + 0 = -1 Bz = 0.5 - 0.15 = 0.35
Δ « descend » légèrement.
(x=1, y=0) : T=0 Oz = f(0.5) ~= 0.7 Dz = -0.7 0.7 0.3 ~= -0.15
Wxz = 0 -0.15 = -0.15 Wyz = -1 + 0 = -1 Bz = 0.5 - 0.15 = 0.35
Δ « descend » légèrement et « pivote » légèrement vers le bas.
(x=0, y=1) : T=1 Oz = f(-0.5) ~= 0.3 Dz = 0.7 0.3 0.7 ~= +0.15
Wxz = 0 + 0 = 0 Wyz = -1 + 0.15 = -0.85 Bz = 0.5 + 0.15 = 0.65
Δ «monte» légèrement.
(x=1, y=1) : T=1 Oz = f(-0.5) ~= 0.3 Dz = 0.7 0.3 0.7 ~= +0.15
Wxz = 0 + 0.15 = 0.15 Wyz = -1 + 0.15 = -0.85 Bz = 0.5 + 0.15 = 0.65
Δ «monte» légèrement et « pivote » légèrement vers le haut.
On constate que l'ordre de présentation des exemples est crucial. Si on les présente dans un ordre circulaire toujours le même Δ oscillera longtemps avant de trouver dans quel sens tourner.

2) On suppose que initialement, Δ est **verticale**, orientée vers la droite : on a Wxz = 1, Wyz = 0, Bz = -0.5. Pour chacun des exemples, quelles sont les nouvelles valeurs de Wxz, Wyz et Bz après une application des règles (14) et (15) de backprop ? Donner l'interprétation géométrique.

(x=0, y=0) : T=0 Oz = f(-0.5) ~= 0.3 Dz = -0.3 0.7 0.3 ~= -0.06
Wxz = 1 + 0 = 1 Wyz = 0 + 0 = 0 Bz = -0.5 - 0.06 = -0.56
Δ se déplace légèrement vers la droite en restant verticale.
(x=1, y=0) : T=0 Oz = f(0.5) ~= 0.7 Dz = -0.7 0.3 0.7 ~= -0.15
Wxz = 1 - 0.15 = 0.85 Wyz = 0 + 0 = 0 Bz = -0.5 - 0.15 = -0.65
Δ se déplace légèrement vers la droite en restant verticale.
(x=0, y=1) : T=1 Oz = f(-0.5) ~= 0.3 Dz = 0.7 0.3 0.7 ~= +0.15
Wxz = 1 + 0 = 1 Wyz = 0 + 0.15 = +0.15 Bz = -0.5 + 0.15 = -0.35
Δ se déplace légèrement vers la gauche en pivotant vers la gauche.
(x=1, y=1) : T=1 Oz = f(0.5) ~= 0.7 Dz = 0.3 0.3 0.7 ~= +0.06
Wxz = 1 + 0.06 = 1.06 Wyz = 0 + 0.06 = 0.06 Bz = -0.5 + 0.06 = -0.44
Δ se déplace légèrement vers la gauche en pivotant vers la gauche.
On constate que l'ordre de présentation des exemples est moins crucial. Le sens de rotation est identique – trigonométrique - quel que soit l'exemple présenté.

Exercice 2

Concevoir et mettre au point avec « backprop » un MLP prenant un damier 3x3 en entrée avec 1 ou 0 sur chaque case et donnant le *nombre de composantes connexes* en sortie. D'abord, on pourra prendre un réseau à 2 couches avec 9 unités dans la couche d'entrée, quelques unités (3 à 6) dans la couche cachée, et une unité de sortie et vérifier que le réseau n'arrive pas à apprendre. Ensuite, on pourra mettre 3 unités dans la couche de sortie, chaque unité de sortie correspondant à un bit du nombre de composantes connexes (compris entre 0 et 5; 3 bits suffisent), et vérifier que l'apprentissage marche. Enfin, on pourra utiliser un réseau à 3 couches en rajoutant une unité de sortie sur le réseau précédent, et vérifier que l'apprentissage marche. Pour chaque expérience, on mesurera l'effet (nombre d'itérations, convergence ou pas) des entités suivantes sur l'apprentissage: poids initiaux des connexions du réseau, ensemble d'apprentissage, valeur du pas d'apprentissage, nombre d'unités des couches cachées, etc.

Exercice 3

Concevoir et mettre au point avec « backprop » un MLP *évaluant un damier 8x8 d'Othello*.

Références

Antoine Cornuéjols, Laurent Miclet, « Apprentissage artificiel, concepts et algorithmes », Eyrolles, 2002, chapitre 11, pages 311-332.

Christopher Bishop, "Neural Networks for Pattern Recognition", Oxford University Press, 1995

Rumelhart, Hinton, Williams, "Learning internal representations by error propagation", in Rumelhart McClelland, PDP research group (eds), parallel distributed processing explorations in the microstructure of cognition, volume 1, foundations, pp. 318-362, Cambridge MA, MIT Press, 1986, (reprinted in Anderson and Rosenfeld 1988).

Donald Tveter, "Chapter 2, the backprop algorithm", (http://www.dontveter.com)

M. Minsky, S. Papert, "Perceptrons", Cambridge, MA: MIT Press, (expanded edition 1990), 1969.

F. Rosenblatt, "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms", Washington DC, Spartan, 1962.

B. Widrow, M.E. Hoff, "Adaptative switching circuits", IRE WESCON record, vol 4, pp. 96-104, 1960.

W. McCulloch, W. Pitts, "A logical calculus of the ideas immanent in nervous activity", Bulletin of Math. Biophysics, vol 5, pp 115-133, (reprinted in Anderson and Rosenfeld 1988), 1943.

W. Huang, R. Lippmann, "neural net and traditional classifiers, in Anderson ed, Neural Information Processing Systems, pp. 387-396, New York, American Institute of Physics, 1988.

R. Lippmann, "An introduction to computing with neural nets", IEEE ASSP Magazine, April 1987, pp. 105-117.