Module 5
Réseaux de neurones

Sommaire

Dernière mise à jour le 28 mars 2022

5.1 Introduction et définitions

Dans le cerveau, les neurones sont reliés entre eux par l'intermédiaire d'axones et de dendrites. Les dendrites représentent les entrées du neurone et son axone sa sortie. Un neurone émet un signal en fonction des signaux qui lui proviennent des autres neurones. Une intégration des signaux reçus au cours du temps, c'est-à-dire une sommation des signaux, a lieu au niveau du neurone. En général, quand la somme dépasse un certain seuil, le neurone émet à son tour un signal électrique.

Figure 5.1: Neurone formel.

Un **réseau de neurones** aussi appelé **réseau de neurones artificiels** est un modèle mathématique très simple dérivé du neurone biologique dont les composantes élémentaires sont des unités de calcul qui reçoivent des entrées pour produire des sorties. Ce modèle reprend les principes du fonctionnement du neurone biologique, en particulier par la sommation des entrées et la pondération de la somme de ses entrées par des **poids synaptiques** (aussi appelés coefficients synaptiques) tels qu'illustrés à la figure 5.2.

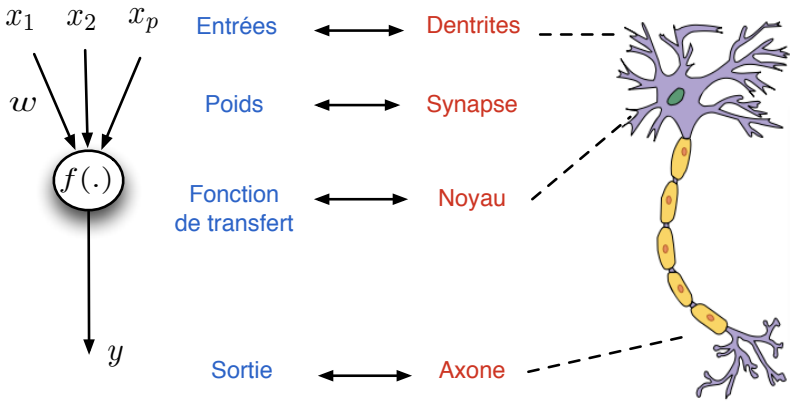


Figure 5.2: Analogie entre le neurone biologique et le neurone artificiel.

Un réseau de neurones est défini par sa structure (c'est-à-dire le nombre de couches, le

nombre de noeuds par couche et le nombre de sorties) et par la règle d'apprentissage qui permet de calculer les poids synaptiques.

La règle d'apprentissage dépend du type d'apprentissage comme nous l'avons introduit dans le module 1 - Introduction à l'apprentissage machine. Nous en faisons ici un rappel tout en utilisant la terminologie spécifique aux réseaux de neurones. Dans le cas de l'apprentissage **supervisé**, le réseau de neurones est entraîné en se basant sur des échantillons d'entrées dont la sortie correspondante est connue (La sortie étant utilisée pour les données d'entrainement unique). Autrement dit, la classe des données d'apprentissage est connue. Dans le cas de l'**apprentissage non supervisé**, nous disposons d'un ensemble de données d'apprentissage et nous cherchons à les regrouper selon des critères de ressemblance qui sont inconnus *a priori*. Ce type d'apprentissage est utilisé dans un but de visualisation ou d'analyse de données.

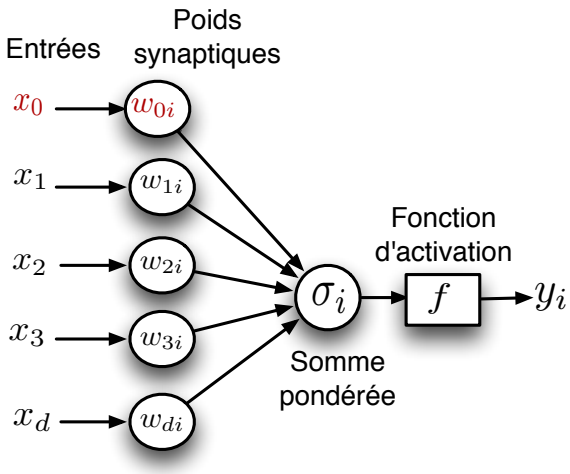


Figure 5.3: Neurone formel.

5.2 Le perceptron

5.2.1 Structure

Le perceptron a été inventé par Rosenblatt en 1957. Il est considéré comme le type de réseau de neurones le plus simple. Dans sa version simplifiée, le perceptron fournit une sortie suite à la pondération de la somme de ses entrées et l'application d'une fonction d'activation (Figure 5.3).

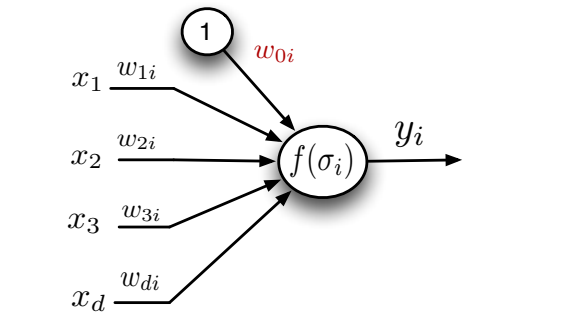


Figure 5.4: Schéma d'un perceptron.

Soit un neurone i qui reçoit les entrées $x_1, x_2, .., x_d$ pondérées par les coefficients synaptiques $w_{1i}, w_{2i}, ..., w_{di}$ (Figure 5.4). d étant la dimension du vecteur d'entrée. Le neurone reçoit également une entrée unitaire $x_0 = 1$ dont le poids synaptique est $w_{0i} = b$. Cette entrée, appelée biais b, sert à contrôler le seuil d'activation. La sortie y_i de la fonction

d'activation est donnée par la relation :

$$y_i = f(\sigma_i) = f\Big(\sum_{j=0}^{d} w_{ji} x_i\Big) = f(\mathbf{w}.\mathbf{x}) \tag{5.1}$$

$\mathbf{w}$ et $\mathbf{x}$ sont respectivement les vecteurs de poids synaptiques et de l'entrée du réseau de neurones. L'opérateur (.) désigne le produit scalaire.
f est une **fonction d'activation** comme décrite dans ce qui suit.

5.2.2 Fonction d'activation

La fonction d'activation (aussi appelée *fonction de transfert*) est, généralement, choisie parmi les fonctions suivantes :

La fonction à seuil

La fonction à seuil aussi appelée *Heaviside* de seuil $\theta \in \mathbb{R}$ est définie par :

$$\begin{cases} f(x) = 0 & si \quad x < \theta \\ f(x) = 1 & si \quad x \geq \theta \end{cases} \tag{5.2}$$

Autrement dit, la fonction *Heaviside* $f(x)$ prend la valeur 0 si $x < \theta$ sinon elle prend la valeur 1.

La fonction linéaire

La fonction linéaire est définie par :

$$\begin{cases} f(x) = 0 & si \quad x < \theta_1 \\ f(x) = 1 & si \quad x \geq \theta_2 \\ f(x) = x & si \quad \theta_1 < x \leq \theta_2 \end{cases} \tag{5.3}$$

Les seuils θ_1 et $\theta_2 \in \mathbb{R}$ sont définis *à priori*.

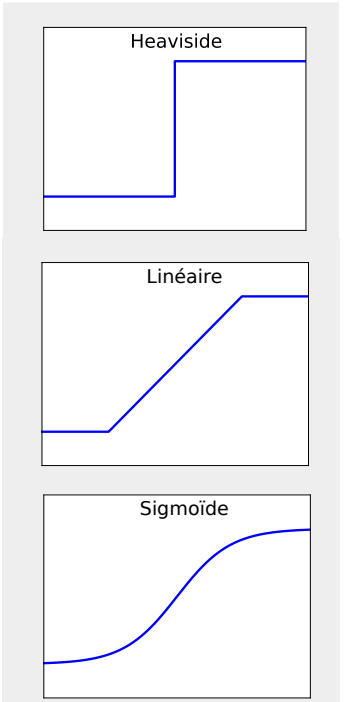


Figure 5.5: Fonctions d'activation.

La fonction sigmoïde

La fonction sigmoïde est définie par :

$$f(x) = \frac{1}{1 + \exp(-\lambda x)} \quad \text{avec} \quad \lambda \in \mathbb{R}^+$$

Alors que la fonction à seuil renvoie la valeur 0 ou la valeur 1, la fonction sigmoïde renvoie une valeur entre 0 et 1. La fonction sigmoïde présente, en plus, l'avantage d'être dérivable.

▶ **Exemple 5.1** L'objectif, ici, est de donner un exemple d'application du perceptron simple. Nous disposons d'un système d'aide à la décision qui permet de décider si un véhicule peut traverser ou pas un pont. Soit $y = 0$ pour *véhicule peut traverser* (ou encore passage non interdit) et $y = 1$ pour *véhicule ne peut pas traverser* (ou encore passage interdit).

Le système est basé sur un perceptron simple dont la fonction d'activation est une fonction à seuil avec $\theta = 0$ et dont les poids synaptiques sont :

$$\begin{cases} w_0 = b = -47 \\ w_1 = 2 \\ w_2 = 1 \end{cases}$$

Soit un ensemble de 8 données d'entrées correspondant chacune à un véhicule. Chaque entrée est décrite par un vecteur de caractéristiques à deux dimensions $\mathbf{x} = [x_1, x_2]$: x_1 correspond à la longueur du véhicule en mètre et x_2 le niveau du bruit engendré par son moteur en dB.

Nous désirons utiliser ces deux caractéristiques et le perceptron précédemment décrit pour décider si un véhicule peut ou non traverser ce pont. Nous disposons, également, de la classe y_j de chacune des entrées ($j = 0$ à 8) avec $y_j = 0$ pour *véhicule peut traverser* et $y_j = 1$ pour *véhicule ne peut pas traverser*. Ceci nous permettra de comparer, pour chaque entrée, la classe prédite en utilisant le perceptron $a(\mathbf{x}_j)$ à la classe réelle du véhicule y_j.

Le tableau suivant résume les sorties du réseau de neurones en utilisant en utilisant la relation de l'équation suivante :

$$a(\mathbf{x}_j) = f\Big(b + \sum_{i=1}^{p} w_i x_i\Big) \tag{5.4}$$

Véhicule $\mathbf{x}_j$	x_1	x_2	$b + w_1x_1 + w_2x_2$	$a(\mathbf{x}_j)$
Camion 1	20	8	$-47 + 2 \times 20 + 1 \times 8 = 1$	1
Camion 1	15	20	$-47 + 2 \times 15 + 1 \times 20 = 3$	1
Bus 1	16	10	$-47 + 2 \times 16 + 1 \times 10 = -5$	0
Bus 2	15	5	$-47 + 2 \times 15 + 1 \times 5 = -12$	0
Voiture 1	5	15	$-47 + 2 \times 5 + 1 \times 15 = -22$	0
Voiture 2	16	6	$-47 + 2 \times 16 + 1 \times 6 = -9$	0
Moto 1	2	20	$-47 + 2 \times 2 + 1 \times 20 = -23$	0
Moto 2	6	16	$-47 + 2 \times 6 + 1 \times 16 = -19$	0

Véhicule ($\mathbf{x}_j$)	x_1	x_2	y_j
Camion 1	20	8	1
Camion 1	15	20	1
Bus 1	16	10	0
Bus 2	15	5	0
Voiture 1	5	15	0
Voiture 2	16	6	1
Moto 1	2	20	0
Moto 2	6	16	0

Le calcul des sorties des fonctions d'activation permet de vérifier que le perceptron permet de bien prédire la classe des véhicules (classifier les classes). Nous remarquons que, pour chaque véhicule, la sortie calculée $a(\mathbf{x}_j)$ correspond bien à la classe désirée y_j sauf pour le véhicule Voiture 2 ou la sortie calculée $a(\mathbf{x})$ est différente de la classe désirée y_j. Il s'agit dans ce cas d'une erreur de décision du perceptron.

5.2.3 Apprentissage du perceptron simple

L'apprentissage du perceptron simple a pour objectif la détermination des valeurs des poids synaptiques qui permettent de fournir la sortie (classe) y_i à partir des données d'apprentissage $\mathbf{x}_i$. Autrement dit, il s'agit de déterminer les valeurs des poids synaptiques à partir des paires $(\mathbf{x}_i, y_i) \in D$.

L'algorithme opère de façon itérative pendant n_{max} itérations. Il permet de calculer la prédiction $a(\mathbf{x}_i)$ de l'entrée $\mathbf{x}_i$ et de la comparer à la classe réelle y_i. La mise à jour des

poids est appelée **règle d'apprentissage** du perceptron. Si $y_i \neq a(\mathbf{x}_i)$, les éléments du vecteur poids $\mathbf{w}$ sont mis à jour en fonction de la différence entre les valeurs de y_i et de $a(\mathbf{x}_i)$. Cette règle de mise à jour est appelée règle Delta (Δ). La constante η est appelée **pas d'apprentissage** ou taux d'apprentissage ; elle est souvent fixée à la valeur 0.1. Le critère d'arrêt peut être le nombre maximal d'itérations ou bien une valeur d'erreur fixée au préalable. Dans l'algorithme d'apprentissage du perceptron suivant, nous avons considéré le nombre maximal d'itérations t_{max} comme critère d'arrêt.

Algorithme d'apprentissage du perceptron.

1. Initialiser aléatoirement les poids synaptiques
2. $t = 0$
3. **Tant que** $t \leq t_{max}$ **Faire**
4. Tirer un échantillon $(\mathbf{x}_i, y_i) \in D$
5. **Pour** $j = 1$ à d **Faire**
6. $\Delta_j \leftarrow \eta\ x_j(y_j - a(\mathbf{x}_i))$
7. $w_j \leftarrow w_j + \Delta_j$
8. **Fin pour**
9. $t \leftarrow t + 1$
6. **Fin tant que**

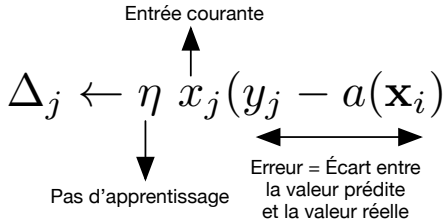


Figure 5.6: Règle de mise à jour

▶ **Exemple 5.2** L'objectif de cet exemple est de simuler à l'aide d'un perceptron la fonction logique `ET` (`AND`) qui prend en entrée deux valeurs binaires (0 ou 1) et qui retourne 1 si les deux entrées valent 1, sinon elle retourne 0. La fonction d'activation considérée est une fonction à seuil avec $\theta = 0$. L'algorithme d'apprentissage sera exécuté pour $t_{max} = 4$, c'est-à-dire pour 4 itérations.

Nous commençons par initialiser aléatoirement les poids synaptiques. Soit :

$$\begin{cases} w_0 = b = 0.1 \\ w_1 = 0.2 \\ w_2 = 0.05 \end{cases}$$

Ceci veut dire qu'initialement la frontière de décision est décrite par l'équation :

$$0.1 + 0.2x_1 + 0.05x_2 = 0 \iff x_2 = -4x_1 - 2$$

La constante d'apprentissage η étant fixée à 0.1.
Nous considérons la première observation $\mathbf{x}_1 = [0, 0]$ dont la sortie $y_1 = 0$,

$$\begin{cases} x_1 = 0 \\ x_2 = 0 \\ y_1 = 0 \end{cases}$$

Selon l'équation 5.4, $a(\mathbf{x}_1) = f(0.1 + 0.2 \times 0 + 0.05 \times 0) = f(0.1) = 1$

Nous procédons, ensuite, à une mise à jour des poids selon :

$$\begin{cases} w_0 = 0.1 + 0.1 \times 1 \times (0 - 1) = 0 \\ w_1 = 0.2 + 0.1 \times 0 \times (0 - 1) = 0.2 \\ w_2 = 0.05 + 0.1 \times 0 \times (0 - 1) = 0.05 \end{cases}$$

Nous considérons, ensuite, la deuxième observation $\mathbf{x}_2 = [0, 1]$ dont la sortie $y_2 = 0$,

$$\begin{cases} x_1 = 0 \\ x_2 = 1 \\ y_2 = 0 \end{cases}$$

On a,

$$a(\mathbf{x}_2) = f(0 + 0.2 \times 0 + 0.05 \times 1) = f(0.05) = 1$$

Nous procédons à une mise à jour. Nous obtenons :

$$\begin{cases} w_0 = 0 + 0.1 \times 1 \times (0 - 1) = -0.1 \\ w_1 = 0.2 + 0.1 \times 0 \times (0 - 1) = 0.2 \\ w_2 = 0.05 + 0.1 \times 1 \times (0 - 1) = -0.05 \end{cases}$$

$\mathbf{x}_i$	x_1	x_2	y_i
$\mathbf{x}_1$	0	0	0
$\mathbf{x}_2$	0	1	0
$\mathbf{x}_3$	1	0	0
$\mathbf{x}_4$	1	1	1

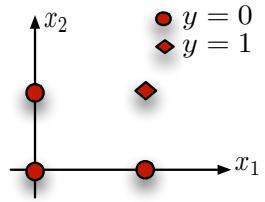


Figure 5.7: Fonction logique ET.

Nous considérons, la troisième observation à traiter $\mathbf{x}_3 = [1, 0]$ dont la sortie $y_3 = 0$,

$$\begin{cases} x_1 = 1 \\ x_2 = 0 \\ y_3 = 0 \end{cases}$$

$$a(\mathbf{x}_3) = f(-0,1 + 0.2 \times 1 - 0.05 \times 0) = f(0.1) = 1$$

Nous procédons, ensuite, à une mise à jour selon :

$$\begin{cases} w_0 = -0.1 + 0.1 \times 1 \times (0 - 1) = -0.2 \\ w_1 = 0.2 + 0.1 \times 1 \times (0 - 1) = 0.1 \\ w_2 = -0.05 + 0.1 \times 0 \times (0 - 1) = -0.05 \end{cases}$$

Nous considérons, la quatrième observation à traiter $\mathbf{x}_4 = [1, 1]$ dont la sortie $y_4 = 1$,

$$\begin{cases} x_1 = 1 \\ x_2 = 1 \\ y_4 = 1 \end{cases}$$

$$a(\mathbf{x}_4) = f(-0.2 + 0.1 \times 1 - 0.05 \times 1) = f(-0.15) = 0$$

Nous procédons à une mise à jour selon :

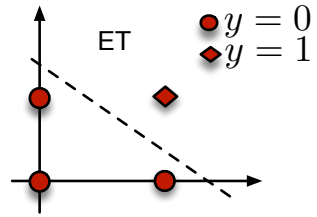
$$\begin{cases} w_0 = -0.2 + 0.1 \times 1 \times (1 - 0) = -0.1 \\ w_1 = 0.1 + 0.1 \times 1 \times (1 - 0) = 0.2 \\ w_2 = -0.05 + 0.1 \times 1 \times (1 - 0) = 0.05 \end{cases}$$

La frontière de décision est alors décrite par l'équation : $-0.1 + 0.2x_1 + 0.05x_2 = 0$

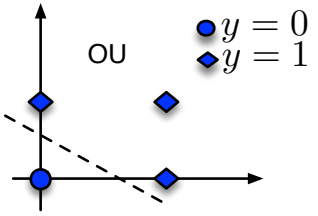
Limitations du perceptron

Le perceptron est incapable de distinguer les données non séparables linéairement. En effet, il ne peut construire qu'une frontière linéaire comme pour la fonction `ET` et `OU` logique (Figure 5.8). Or, la majorité des problèmes de classification ne sont pas linéairement séparables, d'où le besoin de déterminer des modèles plus complexes qui permettent de modéliser des frontières plus complexes. C'est le cas de la fonction `XOR` logique qui en dépit de sa simplicité ne peut être modélisée par une frontière linéaire (Figure 5.8).

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

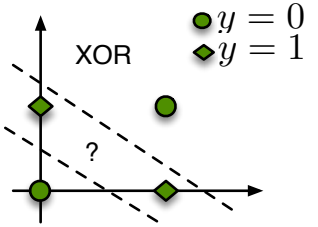


Figure 5.8: Fonctions logiques.

5.3 Le perceptron multicouche (PMC)

5.3.1 Structure

Comme son nom l'indique, le PMC est une succession de neurones organisés en plusieurs couches. La première couche est reliée aux entrées. Ensuite, on retrouve un ensemble de couches intermédiaires qui ne sont pas visibles et qui sont appelées des **couches cachées**. Chaque couche est reliée à la couche précédente. La dernière couche du PMC définit la réponse des neurones et produit les différentes sorties (Figure 5.9).
Soit $w_{i,j}$ (aussi noté parfois $w(i,j)$ ou w_{ij}) le poids de la connexion menant de l'entrée d'indice i à la sortie d'indice j. La règle de propagation qui mène à la transmission de l'information dans un réseau multicouche se fait de couche en couche en calculant la somme des entrées. Ces entrées, notées a_i correspondent à x_i à la couche d'entrée ensuite elles correspondent à $f(\sigma_j)$

$$\sigma_j = \sum_i w_{i,j} y_i \tag{5.5}$$

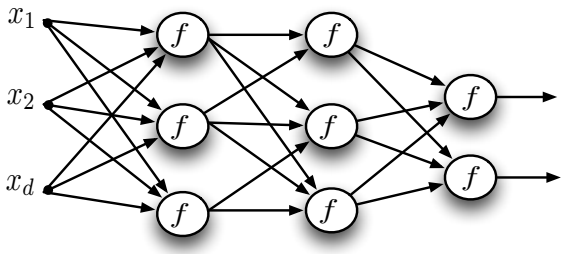


Figure 5.9: Architecture d'un perceptron multicouche.

▶ **Exemple 5.3** L'objectif de cet exemple est de simuler la règle de propagation dans un perceptron multicouche. Considérons le PMC à deux couches de la figure 5.10. La fonction d'activation considérée est la fonction sigmoïde. Les neurones de la couche d'entrée sont numérotés par x_1 (pour 1) et x_2 (pour 2). Les neurones de la couche cachée sont numérotés par 3 et 4 et le neurone de la couche de sortie par 5. La la fonction d'activation considérée est la fonction sigmoïde et les poids synaptiques étant fixés selon le tableau suivant :

$w_{0,3} = 0.2$	$w_{1,3} = 0.1$	$w_{2,4} = 0.4$
$w_{0,4} = -0.3$	$w_{1,4} = -0.2$	$w_{3,5} = 0.5$
$w_{0,5} = 0.4$	$w_{2,3} = 0.3$	$w_{4,5} = -0.4$

Soit le vecteur d'entrée $\mathbf{x} = [1, 1]$, la transmission de l'information dans chacun de neurone se fait comme suit :

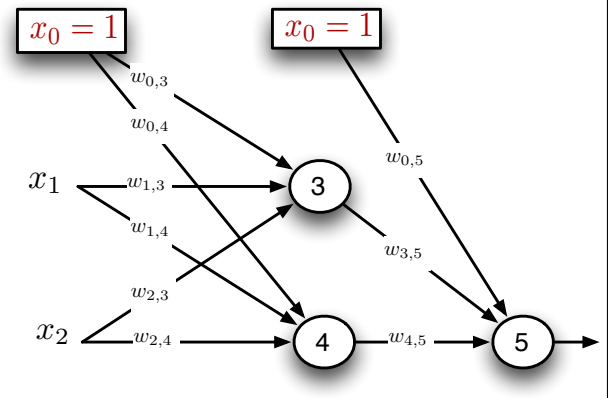


Figure 5.10: Architecture d'un perceptron multicouche.

Le neurone 3 :

$$\sigma_3 = w_{0,3} + \sum_{j=1}^{2} w_{j,3} x_j = w_{0,3} + w_{1,3} \times x_1 + w_{2,3} \times x_2 = 0.1 + 0.2 + 0.3 = 0.6$$

La sortie du neurone 3 est alors :

$$a_3 = f(\sigma_3) = \frac{1}{1 + e^{-0.6}} = 0.65$$

Le neurone 4 :

$$\sigma_4 = w_{0,4} + \sum_{j=1}^{2} w_{j,4} y_j = w_{0,4} + w_{1,4} \times x_1 + w_{2,4} \times x_2 = -0.3 - 0.2 \times 1 + 0.4 \times 1 = -0.1$$

La sortie du neurone 3 est alors :

$$a_4 = f(\sigma_4) = \frac{1}{1 + e^{0.1}} = 0.48$$

Le neurone 5 :

Les entrées du neurone 5 sont les sorties des neurones 2 et 3. Il faut donc faire attention aux appellations $\mathbf{x}_i$ et y_i puisque les entrées d'une couche cachée sont ou bien les données d'entrées (souvent notées $\mathbf{x}_i$) ou bien les données de sorties de la couche précédente (souvent notées $\mathbf{y}_i$).

$$\sigma_5 = w_{0,5} + \sum_{j=3}^{4} w_{j,5} x_j = w_{0,5} + w_{3,5} \times y_3 + w_{4,5} \times y_4 = 0.4 + 0.5 \times 0.65 - 0.4 \times 0.48 = 0.53$$

La sortie du neurone 5 est alors :

$$a_5 = f(\sigma_3) = \frac{1}{1 + e^{-0.53}} = 0.63$$

En conclusion, pour une entrée $\mathbf{x} = [1, 1]$, la propagation se fait selon :

$$\begin{cases} a_3 = 0.65 \\ a_4 = 0.48 \\ a_5 = 0.63 \end{cases}$$

5.3.2 Apprentissage par rétropropagation

Comme décrit dans la section 5.2.3, l'apprentissage du perceptron se fait selon la règle Delta. Cette règle a été généralisée en 1986 aux réseaux multicouches et formalisée sous l'appellation de la règle de la rétropropagation du gradient de l'erreur. Cette dernière consiste à propager l'erreur obtenue à un neurone de sortie d'un réseau à couches à travers le réseau par descente du gradient dans le sens inverse de la propagation des activations.

Algorithme de rétropropagation du gradient.

1. Initialiser aléatoirement les poids synaptiques
2. **Répéter**
3. **Pour** chaque $(\mathbf{x}_i, y_i) \in D$
4. **Pour** chaque couche du réseau de neurones
5. **Pour** chaque neurone de la couche considérée
6. Calculer δ_j selon les éq. 5.7 et 5.8
7. **Pour** chaque connexion w_{ij}
8. Calculer $\Delta w_{ij} = \eta \delta_j y_i$
9. **FinPour**
10. **FinPour**
11. **FinPour**
14. **FinPour**
12. **Pour** chaque connexion w_{ij} **Faire**
13. $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$
14. **FinPour**
15. **Jusqu'à** atteinte d'un critère d'arrêt

La règle de la rétropropagation du gradient de l'erreur consiste alors à mettre jour le poids entre le neurone i et le neurone j en utilisant la relation suivante :

$$\Delta w_{i,j} = \eta \delta_j a_i \tag{5.6}$$

Pour les connexions aboutissant aux neurones de sortie l, δ_j est donnée par la relation :

$$\delta_j = a_j(1 - a_j)(y_l - a_j) \tag{5.7}$$

a_j étant la valeur calculée (prédite) au neurone j.
Pour les unités cachées, le calcul de δ_j se fait récursivement par la descente du gradient selon :

$$\delta_j = a_j(1 - a_j) \sum_i \delta_i w_{i,j} \tag{5.8}$$

i est l'indices des neurones de la couche de sortie qui sont connectés au neurone concerné (neurone j)

▶ **Exemple 5.4** Reprenons l'exemple précédent pour appliquer l'algorithme de rétropropagation du gradient de l'erreur en supposant que la sortie désirée soit égale à 0.

Rappelons les valeurs des poids synaptiques :

$w_{0,3} = 0.2$	$w_{1,3} = 0.1$	$w_{2,4} = 0.4$
$w_{0,4} = -0.3$	$w_{1,4} = -0.2$	$w_{3,5} = 0.5$
$w_{0,5} = 0.4$	$w_{2,3} = 0.3$	$w_{4,5} = -0.4$

et les valeurs des sorties des neurones que nous avons obtenus suite à la fonction de propagation :

$$\begin{cases} a_3 = 0.65 \\ a_4 = 0.48 \\ a_5 = 0.63 \end{cases}$$

Le neurone 5 : Pour appliquer l'algorithme de rétropropagation du gradient de l'erreur, nous commençons par le neurone de sortie en appliquant l'Eq. 5.7. Dans notre cas, il s'agit du neurone 5.

$$\delta_5 = a_5(1-a_5)(y_5-a_5) = 0.63 \times (1-0.63) \times (0-0.63) = -0.147$$

d'où

$$\begin{cases} \Delta w_{0,5} = \eta\ \delta_5\ a_0 = 1 \times (-0.147) \times 1 = -0.147 \\ \Delta w_{3,5} = \eta\ \delta_5\ a_3 = 1 \times (-0.147) \times 0.65 = -0.1 \\ \Delta w_{4,5} = \eta\ \delta_5\ a_4 = 1 \times (-0.147) \times 0.48 = -0.07 \end{cases}$$

Le neurone 4 : Le neurone 4 est un neurone caché qui est connecté uniquement au neurone 5.

$$\delta_4 = a_4\ (1-a_4)\ \delta_5\ w_{4,5} = 0.48 \times (1-0.48) \times (-0,147) \times (-0.4) = 0.015$$

d'où

$$\begin{cases} \Delta w_{0,4} = \eta\ \delta_4\ a_0 = 1 \times (0.015) \times 1 = 0.015 \\ \Delta w_{1,4} = \eta\ \delta_4\ a_1 = 1 \times (0.015) \times 1 = 0.015 \\ \Delta w_{2,4} = \eta\ \delta_4\ a_2 = 1 \times (0.015) \times 1 = 0.015 \end{cases}$$

Le neurone 3 : Le neurone 3 est, aussi, un neurone caché qui est connecté uniquement au neurone 5.

$$\delta_3 = a_3\ (1-a_3)\ \delta_5\ w_{3,5} = 0.65 \times (1-0.65) \times (-0,147) \times (0.5) = 0.017$$

d'où

$$\begin{cases} \Delta w_{1,3} = \eta\ \delta_3\ a_1 = 1 \times (0.017) \times 1 = 0.017 \\ \Delta w_{2,3} = \eta\ \delta_3\ a_2 = 1 \times (0.017) \times 1 = 0.017 \\ \Delta w_{0,3} = \eta\ \delta_3\ a_0 = 1 \times (0.017) \times 1 = 0.017 \end{cases}$$

Nous calculons ensuite la valeur des poids synaptiques après les mises à jour tel que décrit à la ligne 13 de l'algorithme de rétropropagation du gradient :

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

Nous obtenons le tableau suivant :

$w_{0,3} = 0.217$	$w_{1,3} = 0.117$	$w_{2,4} = 0.415$
$w_{0,4} = -0.285$	$w_{1,4} = -0.185$	$w_{3,5} = 0.4$
$w_{0,5} = 0.253$	$w_{2,3} = 0.317$	$w_{4,5} = -0.47$

Une fois les poids synaptiques mis à jour, nous recalculons les valeurs des sorties des neurones en appliquant la propagation de gradient (comme nous l'avons fait à l'exemple 5.3). Les valeurs obtenues des y_i sont :
Le neurone 3 :$y_3 = 0.65$
Le neurone 4 :$y_4 = 0.48$
Le neurone 5 :$y_5 = 0.54$

5.4 Le réseau de Kohonen

5.4.1 Architecture

Le fonctionnement du réseau de Kohonen aussi appelée carte auto-organisatrice de Kohonen (*self-organizing map* ou SOM) s'inspire du fonctionnement du neurone biologique. En effet, les neurones du cerveau sont organisés en régions correspondant à des fonctions sensorielles différentes, mais proches. De ce fait, des stimulus proches d'organes sensoriels entraînent l'activation de neurones proches du cortex cérébral, c'est-à-dire que ce type de réseau respecte la notion de voisinage ; d'où provient l'appellation de la carte auto-organisatrice.

Les cartes auto-organisatrice ont été introduites pour la première fois par Kohonen qui cherchait à représenter des données multidimensionnelles dans un espace de faible dimension. Pour y parvenir, Kohonen cherche à partitionner, par entrainement, les données en groupements dont la structure de voisinage peut être représentée dans un espace de faible dimension (1, 2 ou 3D) appelé «carte topologique».

Les cartes de Kohonen trouvent leurs applications dans plusieurs domaines comme la reconnaissance de formes, le traitement d'images, la robotique, l'aide à la décision et l'optimisation.
Le réseau de Kohonen est composé de deux couches : une couche d'entrée et une couche de sortie. Les nœuds de la couche d'entrée sont liés à ceux de la couche de sortie par l'intermédiaire des poids synaptiques (Figure 5.11).

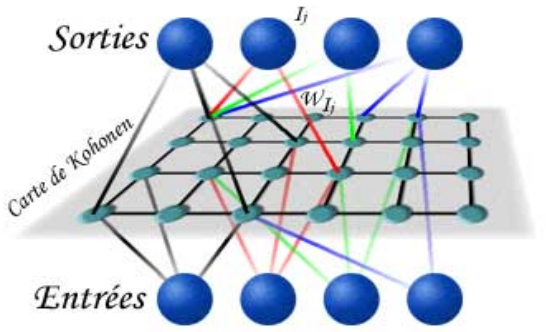


Figure 5.11: Architecture rectangulaire (2D) de la carte auto-organisatrice de Kohonen.

5.4.2 Apprentissage de la carte de Kohonen

L'entrée du réseau est constituée des vecteurs de caractéristique $\mathbf{x}_i = (x_i^1, x_i^2, ..., x_i^K)'$ de dimension K et la sortie est composée de J nœuds contenants des vecteurs de poids $\mathbf{w}_j = (w_j^1, w_j^2, ...w_j^K)$. Les poids $\mathbf{w}_j$ sont initialement de petites valeurs aléatoires.
L'entraînement de la carte de Kohonen peut se faire selon l'algorithme qui suit.

Algorithme d'apprentissage de la mémoire de Kohonen : On cherche à déterminer la valeur des vecteurs de poids synaptiques $\mathbf{w}_j = (w_{1j}, w_{2j}, ...w_{Kj})^T$ du noeud j, $j \in [1, J]$.

1. Initialiser les poids W_j à de petites valeurs aléatoires, $j \in [1, J]$.
2. **Pour** $n = 1 \longrightarrow n_{iter}$ **Faire**
3. **Pour** $i = 1 \longrightarrow I$ **Faire**
4. Présenter une entrée $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{iK})^T$,
5. Calculer la distance $d(X_i, W_j)$, $i \in [1, I]$
6. Sélectionner le noeud j^* le plus proche de l'entrée $\mathbf{x}_i$
7. Mettre à jour les poids de la mémoire selon la relation :
   $w_{ij}(n+1) = w_{ij}(n) + \epsilon_n h_n^{j,j^*}(x_{ik}^n - w_{ij}(n))$
   avec $\epsilon_n = \epsilon_i(\frac{\epsilon_f}{\epsilon_i})^{\frac{n}{n_{max}}}$, $h_n^{j,j^*} = \exp -\frac{||j-j^*||^2}{2\sigma_n^2}$, $\sigma_n = \sigma_i(\frac{\sigma_f}{\sigma_i})^{\frac{n}{n_{max}}}$
8. **FinPour**
9. **FinPour**

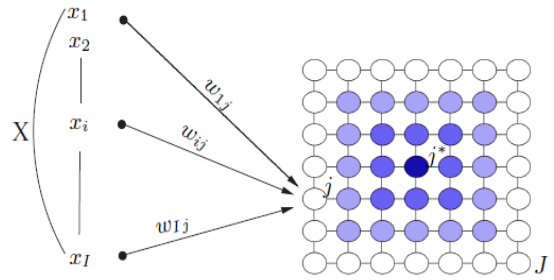


Figure 5.12: Entraînement de la mémoire de Kohonen 2D [2].

L'entraînement consiste à présenter une entrée $\mathbf{x}_i$ parmi les échantillons de la base d'entraînement et à déterminer le noeud gagnant j^* le plus proche de cette entrée en terme de distance minimale. Ensuite, il s'agit de modifier le contenu de ce noeud ainsi que celui des autres noeuds selon la relation :

$$w_{ij}(n+1) = w_{ij}(n) + \epsilon_n h_n^{j,j^*}(x_{ik}^n - w_{ij}(n))$$

La fonction h_n^{j,j^*} définit l'importance de l'influence du noeud j^* sur le noeud j ; elle décroît selon la distance entre les positions de j et j^* dans la grille de sortie. À l'itération n (ligne 2 de l'algorithme), la fonction h_n^{j,j^*} dépend du paramètre σ_n, lequel varie entre σ_i et σ_f au cours de l'apprentissage.

L'entraînement de la carte de Kohonen donne naissance à des regroupements de données qui se forment selon la notion de voisinage. La figure 5.12 montre un vecteur d'entrée $\mathbf{x} = (x_1, x_2, .., x_K)^T$ de dimension K et un espace de sortie qui comprend J neurones. Un vecteur $\mathbf{x}$ est lié aux J nœuds de sortie par l'intermédiaire de J coefficients w_{ij}. Chaque sortie j peut donc être considérée comme porteuse d'un vecteur image $\mathbf{w}_j = (w_{1j}, w_{2j}, ..., w_{Kj})^T$.

5.5 Les réseaux de neurones profonds

Les réseaux de neurones profonds (*Deep Neural Networks*) sont similaires aux réseaux PMC mais avec plus de couches cachées. L'augmentation du nombre de couches, permet à un réseau de neurones de détecter de légères variations du modèle d'apprentissage. Le cours INF1421 ne traite pas ce type de réseaux de neurones vue sa complexité. L'objectif est de vous mettre au courant de ce type de réseaux de neurones qui est devenu la vedette de l'apprentissage machine après plusieurs de controverse.
Actuellement, toutes les grandes entreprises technologiques s'y mettent : Google, IBM, Microsoft, Amazon ou Adobe y investissent des fortunes. Facebook également, qui, signal fort, a placé Yann LeCun à la tête de son nouveau laboratoire d'intelligence artificielle installé à Paris [1].

5.6 Annexes

Dans cette section, nous présentons quelques fonctions R utiles pour la compréhension et l'application des notions étudiées précédemment en utilisant le logiciel R.

— `neuralnet` (*Training of Neural Networks*) permet de mesurer, manipuler, analyser et interpréter les réseaux de neurones. https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf

Références

[1] G. Benicourt. Introduction au deep learning. http://www.benicourt.com/blender/2016/05/introduction-au-deep-learning/. Consulté : 20 Décembre 2016.

[2] Neila Mezghani, Amar Mitiche, and Mohamed Cheriet. A new representation of shape and its use for high performance in online arabic character recognition by an associative memory. *IJDAR*, 7(4) :201–210, 2005.

