CHAPITRE 7

INITIATION PRATIQUE AUX RÉSEAUX DE NEURONES MULTICOUCHES

Car, c'est un fait, Dieu se trouve dans les détails, et une compréhension intuitives des choses ne peut surgir que d'une fréquentation assidue de leurs aspects particuliers
STEPHEN JAY GOULD*

J'ADHÈRE totalement à l'idée exprimée dans l'épigraphe de ce chapitre ; apprendriez-vous la natation en consultant uniquement des livres théoriques traitant des mouvements nécessaires ou de l'hydrodynamique . . . ? Non, n'est-ce pas ? Vous commenceriez par vous jeter à l'eau ! À mon avis, il est illusoire d'essayer de comprendre *réellement* les réseaux de neurones sans se frotter directement à leur programmation. Ce chapitre vous propose quelques exercices simples qu'il s'avère sage de pratiquer. D'autant plus que le premier (sur le XOR) est souvent une référence dans les discussions sur les réseaux de neurones.

7.1 Introduction

Les réseaux de neurones artificiels sont des modèles informatiques de réseaux d'automates dont la structure et le comportement sont "copiés" sur ceux des neurones réels. À la façon du cerveau, ils peuvent reconnaître des formes, réorganiser des données et, de façon plus intéressante, apprendre. Les réseaux de neurones artificiels sont constitués d'objets qu'on appelera *unités* qui représentent le corps cellulaire des neurones réels. Les unités sont interconnectées par des liens qui agissent comme des axones ou des dendrites. Un lien possède la propriété de multiplier la sortie d'une unité par un facteur appelé *poids*, une quantité analogue à la force de connexion d'une synapse, qui peut être positif ou négatif. Le lien envoie sa valeur, pondérée par le poids, à une autre cellule qui réalise la somme des valeurs reçues par tous les liens. Cette somme pondérée est appellée *entrée totale*. Dans les modèles de conception simple, lorsque l'entrée totale est supérieure à un certain seuil, la cellule décharge, c'est-à-dire qu'elle transmet une valeur non nulle aux neurones qu'elle stimule ; dans les modèles plus complexes, les sorties prennent des valeurs réelles.

Les modifications intervenant dans la séquence de décharge constituent l'*apprentissage*. On pense que dans les neurones réels l'apprentissage se fait grâce aux modifications des synapses : lorsque

*Un hérisson dans la tempête (Grasset ed. 1994)

les forces de connexion entre synapses changent, alors le comportement de tout le réseau est modifié. Dans les réseaux de neurones artificiels l'apprentissage se produit grâce à des modifications des poids des liens, les coefficients synaptiques.

Les réseaux de neurones artificiels que l'on va étudier dans les exercices qui vont suivre, sont constitués de trois types d'unités.

Les unités d'entrée qui reçoivent les informations provenant du monde extérieur. Généralement elles ne transforment pas ces signaux.

Les unités de sortie qui émettent des signaux visibles par le monde extérieur.

Les unités cachées qui se trouvent entre les unités d'entrée et celles de sortie ; elles ne reçoivent pas d'entrées directement de l'extérieur et produisent des sorties qui ne sont pas visibles.

Les unités cachées ainsi que les unités de sortie effectuent généralement une transformation non linéaire de leur entrée totale.

Une illustration d'un réseau simple, ou les entrées et les sorties ne peuvent prendre que des valeurs binaires et où les fonctions de transfert sont à seuil, est donnée sur la figure 7.1. Les nombres situés dans les unités indiquent les valeurs des seuils. Ceux situés le long des lignes de connexion sont les poids des liens. Notez qu'il peut y avoir des courts-circuits : certaines connexions peuvent éviter les unités cachées.

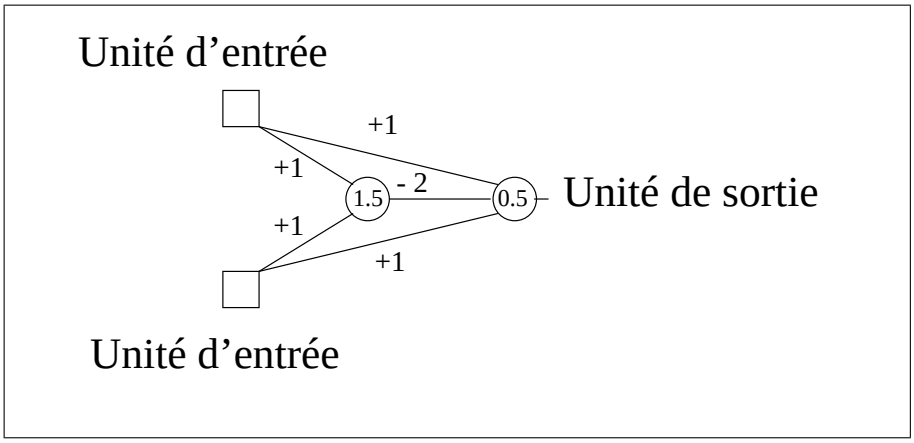


FIG. 7.1 – Le réseau de neurones artificiels représenté ici est une porte XOR "OU exclusif" : la porte ne s'ouvre que si on présente un 1 à une unité d'entrée en même temps qu'un 0 à l'autre unité. Les nombres placés près des connexions représentent les poids, et ceux situés à l'intérieur des unités représentent les valeurs des seuils.

Quand on présente un 1 à une unité d'entrée et un 0 à l'autre unité, l'entrée de l'unité cachée sera alors $(1\times1)+(0\times1)=1$. Cette valeur étant inférieure au seuil, l'unité cachée ne va pas décharger (c'est-à-dire que sa sortie sera égale à 0). Pour l'unité de sortie nous aurons $(1\times1)+(0\times(-2))+(0\times1)=1$, valeur supérieure au seuil (0.5). Le neurone de sortie va donc être actif.

Vous pouvez vérifier que les trois autres possibilités d'entrées conduisent bien aux sorties qui correspondent au fonctionnement d'une porte XOR.

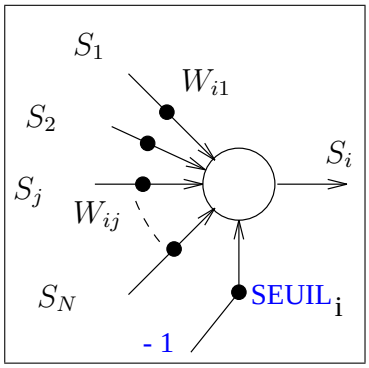
La première simulation va consister à modéliser une porte XOR à l'aide d'un réseau de neurones artificiels. Il est intéressant de savoir qu'en choisissant soigneusement les poids et les seuils d'un réseau simple vous pouvez modéliser n'importe quelle fonction logique. Mais ce qui est excitant avec de tels réseaux c'est que vous n'avez, en fait, pas besoin de choisir les poids et les seuils à la main. Pour notre porte XOR, nous aurions pu partir de n'importe quels poids ou seuils *a priori*.

En présentant alors, de façon répétitive, les motifs d'entrée et de sortie désirée correspondants, le réseau aurait pu apprendre les poids nécessaires à la réalisation d'un XOR. Plus intéressant encore, le réseau peut généraliser ce qu'il a appris. Dans des grands ensembles de données, il peut reconnaître des motifs qu'il n'a jamais vu auparavant.

Pour créer de tels réseaux nous devons apporter quelques modifications à nos neurones artificiels. Tout d'abord les mathématiques utilisées dans l' apprentissage deviennent plus faciles si nous n'avons pas à changer à la fois les poids et les seuils durant l'entraînement du réseau. C'est simple à réaliser. Chaque unité ayant un seuil T et n liaisons entrantes est remplacée par une unité ayant un seuil égal à 0 et $n+1$ liaisons. La liaison supplémentaire ainsi crée est munie d'un poids valant $-T$ et est alors connectée à une cellule qui décharge tout le temps (dont la sortie vaut toujours 1). Ce truc est appelé *polarisation* ; le neurone qui est toujours à 1 est quelquefois appelé *neurone de bias*. Les réseaux de neurones artificiels introduisent souvent cette unité de polarisation qui est habituellement reliée à toutes les unités du réseau pour transformer les seuils en poids. Les mathématiques sont aussi simplifiées lorsque la sortie d'une unité est une fonction sigmoïdienne de l'entrée totale. La sigmoïde est juste une approximation douce de la fonction seuil.

Bien qu'il existe plusieurs façons d'entraîner un réseau, j'ai choisi une méthode provenant d'une classe particulière d'algorithmes appelée *apprentissage supervisé* (il en existe d'autres types comme l'apprentissage non supervisé ou par renforcement). Dans l'apprentissage supervisé, les poids du réseau sont ajustés de façon à ce que les sorties se rapprochent des sorties désirées. La méthode d'apprentissage qui a obtenu le plus de succès est l'algorithme de la *rétro-propagation du gradient stochastique*, c'est celle que nous avons vu en cours et que je rappelle ci-dessous.

7.2 RAPPELS



L'entrée totale et la sortie du neurone i valent respectivement :
$a_i = \sum_{j=0}^{N} W_{ij} s_j - SEUIL_i$
$s_i = f(a_i)$
(Notez l'ordre des indices).

FIG. 7.2 – Un neurone effectue une transformation non linéaire, f, de la somme pondérée des signaux arrivant sur ses entrées. f peut être la fonction Signe, de Heaviside ou une sigmoïde.

Un réseau apprend en résolvant plusieurs fois un même problème, par approximations successives, en minimisant l'erreur à chaque itération. La fonction d'erreur, ou *fonction de coût*, la plus communément utilisée est la somme du carré des erreurs des unités de sortie :

$$C = \frac{1}{2} \sum_{i \in sortie} (s_i - d_i)^2 \tag{7.1}$$

où d_i est la sortie désirée de la cellule i. Il s'agit de l'erreur faite par le réseau lors de la présentation d'un seul motif.

La sortie effective est $s_i = f(a_i)$, où f est une fonction sigmoïde ; par exemple :

$$f(x) = \frac{1}{1 + \exp^{-x}} \tag{7.2}$$

Pour minimiser l'erreur il faut prendre la dérivée de l'erreur par rapport aux w_{ij}, le poids de la connexion allant de l'unité j vers l'unité i. Nous aurons donc à calculer :

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_i} \cdot \frac{\partial a_i}{\partial w_{ij}} = s_j \cdot \frac{\partial C}{\partial a_i} = s_j \cdot Y_i \tag{7.3}$$

L'expression de Y_i pour les cellules de sortie est donnée par :

$$Y_i = (s_i - d_i) \cdot f'(a_i) \tag{7.4}$$

et pour les cellules cachées :

$$Y_i = \sum_k \frac{\partial C}{\partial a_k} \cdot \frac{\partial a_k}{\partial s_i} \cdot \frac{\partial s_i}{\partial a_i} = f'(a_i) \cdot \sum_k Y_k \cdot w_{ki} \tag{7.5}$$

Remarque : avec l'expression de f donnée plus haut nous avons : $f' = f \cdot (1 - f)$.

Comme vous pouvez le voir, il est facile de calculer l'erreur pour les liens qui vont vers les unités de sortie. Mais pour les unités cachées le gradient partiel Y dépend de toutes les unités situées dans la couche après elles (La somme selon k est effectuée sur les unités situées dans la couche supérieure). On voit que la valeur de Y doit être rétro-propagée au travers du réseau pour calculer toutes les dérivées ; d'où le nom donné à la procédure : *rétro-propagation du gradient*.

On a donc :

$$\boxed{\begin{array}{c} w_{ij}^{t+1} \leftarrow w_{ij}^t - \lambda \cdot \frac{\partial C}{\partial w_{Ij}} \\ w_{ij}^{t+1} \leftarrow w_{ij}^t - \lambda \cdot s_j \cdot Y_i \end{array}} \tag{7.6}$$

7.3 Application au XOR

Prendre un réseau ayant 2 entrées, 2 neurones cachés, un neurone de sortie.

Quels sont les vecteurs d'**entrée** et de **sortie** qu'il faut apprendre ?

Que vaut-il mieux prendre comme représentation des états **actif** et **inactif** des neurones ?

Que choisissez-vous comme fonction de transfert f ?

Utilisant les équations décrites plus haut, nous pouvons poser l'algorithme de la manière suivante :

- Choisir un pas λ (entre 0.1 et 0.001)
- Tirer au hasard la valeur des poids (par exemple entre -1 et +1)
- Choisir un coût total minimal (cout_min = 0.01 par exemple)
  - Jusqu'à ce que le réseau ait appris, faire :
    compteur = compteur + 1
  - Pour chaque motif d'entrée
    - ⋆ Faire une passe en avant au travers le réseau pour calculer le motif de sortie
    - ⋆ calculer le coût partiel.
    - ⋆ Pour toutes les unités de sortie calculer les gradients Y.

- ⋆ Pour les unités cachées calculer les gradients en utilisant les résultats du calcul obtenu à la couche précédente.
- ⋆ Pour tous les poids du réseau, changer les poids d'une quantité : $\Delta w_{ij}^t = -\lambda \cdot s_j \cdot Y_i$

◦ Calculer et affichez le coût total

1. Faites une première simulation sans vous soucier de la beauté du programme. Vous pouvez même donner à chacune des connexions (il y en a peu ; COMBIEN ?) un nom différent et dupliquer, pour chacun des motifs, le morceau de programme correspondant.

   Noter la valeur du compteur et recommencez en changeant le pas de calcul. Qu'observez-vous ?
2. Ces essais vous ont permis de "voir" comment fonctionne le réseau.
3. Lisez la suite avant de :
   reprogrammer le réseau en utilisant pointeurs, tableaux, structures. . .
4. Faire varier systématiquement le pas de calcul pour de mêmes conditions initiales de poids et tracer la courbe représentant le temps de convergence en fonction du pas.
   Observez l'évolution du coût total.
5. Pour un pas donné, faire varier les conditions initiales de poids et tracer l'histogramme représentant le nombre de fois où le temps de convergence a atteint une certaine valeur.
6. Représenter, pour un cas particulier (qui converge), les droites séparatrices correspondant aux deux neurones cachés, dans l'espace des entrées.
7. On utilise souvent la formule suivante au lieu de la descente de gradient vue en cours et utilisée plus haut (Equation 7.6)

$$w_{ij}^{t+1} \leftarrow w_{ij}^t - \lambda \cdot \frac{\partial C}{\partial w_{Ij}} + \mu \cdot \Delta w_{ij}^{t-1} \tag{7.7}$$

   La procédure est très efficace.

   On appelle μ le coefficient d'inertie. Pouvez-vous justifier ce terme ?

   Une valeur convenable pour ce coefficient est 0.9.
8. Vous pouvez recommencer les exercices ci-dessus en utilisant cette dernière procédure à moins que vous ne l'utilisiez directement si vous avez prévu, dans votre programme, de stocker la variation des poids pour chaque itération.

FIN[1] !

7.4 Un encodeur 4–2–4

Une partie importante du texte qui suit est tirée d'une traduction que j'ai effectuée pour le No 180 de novembre 1992 de la revue "Pour la Science". L'auteur du texte original s'appelle Drew Van Camp, un informaticien de Toronto.

Pour ceux qui ne seraient pas rassasiés, je lui laisse la parole :

La première implémentation que je vous propose est un réseau auto-encodeur qui comprime les données. Considérons par exemple un réseau possédant quatre unités d'entrée, deux unités cachées

[1] Si vous arrivez jusque là (car il faut compter une dizaine d'heures) vous aurez énormément appris sur les réseaux de neurones ! !

et quatre unités de sortie (un encodeur 4–2–4). Quelle que soit l'entrée présentée au réseau, les quatre entrées doivent se combiner dans les deux unités cachées. Si le codage est bien fait, ces dernières doivent être capable de reproduire le motif original des quatre valeurs sur les unités de sortie. Le motif d'entrée à quatre composantes est ainsi codé par les deux unités cachées.

Remarquez que dans l'encodeur 4–2–4 de la figure 7.3, chaque unité d'entrée est connectée à toutes les unités cachées et que chaque unité cachée est connectée à toutes les unités de sortie. L'unité de polarisation est reliée à toutes les unités cachées et de sortie (il n'est pas nécessaire de relier l'unité cachée aux unités d'entrée puisque ces dernières doivent seulement reproduire les entrées). Les quatre motifs que nous enseignerons au réseau sont des groupes de quatre chiffres, dont un seul est non nul. Un encodeur 4–2–4 compresse les données. Le réseau redonne les entrées en les représentant essentiellement comme des codes binaires (on peut le voir grâce aux unités cachées cf : tableau 7.4).

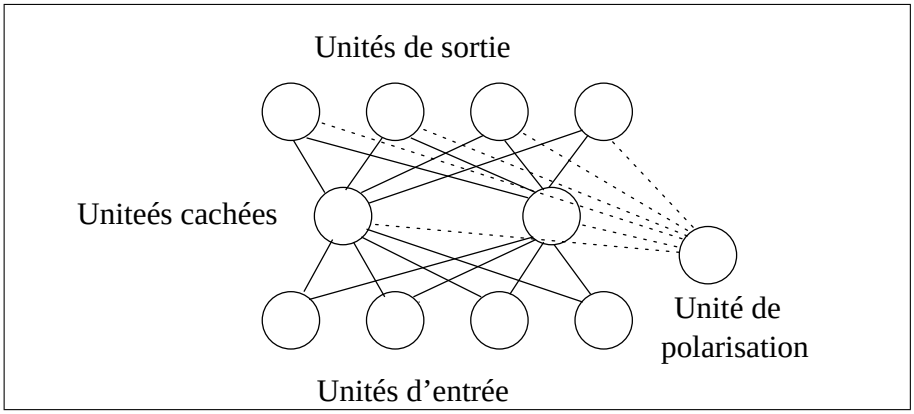


FIG. 7.3 – Un encodeur 4–2–4 compresse les données. Le réseau redonne les entrées en les représentant essentiellement comme des codes binaires (on peut le voir grâce aux valeurs de sortie des unités cachées.

Activités des unités d'entrée				Activités des unités cachées		Activités des unités de sortie			
1	0	0	0	0.03	0.097	0.91	0.10	0.00	0.07
0	1	0	0	0.98	0.96	0.07	0.88	0.06	0.00
0	0	1	0	0.91	0.02	0.00	0.10	0.91	0.06
0	0	0	1	0.03	0.07	0.07	0.00	0.09	0.90

TAB. 7.1 – Activités des différentes cellules en fonction des entrées

Avant de commencer l'apprentissage, fixons les poids du réseau à des valeurs aléatoires comprises entre -1 et 1. Lorsque je présente les quatre motifs au réseau non entraîné, l'erreur totale (la valeur de C) varie entre trois et cinq (dépendant des poids initiaux).

L'entraînement consiste à présenter répétitivement les motifs au réseau et à ajuster les poids après chaque présentation. Nous déciderons que le réseau a atteint des performances acceptables lorsque l'erreur totale pour les quatre motifs est inférieure à 0,1. En utilisant ce critère, il faut entre 800 et 2000 présentations pour que le réseau apprenne la tâche, selon le pas de calcul l. De fait, une des principales difficultés de l'apprentissage réside dans le choix de l. Si la valeur choisie est trop faible, le réseau mettra longtemps à converger. Si la valeur est trop forte le réseau peut devenir

instable et ne jamais converger. Après des essais et des erreurs vous observerez que la valeur de 0,5 n'est pas mauvaise. Si la convergence n'est pas très rapide, le réseau n'explose néanmoins jamais.

Quand le réseau est entraîné, comment code-t-il les motifs présentés ? Faites l'essai et vous verrez que les unités cachées contiennent des valeurs comme 0,03, 0,97, 0,98 et 0,07. Pour l'essentiel le réseau a développé un codage binaire des motifs d'entrée.

Finalement, il est possible de construire un réseau pouvant apprendre l'arithmétique - plus précisément, l'addition de deux nombres binaires codés sur trois bits. Les six bits spécifiant les deux nombres à additionner seront les entrées du réseau. La sortie aura quatre bits (nécessaires en cas de retenue 111 + 111 = 1110).

Avant de construire ce réseau, je vous propose d'écrire un petit programme pour générer l'ensemble des 64 motifs d'apprentissage ainsi que les sorties correspondantes. Déterminons maintenant l'architecture du réseau de neurones. Cette fois encore, je vous propose de n'utiliser qu'une seule couche cachée mais quel nombre de cellules doit-elle contenir ? La question se pose, car un problème survient fréquemment durant l'entraînement des réseaux de neurones artificiels : leurs performances sur l'ensemble d'apprentissage ne cessent de progresser (s'il existe une procédure stable d'actualisation des poids), mais si on leur donne des motifs qui n'appartiennent pas à cet ensemble, leurs performances, après avoir augmenté dans un premier temps, diminuent ensuite.

Ce processus est appelé sur-apprentissage ou encore apprentissage "par coeur". Il survient après que le réseau a appris quelques règles générales concernant les données. Plus l'entraînement se poursuit, plus le réseau apprend les anomalies de l'ensemble d'apprentissage. Il essaye de généraliser ces anomalies aux autres données de sorte qu'une erreur plus importante apparaît.

Pour éviter ce sur-apprentissage, le nombre de poids du réseau doit être inférieur au nombre de bits nécessaires pour spécifier toutes les sorties désirées de l'ensemble d'apprentissage. Pour la tâche d'addition binaire, il doit y avoir beaucoup moins que 256 poids (64 motifs multipliés par les 4 bits de la sortie). En appliquant cette règle approximative j'ai testé un réseau à quinze unités cachées, six unités d'entrée, quatre unités de sortie et une cellule de polarisation, ce qui donne 169 connexions. L'apprentissage est plus long que pour le réseau autocodeur examiné précédemment. En fait il prend plus que 30 000 itérations.

Quand vous serez persuadés que le réseau peut "apprendre", vous pourrez tester le rôle des données d'apprentissage en supprimant quatre motifs de l'ensemble d'apprentissage par exemple. Repartez de poids aléatoires et entraînez à nouveau le réseau avec les 60 motifs restants. Une fois l'apprentissage effectué, proposez au réseau les quatre motifs qu'il n'a jamais rencontrés et examinez sa sortie : Il affiche les bonnes réponses pour ces motifs ! Autrement dit, le réseau a appris à faire des additions binaires.

De nombreuses modifications peuvent augmenter notablement la vitesse d'apprentissage. En particulier, vous pouvez choisir des méthodes plus complexes pour choisir la direction dans laquelle vous allez modifier les poids ou faire des calculs en ligne pour adapter le pas de calcul. Il existe aussi, et vous pouvez les essayer, des algorithmes plus complexes comme par exemple des algorithmes constructifs qui ajoutent des unités cachées en cours d'apprentissage.

Bibliographie