FACULTÉ DES SCIENCES

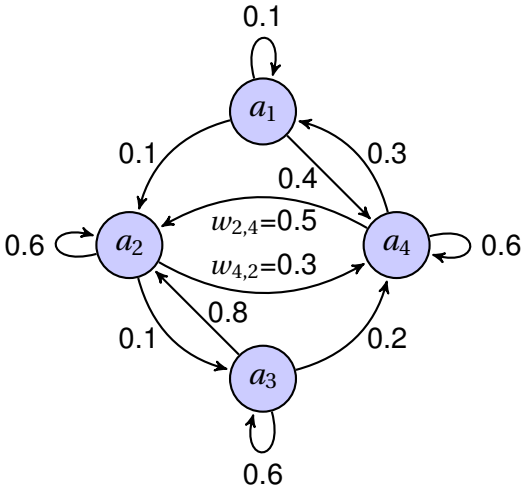
INSTITUT DE STATISTIQUES DE L'UNIVERSITÉ DE PARIS

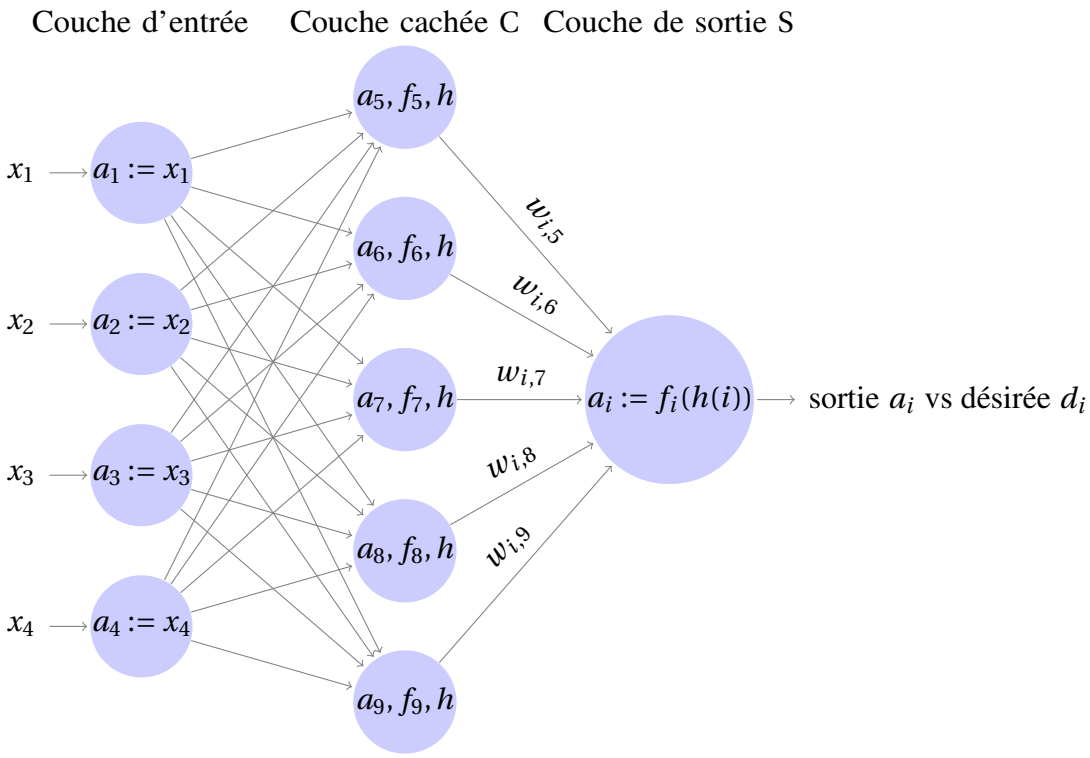
ISUP TROISIÈME ANNÉE
MASTER 2 DE STATISTIQUES - MAJOR DATA SCIENCE

RÉSEAUX DE NEURONES ARTIFICIELS

Professeure :
Annick VALIBOUZE

Année 2018-2019

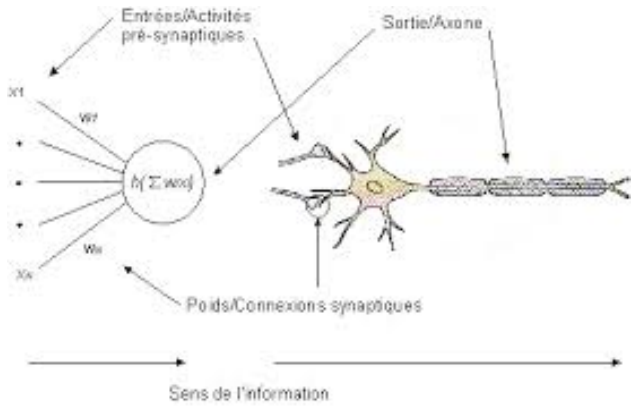



Couche d'entrée
Couche cachée C
Couche de sortie S
x_1
x_2
x_3
x_4
$a_1 := x_1$
$a_2 := x_2$
$a_3 := x_3$
$a_4 := x_4$
a_5, f_5, h
a_6, f_6, h
a_7, f_7, h
a_8, f_8, h
a_9, f_9, h
$w_{i,5}$
$w_{i,6}$
$w_{i,7}$
$w_{i,8}$
$w_{i,9}$
$a_i := f_i(h(i))$
sortie a_i vs désirée d_i

Introduction

L'objectif de l'intelligence artificielle est de mettre en place des systèmes informatiques de capacités intellectuelles calquées sur celles de l'être humain.

En 1943, Mc Culloh, un neuro-physiologiste, et Pitts, un logicien, ont proposé la notion de neurones formels. Ces derniers étaient censés imiter les neurones biologiques et être capables de mémoriser des fonctions booléennes simples. Les réseaux de neurones artificiels conçus à partir de ce type de neurones furent inspirés du système nerveux. Ces réseaux doivent être capables d'apprendre, de mémoriser, de généraliser de l'information dans les connexions entre les neurones et de traiter de l'information incomplète. Beaucoup de travaux ont eu lieu sur ce sujet et de nos jours nous avons des modèles performants faisant des réseaux neuronaux un des modèles de calcul les plus prisés en Intelligence Artificielle.



Les réseaux neuronaux fournissent des outils et des algorithmes performants pour : la prédiction, l'interpolation, la classification, la segmentation ou clustering, la reconnaissance des formes …

Ils trouvent leurs applications dans plusieurs domaines : codes postaux, prévision des séries temporelles dans la finance, diagnostic médical, identification de segments de clients potentiels, détection de fraude, modélisations des vers de spin, niveau d'ozone, sociologie, mesures biologiques …

Leur capacité à généraliser et à apprendre des données en font les meilleurs simulateurs de la façon dont le cerveau humain apprend de son expérience.

Seulement, l'adaptation d'un réseau à une application se réalise "à l'aveugle" sans compréhension structurelle de cette application ni de la raison de la validité du réseau obtenu après l'entrainement.

Historique

- 1940 : Alan Türing : Machine de Turing

- 1943 : Warren McCulloch et Walter Pitts : Modèle neuronal artificiel

- 1948 : Von Neuman : Les réseaux d'automates

- 1949 : Donald Hebb : Hypothèse de l'efficacité synaptique, notion de mémoire associative, premières règles d'apprentissage : le principe de Hebb.

- 1960 : Franck Rosenblatt et Bernard Widrow : Perceptron et l'Adaline.

- 1969 : Marvin MIinsky et Seymour Papert : Analyse théorique des capacités de calcul des perceptrons. Exibent un contre exemple.

- 1980 et plus : Stephen Grossberg et Teuvo Kohonen : Découvertes de nouvelles voies : auto-organisation des réseaux et processus d'adaptation. Renaissance du Connexionnisme.

- 1982 : John Hopfield : Analogies avec la Mécanique Statistique. Comportement de systèmes constitués d'un grand nombre d'éléments simples interagissant fortement. Application des nombreux résultats de la physique théorique au connexionnisme.

- 1985 : Machine de Boltzmann et Perceptron Multi-Couches (PMC) : Mécanismes d'apprentissage performants.

Chapitre 1

Principes généraux

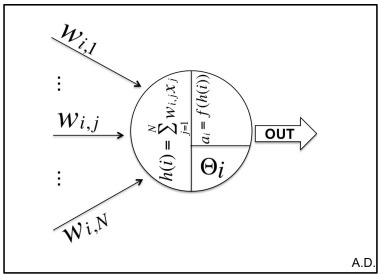
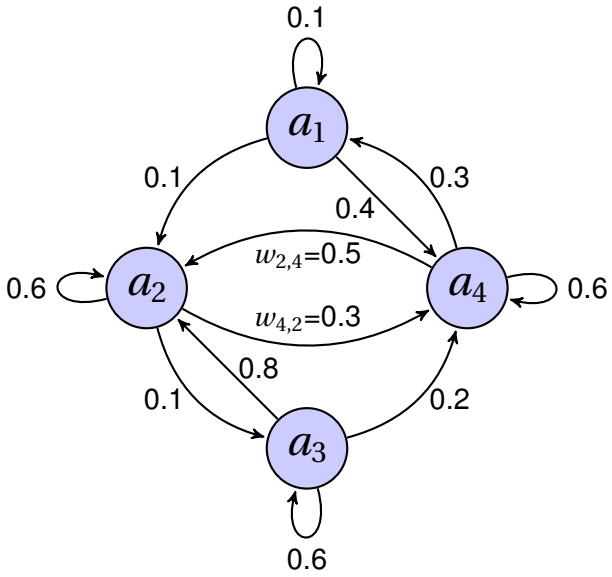


FIGURE 1. Illustration d'un neurone artificiel

1. Définitions

Un *réseau neuronal* est un graphe orienté pondéré. Un noeud de ce réseau est appelé un *neurone formel* ou une *unité connexioniste* (ou simplement *neurone*).



Nous considérons un réseau neuronal comportant N > 1 neurones.

1.1. Activation.

Chaque neurone formel est doté d'un état interne appelé l' *activation*. Notons $a_1, \ldots, a_N$ les activations respectives des neurones du réseau. Le vecteur $\boldsymbol{a}$ tel que $\boldsymbol{a} = \begin{pmatrix} a_1 \\ \vdots \\ a_N \end{pmatrix}$ est le *vecteur d'activation*.

En général, les a_i appartiennent à un ensemble d'états $\mathscr{A}$ possédant un structure d'algèbre (booléens, entiers, réels, ...).

1.2. Poids synaptiques.

Un arc pondéré entre deux neurones formels est appelé un *lien synaptique* et la pondération un *poids synaptique*. Le poids synaptique du neurone j vers le neurone i sera noté $w_{i,j}$ $(i, j \in [1, N])$ avec $w_{i,j} = 0$ s'il n'existe pas de lien du neurone j vers le neurone i. La *matrice des poids synaptiques*

$$W = (w_{i,j})_{1 \leq i,j \leq N}$$

mesure la connectivité du réseau.

1.3. Seuil.

Soit le neurone $i \in [1, N]$. Une valeur appelée *seuil* et notée Θ_i est parfois fixée.

1.4. Fonction d'entrée.

Soit le neurone $i \in [1, N]$. Une fonction h_i dite *d'entrée* du neurone i est donnée. En général, elle sera identique pour tous les neurones du réseau ou ceux d'une même couche et notée alors simplement h. Elle dépend du vecteur d'activation $\boldsymbol{a}$ et de la matrice des poids W ou plus exactement de W_i son i-ième vecteur ligne.

Lorsque le vecteur d'activation $\boldsymbol{a}$ est binaire, la fonction h sera booléenne et pourra dépendre d'un **biais** ϕ_i. Dans les cas où le vecteur $\boldsymbol{a}$ est numérique, sauf cas particuliers, la fonction h est ou bien linéaire (cas $\phi_i = 0$) :

$$h(i) = h(i, \boldsymbol{a}, W) := W_i \,.\, \boldsymbol{a} = \sum_{j=1}^{N} w_{i,j} a_j$$

ou bien affine :

$$h(i) = h(i, \boldsymbol{a}, W, \phi_i) := \sum_{j=1}^{N} w_{i,j} a_j - \phi_i = \sum_{j=0}^{N} w_{i,j} a_j \,,$$

avec $w_{i,0} = \phi_i$ et $a_0 = -1$. Les biais non nuls seront donc traités avec un neurone supplémentaire. Par la suite, nous ne considérerons donc pas les fonctions affines (sauf précision).

Il existe un cas particulier pour les réseaux dits *RBF* (voir Exemple 1.6) dans lesquels la fonction d'entrée de la couche RBF est une distance du vecteur d'activation au (transposé du) vecteur W_i :

$$h(i) = h(i, \boldsymbol{a}, \mathrm{W}, \phi_i) := \left\| \boldsymbol{a} - {}^t\mathrm{W}_i \right\| \quad .$$

La valeur $e_i(t) = h(i, \boldsymbol{a}(t), \mathrm{W})$ est appelée l'*activation pondérée* ou le *potentiel* du neurone i à l'instant $t \geq 0$. Dans la littérature, l'activation pondérée est parfois notée Net_i.

Lorsque la *propagation* est synchrone et h linéaire, le vecteur colonne $\boldsymbol{e} = (e_i)_{1 \leq i \leq \mathrm{N}}$ des activations pondérées à l'instant $t \geq 0$ est donné par

$$\boldsymbol{e}(t) = \mathrm{W}.\boldsymbol{a}(t) \, .$$

1.5. Règle ou fonction d'activation.

Chaque neurone $i \in [1, \mathrm{N}]$ possède une *règle* f_i appelée aussi *fonction d'activation*. En général, elle est identique à tous les neurones du réseau ou ceux d'une même "couche", notée alors simplement f. Elle s'applique sur le résultat de h_i la fonction d'entrée, c'est-à-dire sur l'activation pondérée e_i. A l'instant $t > 0$, nous avons :

$$a_i(t) = f_i(e_i(t-1), \Theta_i) \quad .$$

La fonction d'activation peut être

(1) à valeurs continues ou discrètes (pour les VLSI)

(2) déterministe ou stochastique (différentes réponses selon une distribution de probabilité donnée)

(3) à mémoire ou sans mémoire.

Ses caractéristiques sont :

— **monotonie** : la fonction d'activation f est en général croissante ;
— **seuillage** : pour la résistance au bruit le neurone i est affecté d'un seuil Θ_i ; si $f(x) < \Theta_i$ alors $f(x)$ est négligeable ;
— **saturation** pour éviter de propager de grandes valeurs : si $f(x) > \mathrm{M}$ alors $a_i = \mathrm{M}$;
— **dérivabilité**,pour l'apprentissage, par exemple.

Les fonctions d'activation sont souvent impaires.

Exemples de fonctions d'activation.

Rappelons que h est la fonction d'entrée du neurone i et Θ_i son éventuel seuil. Nous décrivons ci-dessous les fonctions d'activation les plus usitées.

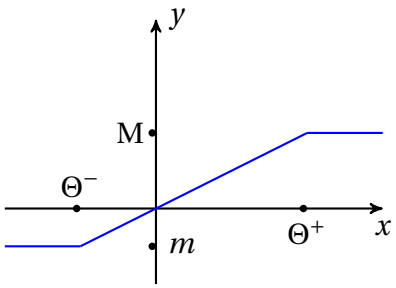
Exemple 1.1. Fonction linéaire : $f(x) = \lambda x$. D'où $a_i = \lambda h(i)$.

Exemple 1.2. *Fonction de seuil* ou *fonction de signe*. Soit un seuil Θ.

$$f(x) = \begin{cases} 1 & \text{si } x \geq \Theta \\ 0 & \text{sinon} \end{cases}$$

Exemple 1.3. Fonction linéaire bornée par Θ^- et Θ^+ :

$$f(x) = \begin{cases} \mathrm{M} & \text{si } x > \Theta^+ \\ k\,x & \text{si } \Theta^- \leq x \leq \Theta^+ \\ m & x < \text{si } \Theta^- \end{cases}$$



Exemple 1.4. Fonction *sigmoïde exponentielle* :

$$f(x) = \frac{1}{1+e^{-x}} \qquad .$$

Sa dérivée est

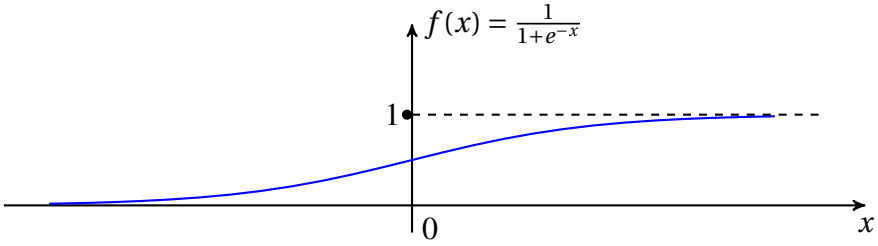
$$f'(x) = f(x)(1-f(x))\,, \tag{1}$$

ce qui montre, avec la notation $a_i = f(e_i)$, l'identité suivante :

$$\frac{da_i}{de_i} = a_i(1-a_i) \qquad . \tag{2}$$

DÉMONSTRATION. Avec $u(x) = 1+e^{-x} = 1/f$, nous avons $u'(x) = -e^{-x} = 1-u(x) = 1-1/f$ et donc $f'(x) = -\frac{u'(x)}{u(x)^2} = (1/f-1)f^2 = (1-f)f$. □

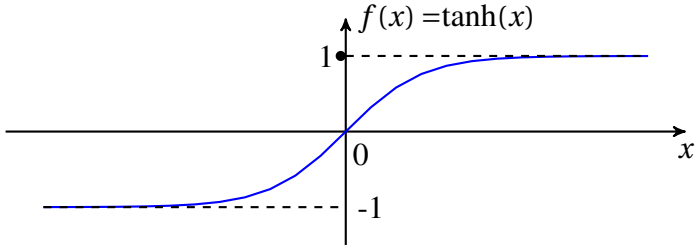
$$\lim_{x\to\infty} f(x) = 1 \qquad \text{et} \qquad \lim_{x\to-\infty} f(x) = 0\,.$$



Cette fonction continue monotone croissante bornée est celle souvent utilisée dans le réseau appelé le *perceptron multicouches* que nous étudierons plus tard.

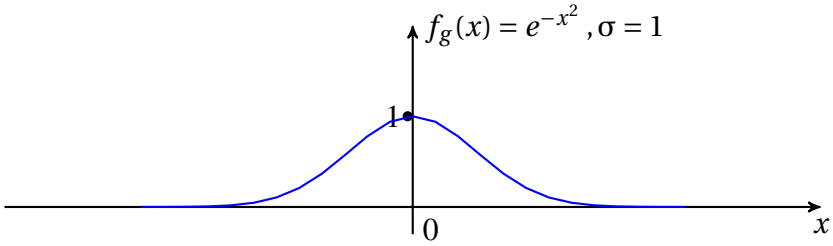
Exemple 1.5. La fonction *sigmoïde tangentielle* converge plus vite que la sigmoïde exponentielle.

$$f(x) = \tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \qquad \text{et } f'(x) = 1 - f(x)^2 \qquad .$$



Exemple 1.6. Fonction *gaussienne* :

$$f(x) = e^{-x^2/\sigma^2} \qquad \text{avec} \qquad f'(x) = -\frac{2x}{\sigma^2} f(x)\,.$$



Ces fonctions sont utilisées dans les réseaux *Radial Basis Function* dits **RBF**. Dans le cas particulier des réseaux **RBF**, la fonction d'entrée h du neurone i est donnée par la distance du vecteur $\mathbf{a}$ d'activation au i-ième vecteur ligne W_i de la matrice des poids W :

$$h(i) = \sqrt{\sum_{j=1}^{\mathrm{N}} (w_{i,j} - a_j)^2} = \left\| {}^t\mathrm{W}_i - \mathbf{a} \right\| \qquad .$$

Nous verrons au paragraphe 5 du Chapitre 2 que ce réseau est "à couches" et que les vecteurs de poids ${}^t\mathrm{W}_i$ sont en fait les "centres" ou "noyaux" notés $\mathbf{c}^i$ du réseau RBF (voir précisément la figure

page 46). Soient d le nombre de neurones de la couche précédent la couche RBF et $\mathbf{c}^i \in \mathbb{R}^d$, le noyau du neurone i de la couche RBF. Alors la fonction

$\psi = f o \delta$ où $\delta(\boldsymbol{x}) = \left\|\mathbf{c}^i - \mathbf{x}\right\|^2$ est quant à elle une fonction dite "radiale de base" de $\mathbf{x} \in \mathbb{R}^d$ dans $\mathbb{R}$.

Note : Dans la pratique, nous pourrons aussi choisir $h(i)^2$ comme fonction d'entrée avec la fonction d'activation $g(x) = e^{-x/\sigma^2}$ (qui n'est pas une gaussienne). En effet, afin d'éviter de considérer le couple (f, h) où $h(i)$ est une racine carrée, nous pouvons choisir le couple (g, h^2), ce qui revient exactement au même puisque $a_i = f(h(i)) = g(h(i)^2)$.

Exemple 1.7. Fonction *stochastique* : soit T la température, nous avons

$$f(x) = \begin{cases} 1 & \text{avec la probabilité } \frac{1}{1+e^{-x/\mathrm{T}}} \\ 0 & \text{sinon} \end{cases}$$

Nous écrivons aussi :

$$\mathrm{P}[a_i = 1] = \frac{1}{1 + e^{-\mathrm{e}_i/\mathrm{T}}} .$$

Lorsque la température T tend vers 0, la fonction f tend vers une fonction de seuil.

Exemple 1.8. *Fonction à mémoire*. Le but est de tenir compte de l'état d'activation du neurone avant qu'il soit de nouveau activé. Soit t un entier, le vecteur d'activation $a(t)$ du réseau est obtenu après avoir été activé t fois et $e_i(t)$ est l'activation pondérée sur le neurone i au temps t. La variable t représente des valeurs entières du temps. Définissons donc la fonction d'activation du neurone i en fonction du temps t :

$$f(x(t+1)) = f(x(t)) + \begin{cases} (\Theta^+ - f(x(t)))x(t) - \gamma(f(x(t)) - \Theta^0) & \text{si } x \geq 0 \\ (f(x(t)) - \Theta^-)x(t) - \gamma(f(x(t)) - \Theta^0) & \text{sinon} \end{cases} .$$

Nous avons $f(e_i(t)) = a_i(t)$ en posant $x(t) = e_i(t)$.

1.6. Équation générale.

Pour tenir compte de tous les types de fonctions d'activation nous pouvons écrire que l'activation $a_i(t+1)$ du neurone i à l'instant $t+1$ est donné par :

$$a_i(t+1) = a_i(t) + f(h(i)(t), \Theta_i, \mathrm{I}(t)) \tag{3}$$

où f est la fonction d'activation du neurone i, $h(i)(t)$ la fonction d'entrée à l'instant t, Θ_i le seuil et $\mathrm{I}(t)$ une valeur d'entrée à l'instant t. Ce qui dans la littérature s'écrira aussi sous la forme :

$$\frac{dx}{dt} = \mathrm{F}(x, \mathrm{W}, \Theta, \mathrm{I}) \qquad , \tag{4}$$

permettant d'étudier l'évolution du réseau comme un système dynamique.

1.7. Types de réseaux les plus utilisés.

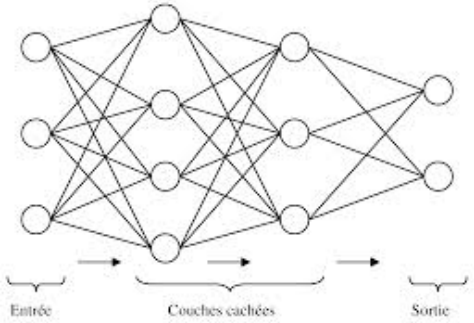
— Automates booléens : les entrées et sorties sont booléennes ou bien les sorties sont des fonctions booléennes des entiers.
— Automates à seuil : entrées binaires ou réelles, sorties binaires, fonction d'entrée h affine, fonction d'activation binaire à seuil.
— Automates linéaires : entrées et sorties réelles, fonction d'entrée h linéaire et l'identité est la fonction d'activation f.
— Automates à saturation : entrées et sorties dans un intervalle $[u, v]$, fonction d'entrée h linéaire et fonction d'activation à saturation ; si les entrées et sorties sont entières ce sont des *automates multi-seuils*.
— Automates continus : entrées et sorties réelles, fonction d'entrée h affine, fonction d'activation sigmoïde.
— Automates probabilistes : sorties binaires, entrées quelconques, fonction d'entrée h linéaire ou affine, fonction d'activation stochastique.

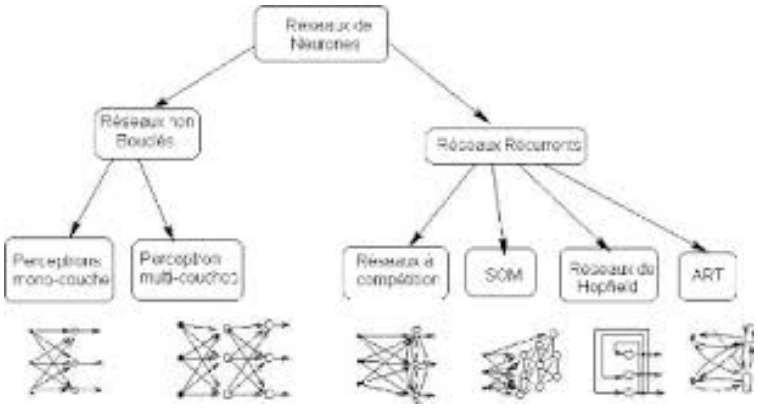
1.8. Architecture du réseau.

L'architecture du réseau est déterminée par la connectivité (poids $w_{i,j}$ nuls ou non nuls), le nombre et le type des neurones. Elle est déterminée par les caractéristiques décrites ci-après.

— Le réseau comporte des *boucles* s'il existe $i_1, \ldots, i_p \in [1, \mathrm{N}]$ tels que $i_1 = i_p$ et $w_{i_k, i_{k+1}} \neq 0$ pour $k \in [1, p-1]$. S'il en est ainsi, le réseau est dit *récurrent*. Si le réseau est non récurrent alors la matrice des poids W est triangularisable.
— Structuration en *couches de neurones* : connectivité intra-couche.
— *Connectivité inter-couches*. Supposons que le réseau soit organisé en couches $\mathrm{C}_1, \ldots \mathrm{C}_p$ de neurones ; la connectivité inter-couches est *complète* si pour tout $i, j \in [1, p]$ alors tout neurone de la couche C_i est connecté à chaque neurone de la couche C_j. La connectivité est dite *bijective* si pour tout $i, j \in [1, p]$ alors tout neurone de la couche C_i est connecté a exactement un unique neurone de la couche C_j. La connectivité est dite *probabiliste* si elle est distribuée selon une probabilité ou une distribution (gaussienne).
— *Symétries* dans les connections comme dans le **réseau de Hopfield** que nous étudierons dans l'exemple 3.1.

Un réseau multi-couches





2. Modèle de McColloch et Pitts (1943)

L'idée est de comparer les cellules nerveuses à des fonctions logiques.

Nous supposons toujours que N neurones constituent notre réseau.

Les états d'activation sont binaires et prennent leurs valeurs dans $\{0,1\}$. Le 0 signifie qu'il n'y a pas d'émission et si l'activation vaut 1 alors le neurone émet.

Les valeurs de la matrice des poids sont également binaires (0 ou 1). Soient $i, j \in [1,\mathrm{N}]$. Si le lien d'un neurone j vers un neurone i a pour poids synaptique $w_{i,j} = 0$ alors le neurone j est qualifié d'*inhibiteur* et si $w_{i,j} = 1$ alors il est qualifié d'*excitateur*.

Le neurone i est affecté d'une valeur Θ_i, son seuil. Sa fonction d'entrée est une variante de la linéaire car son image n'est pas une unique valeur mais un doublet. Elle est définie par :

$$h(i) = (\sum_{j=1}^{\mathrm{N}} w_{i,j}\, a_j, \sum_{j=1}^{\mathrm{N}} (1 - w_{i,j})\, a_j) \in [0,\mathrm{N}]^2;$$

la valeur de la première coordonnée est le nombre de neurones activés excitateurs et celle de la seconde coordonnée est le nombre de neurones activés inhibiteurs. Nous avons toujours $a_i = f(h(i))$. Ainsi la fonction d'activation possède deux paramètres. Elle est donnée par :

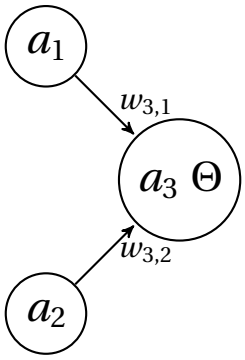
$$f(x,y) = \begin{cases} 1 & \text{si } x > \Theta \text{ et } y = 0 \\ 0 & \text{sinon} \end{cases}.$$

Ce qui signifie que le neurone i est activé si les neurones excitateurs (i.e. $w_{i,j} = 1$) activés (i.e. $a_j = 1$) sont en nombre supérieur à un seuil Θ et qu'aucun des neurones inhibiteurs (i.e. $w_{i,j} = 0$) n'est activé (i.e. $a_j = 0$).

Exemple 2.1. **Modélisations booléennes**

Il s'agit de modéliser les tables de vérité du `ET`, du `OU` et du `NON`. Le vrai vaut 1 et le faux vaut 0.

Le nombre N de neurones est 3 avec 2 couches : 2 neurones notés N_1 et N_2 en entrée et un, N_3, en sortie. Le neurone de sortie possède un seuil noté Θ.



Il nous faut définir le seuil de N_3 et la matrice des poids W pour ET, OU et NON qui prend des valeurs $w_{i,j}$ nulles sauf éventuellement en $(i,j) = (3,1)$ ou $(i,j) = (3,2)$.

Tables de vérités

ET	0	1
0	0	0
1	0	1

OU	0	1
0	0	1
1	1	1

Au temps $t = 0$, $a_3 = 0$. Nous avons $h(3) = (w_{3,1}a_1 + w_{3,2}a_2, (1 - w_{3,1})a_1 + (1 - w_{3,2})a_2) = (e_3, g_3)$ et pour modéliser une table de vérité, le réseau doit satisfaire l'équation $a_3 = f(e_3, g_3)$ pour tout couple (a_1, a_2), où a_3 est donné par le couple (a_1, a_2) dans la table de vérité.

Pour le `ET` : Vérifions que $w_{3,1} = w_{3,2} = 1$ et $\Theta = 1$.

Ici $a_3 = f(a_1 + a_2, 0)$. Donc aucun neurone activé n'est inhibiteur. Si $a_1 = a_2 = 1$ (i.e. les neurones N_1 et N_2 sont tous deux activés) alors $a_1 + a_2 = 2 > 1 = \Theta$ et donc $a_3 = f(2,0) = 1$. Dans les trois autres cas, $a_3 = f(a_1 + a_2, 0) = 0$ puisque $a_1 + a_2 \leq 1 \not> \Theta = 1$. La table de vérité du `ET` est donc modélisée par ce réseau.

Pour le `OU` : Retrouvons : $w_{3,1} = w_{3,2} = 1$ et $\Theta = 0$.

Pour $(a_1, a_2) = (0,0)$, nous devons avoir $a_3 = 0$. Puisque $h(3) = (0,0)$, la fonction d'activation doit satisfaire l'équation $a_3 = f(0,0) = 0$; ce qui impose $e_3 = 0 \leq \Theta$ pour le seuil. Pour les trois autres couples $(1,0)$, $(0,1)$ et $(1,1)$ de valeurs de (a_1, a_2), la fonction f doit satisfaire respectivement les équations $f(w_{3,1}, 1 - w_{3,1}) = 1$, $f(w_{3,2}, 1 - w_{3,2}) = 1$ et $f(w_{3,1} + w_{3,2}, (1 - w_{3,1}) + (1 - w_{3,2})) = 1$. D'après la définition de f, sa seconde coordonnée y doit être nulle pour que l'image $f(x,y)$ ne le soit pas. D'où $1 - w_{3,1} = 1 - w_{3,2} = 0$. Comme pour le `ET`, pour toutes les valeurs de (a_1, a_2), nous avons $w_{3,1} = w_{3,2} = 1$ et $h(3) = (a_1 + a_2, 0)$. Il reste à déterminer le seuil Θ. En considérant $(a_1, a_2) \neq (0,0)$, pour qu'à la fois $f(a_1 + a_2, 0) = 1$ et $f(0,0) = 0$, il faut et il suffit que $0 \leq \Theta < a_1 + a_2 \in \{1,2\}$. Il n'existe qu'une seule valeur entière pour le seuil Θ, c'est zéro.

Pour le `NON` : Vérifions que $w_{3,1} = w_{3,2} = 0$ avec $\Theta = -1$ et $a_1 = 0$ (imposé).

Tous les neurones sont inhibiteurs et $a_3 = f(0, a_2)$. Si N_2 est vrai, i.e. $a_2 = 1$, alors $a_3 = f(0,1) = 0$, i.e. N_3 est faux (a_2 neurone activé inhibiteur). Si $a_2 = 0$ alors $a_3 = f(0,0) = 1$ car $0 > \Theta = -1$. Donc

le réseau modélise bien le NON (a_2 neurone inhibiteur non activé).

3. Fonctionnement et principe du réseau de neurones

3.1. Construction d'un réseau.

En fonction du problème à résoudre, le principe des réseaux de neurones est de se fixer :

— une architecture (nombre de neurones et connectivité) ;
— une matrice W dont les zéros sont induits par la connectivité du réseau ; i.e. $w_{i,j} = 0$ lorsque les neurones i et j ne sont pas connectés ;
— les fonctions d'entrée et d'activation de chaque neurone : en général identiques pour tous les neurones ou au moins pour les neurones d'une même couche.

Ces choix sont guidés par le type d'applications à traiter. Il existe plusieurs modèles connus qui répondent à des types d'applications particulières. Nous étudierons les plus classiques qui répondent à de nombreux de problèmes.

3.2. Fonctionnement du réseau.

Le réseau fonctionne selon deux modes :

— en *mode de reconnaissance* (voir Paragraphe 3.2.1) : le réseau est utilisé pour calculer ;

— en *mode d'apprentissage* (voir Paragraphe 3.2.2) : le réseau s'adapte à l'application à l'aide d'exemples ; cette adaptation affecte la matrice des poids W, le nombre de neurones, parfois le "pas d'apprentissage" (voir la règle Delta-Bar-Delta Chapitre 2 Paragraphe 1.3,) et très rarement les fonctions d'activation.

3.2.1. *Mode de reconnaissance - Propagation de l'activation.*

Il s'agit de savoir comment faire fonctionner le réseau.

Un *cycle de propagation* est le calcul de l'activation, en temps discret, de tous les neurones du réseau.

La propagation d'un cycle est soit *synchrone* (i.e. simultanée), soit *synchrone par couches* (par ex., les perceptrons), soit *asynchrone* (i.e. séquentielle avec un ordre fixé) soit *aléatoire* (i.e. séquentielle avec un ordre aléatoire). Dans le cas aléatoire la trajectoire peut être affectée par l'ordre.

Exemple 3.1. **Connexions symétriques (Hopfield, 1982)**.

Nous reviendrons sur ce type de réseaux au paragraphe 6.

Nous allons faire fonctionner un réseau symétrique particulier (N = 4) d'abord en mode synchrone puis en mode asynchrone. Ensuite, dans l'espace discret des 2^N états possibles du réseau, nous comparerons les trajectoires respectives de chacun de ces deux modes. Sera étudié ensuite le comportement de "fonction E d'énergie" de la trajectoire en asynchrone ; fonction d'énergie sur laquelle nous reviendrons ultérieurement dans un cadre général avec la "fonction de Lyapunov" d'un système dynamique (voir Paragraphe 4.2).

Description du réseau

Un réseau neuronal de N neurones et de matrice des poids $W = (w_{i,j})_{i,j\in[1,N]}$ (dite aussi de *transition*)) est dit *symétrique* si les poids synaptiques $w_{i,j}$ vérifient : $w_{i,j} = w_{j,i}$ pour tout $i, j \in [1, N]$ (i.e. la matrice W est symétrique).

Le *réseau symétrique de Hopfield* est un réseau symétrique caractérisé ainsi :

- il est booléen ; les activations des N neurones sont à valeur dans $\{0, 1\}$,
- sa matrice symétrique des poids $W \in \{0, 1\}^{N^2}$ est de diagonale nulle : $w_{i,i} = 0$ pour tout $i \in [1, N]$,
- la fonction d'entrée de chaque neurone $i \in [1, N]$ à l'instant t est définie par :

$$h(i, t) = \sum_{j=1}^{N} w_{i,j} a_j(t)$$

- la fonction d'activation commune à tous les neurones du réseau est ainsi définie de $\mathbb{N}$ dans $\{0, 1\}$:

$$f(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{sinon} \end{cases} .$$

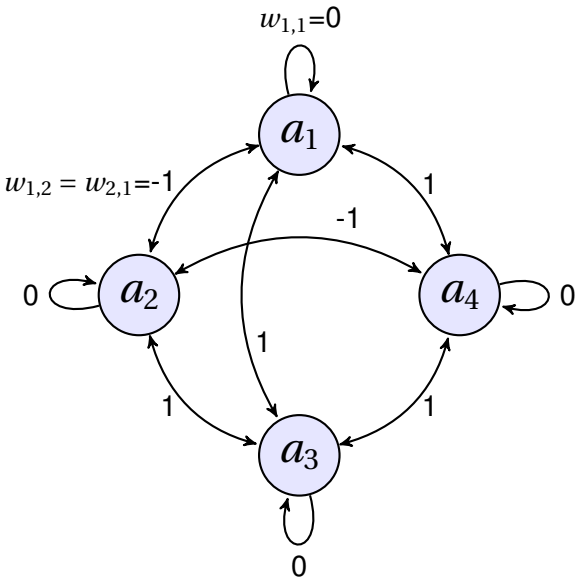
Pour chaque instant $t \in \mathbb{N}$, nous notons $a_i(t)$ l'activation du neurone i, $e_i(t) = h(i, t)$ son activation pondéré et $\boldsymbol{a}(t)$ le vecteur d'activation du réseau.

L'équation du réseau symétrique de Hopfield est :

(5) $$a_i(t+1) = f(e_i(t)) \quad \text{où} \quad e_j(t) = \sum_{j=1}^{N} w_{i,j} a_j(t) \qquad .$$

Fixons N = 4 et donnons-nous le réseau neuronal symétrique (de Hopfield) dont la matrice W symétrique des poids est la suivante :

$$W = \begin{pmatrix} 0 & -1 & 1 & 1 \\ -1 & 0 & 1 & -1 \\ 1 & 1 & 0 & 1 \\ 1 & -1 & 1 & 0 \end{pmatrix}$$



Propagations synchrones et asynchrones

Nous allons comparer le fonctionnement de notre réseau en synchrone et en asynchrone. Pour expliquer ces deux fonctionnements, nous étudions une trajectoire avec comme vecteur d'activation $\boldsymbol{a}(0) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ au temps $t = 0$.

Propagation synchrone :

A chaque instant t, il suffit de calculer l'activation pondérée du réseau $\boldsymbol{e}(t) = {}^t((e_1(t), e_2(t), e_3(t), e_4(t))$:

$$\boldsymbol{e}(t) = \mathrm{W}.\boldsymbol{a}(t)$$

puis, selon l'équation (5) du réseau, l'activation du neurone i à l'instant $t+1$ est donnée par $a_i(t+1) = f(e_i(t))$.

Posons $\mathrm{F} = \begin{pmatrix} f \\ f \\ f \\ f \end{pmatrix}$, où f est la fonction d'activation. Au temps $t = 1$, le vecteur d'activation du réseau est :

$$\boldsymbol{a}(1) = \mathrm{F}(\mathrm{W}.\boldsymbol{a}(0)) = \mathrm{F}\begin{pmatrix} 0 \\ -1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

et aux temps $t = 2$ et $t = 3$, le réseau rentre dans un état stationnaire :

$$\boldsymbol{a}(2) = \mathrm{F}(\mathrm{W}.\boldsymbol{a}(1)) = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \boldsymbol{a}(3)\,.$$

Regardons cette évolution comme celle d'un système dynamique discret : Soit

$$\Psi = \begin{pmatrix} \psi_1 \\ \vdots \\ \psi_4 \end{pmatrix} \quad \text{où} \quad a_i(t+1) = f(e_i(t)) = \psi_i(\boldsymbol{a}(t)) \quad .$$

Nous avons alors :

$$\boldsymbol{a}(t+1) = \Psi(\boldsymbol{a}(t)) = \Psi^{t+1}(\boldsymbol{a}(0)) \quad t \geq 0 \quad .$$

L'évolution de la trajectoire à partir de $\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ arrive à l'état stable $\mathbf{v}$:

$$\mathbf{v} := \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \Psi(\begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}) = \Psi^2(\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}) = \Psi^t(\mathbf{v}) \quad \forall\, t \in \mathbb{N} \quad .$$

Propagation asynchrone :

Un ordre de propagation est choisi. Par exemple, commençons par le neurone N_1 (activation a_1) puis le neurone N_2 puis le neurone N_3 et enfin le neurone N_4. Avec au départ (temps $t = 0$) $\boldsymbol{a}(0) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$, nous avons $\boldsymbol{a}_1(1) = f(\boldsymbol{e}_1(0)) = f(0) = 0$ et $\boldsymbol{a}_i(1) = \boldsymbol{a}_i(0)$ pour $i = 2,3,4$ d'où $\boldsymbol{a}(1) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ qui est nécessairement un point fixe.

Nous constatons que le comportement en synchrone diffère de celui en asynchrone. Observons les trajectoires.

Trajectoires

En synchrone, l'équation $\boldsymbol{a}(t) = \Psi(\boldsymbol{a}(t-1))$ est celle d'un système dynamique discret. Les trajectoires en synchrone se calculent donc en appliquant ainsi Ψ à toutes les valeurs initiales possibles $\boldsymbol{a}(0)$:

$$\boldsymbol{a}(t) = \Psi(\boldsymbol{a}(t-1)) = \Psi^2(\boldsymbol{a}(t-2)) = \cdots = \Psi^t(\boldsymbol{a}(0)) \quad .$$

Nous allons réaliser une étude complète des trajectoires en fonctionnement de propagation synchrone puis asynchrone en partant des $2^4 = 16$ valeurs possibles de l'état initial de l'activation $\boldsymbol{a}(0)$.

Notons $\boldsymbol{b} = (\boldsymbol{b}^1, \ldots, \boldsymbol{b}^{16})$ la liste des 16 vecteurs d'états possibles du réseau suivants :

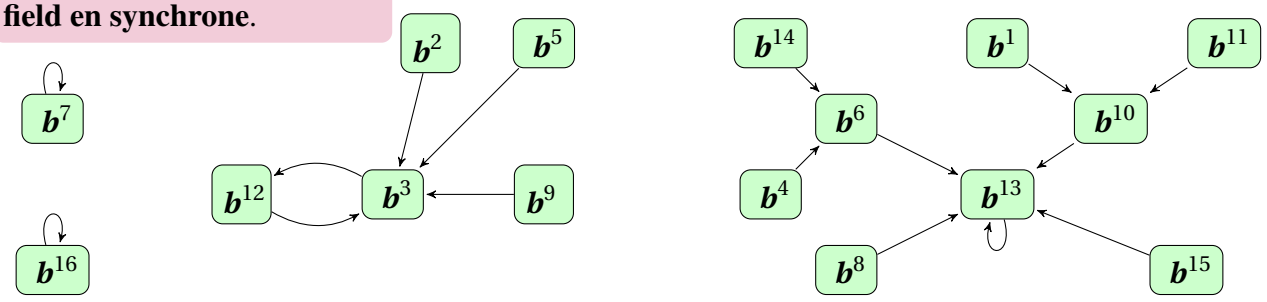
$\boldsymbol{b}^1$	$\boldsymbol{b}^2$	$\boldsymbol{b}^3$	$\boldsymbol{b}^4$	$\boldsymbol{b}^5$	$\boldsymbol{b}^6$	$\boldsymbol{b}^7$	$\boldsymbol{b}^8$	$\boldsymbol{b}^9$	$\boldsymbol{b}^{10}$	$\boldsymbol{b}^{11}$	$\boldsymbol{b}^{12}$	$\boldsymbol{b}^{13}$	$\boldsymbol{b}^{14}$	$\boldsymbol{b}^{15}$	$\boldsymbol{b}^{16}$
1	0	0	0	1	1	0	1	0	0	1	1	1	0	1	0
0	1	0	0	1	0	1	0	1	0	1	1	0	1	1	0
0	0	1	0	0	1	1	0	0	1	1	0	1	1	1	0
0	0	0	1	0	0	0	1	1	1	0	1	1	1	1	0

Nous distinguons les vecteurs d'activation $\boldsymbol{a}(t)$ à l'instant t des 16 vecteurs $\boldsymbol{b}^i$ de leurs valeurs possibles.

Propagation synchrone :

Les trajectoires du réseau en propagation synchrone à partir de chacun des 16 états sont :

Trajectoires réseau de Hopfield en synchrone.



Par exemple, en partant de l'état initial $\boldsymbol{a}(0) = \boldsymbol{b}^4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$, une première propagation met le réseau dans l'état $\boldsymbol{a}(1) = \boldsymbol{b}^6 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$, puis une deuxième met le réseau dans l'état stable $\boldsymbol{a}(2) = \boldsymbol{b}^{13} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$.

Nous avons donc : $\boldsymbol{b}^{13} = \Psi(\boldsymbol{b}^6) = \Psi^2(\boldsymbol{b}^4)$.

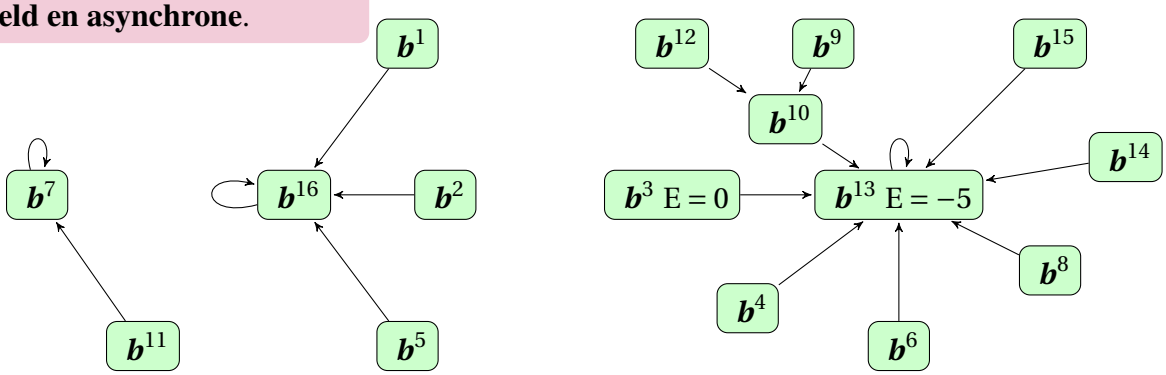
D'après les trajectoires, les 16 valeurs de $\boldsymbol{a}(0)$ forment un système dynamique discret avec :

- 4 trajectoires (i.e. non connexes),
- 3 états stables $\boldsymbol{b}^7, \boldsymbol{b}^{13}$ et $\boldsymbol{b}^{16}$, i.e. $\Psi(\boldsymbol{b}) = \boldsymbol{b}$,
- 1 cycle : $\{\boldsymbol{b}^3, \boldsymbol{b}^{12}\}$ de longueur 2, c'est-à-dire $\Psi(\boldsymbol{b}^3) = \boldsymbol{b}^{12}$ et $\Psi(\boldsymbol{b}^{12}) = \boldsymbol{b}^3$.

En asynchrone :

Avec au départ (temps $t = 0$) $\boldsymbol{a}(0) = \boldsymbol{b}^2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$, la trajectoire converge vers un point fixe $\boldsymbol{b}^{16}$ alors qu'en synchrone elle aboutit au cycle $\{\boldsymbol{b}^3, \boldsymbol{b}^{12}\}$. En calculant les trajectoires en fonction des 16 vecteurs d'états initiaux possibles, nous obtenons :

Trajectoires réseau de Hopfield en asynchrone.



Il n'y a aucun cycle et les trajectoires convergent vers les 3 états stables $\boldsymbol{b}^7, \boldsymbol{b}^{13}$ et $\boldsymbol{b}^{16}$.

Convergence et fonction d'énergie : Théorème de Hopfield

Hopfield a établi le théorème suivant :

THÉORÈME 3.2 (Hopfield). *En propagation asynchrone, ce type de réseau évolue en minimisant la fonction d'énergie :*

$$\mathrm{E}(t) = -\sum_{i=1}^{\mathrm{N}} \sum_{j=1}^{\mathrm{N}} w_{i,j} a_i(t) a_j(t) = -\langle \boldsymbol{a}(t), \mathrm{W}.\boldsymbol{a}(t) \rangle \qquad .$$

Dans notre exemple, pour l'état $\boldsymbol{a}(t) = \boldsymbol{b}^3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$, la fonction d'énergie est $\mathrm{E} = -\langle (0,0,1,0), (\$, \$, 1 * 0 + 1 * 0 + 1 * 0 + 1 * 0, \$) \rangle = 0$ (où \$ désigne une valeur quelconque) et, pour l'état suivant (ici un stable) $\boldsymbol{a}(t+1) = \boldsymbol{b}^{13} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$, nous avons $\mathrm{E}(t+1) = -5 < \mathrm{E}(t)$.

Remarquons que la fonction d'énergie est minorée : $\mathrm{E} \geq -\mathrm{N}(\mathrm{N}-1)$.

DÉMONSTRATION. (Théorème de Hopfield) Nous supposons que le réseau évolue en propagation asynchrone. La fonction d'énergie étant minorée, il suffit de montrer qu'elle décroit nécessairement d'un cycle à l'autre jusqu'à ce que le système parvienne à un état stable lorsqu'elle

aura atteint son minimum sur cette trajectoire. Soit $i \in [1, N]$. La fonction d'énergie s'exprime alors sous la forme :

$$E(t) = -\sum_{i=1}^{N} e_i(t) a_i(t) .$$

Le réseau évoluant en propagation asynchrone, seul un neurone que nous supposons être le i-ième aura son activation modifiée à l'instant t. Puisque $a_j(t) = a_j(t-1)$ pour $j \neq i$ et $w_{i,i} = 0$, nous avons d'une part

$$e_i(t) = \sum_{j \neq i} w_{i,j} a_j(t) + 0.a_i(t) = \sum_{j \neq i} a_j(t-1) + 0.a_i(t-1) = e_i(t-1) \tag{6}$$

et d'autre part $e_j(t) - e_j(t-1) = w_{j,i}(a_i(t) - a_i(t-1))$ d'où, comme $w_{i,j} = w_{j,i}$:

$$\begin{aligned} \Delta E &= E(t) - E(t-1) = -(a_i(t) - a_i(t-1)).e_i(t-1) - \sum_{j \neq i} a_j(t-1)(e_j(t) - e_j(t-1)) \\ &= -(a_i(t) - a_i(t-1)).e_i(t-1) - \sum_{j \neq i} a_j(t-1) w_{i,j}(a_i(t) - a_i(t-1)) \end{aligned}$$

En appliquant une nouvelle fois (6), nous obtenons finalement :

$$\Delta E = -2(a_i(t) - a_i(t-1)).e_i(t-1) \quad .$$

Nous avons $a_i(t) = f(e_i(t-1)) = 1$ si $e_i(t-1) > 0$ et $a_i(t) = 0$ sinon. Lorsque $e_i(t-1) \neq 0$, le neurone i change d'état du temps $t-1$ au temps t ssi $a_i(t-1) = 1$ et $a_i(t) = 0$ donc $e_i(t-1) < 0$ ou bien $a_i(t-1) = 0$ et $a_i(t) = 1$ donc $e_i(t-1) > 0$; ce qui est équivalent à $(a_i(t) - a_i(t-1)).e_i(t-1) > 0$; d'où

$$\Delta(E) = -2(a_i(t) - a_i(t-1)).e_i(t-1) < 0 \quad .$$

Supposons que $e_i(t-1) = 0$. Si pour tout $\lambda \in [\![1, N-1]\!]$ $e_{i+\lambda \mod N}(t-1+\lambda) = 0$ alors pour tout $j \in [\![1, N]\!]$ et $s \geq t$ $a_j(s) = 0$. Le système est alors dans l'état stable correspondant au vecteur d'activation nul. Donc $\Delta(E) < 0$ entre deux cycles ssi il y a changement d'état. Comme la fonction d'énergie est bornée inférieurement, E atteindra un minimum et le système sera alors dans un état stable. □

Exercice. Écrire un programme qui calcule les listes de trajectoires et les fonctions d'énergies pour chaque état. Ce programme s'écrit rapidement dans un système de calcul formel tel `Maxima`, `Maple` ou `Sage`.

3.2.2. *Mode d'apprentissage : présentation générale.*

Les valeurs données aux coefficients de la matrice des poids synaptiques W sont arbitraires et ne sont sûrement pas les plus adaptées pour répondre à l'application considérée. Il faut donc les modifier pour les adapter à cette application. Cette modification est réalisée avec un choix de *règle d'apprentissage*.

Pour corriger la matrice W des poids, nous utiliserons un *corpus* $\mathscr{C}$ d'apprentissage (i.e. un échantillon).

Remarque 1. Il faudra s'assurer que le changement d'activation soit plus rapide que l'évolution des poids afin d'en assurer l'indépendance. Le changement d'activation est dit *adiabatique* par rapport à celui des poids.

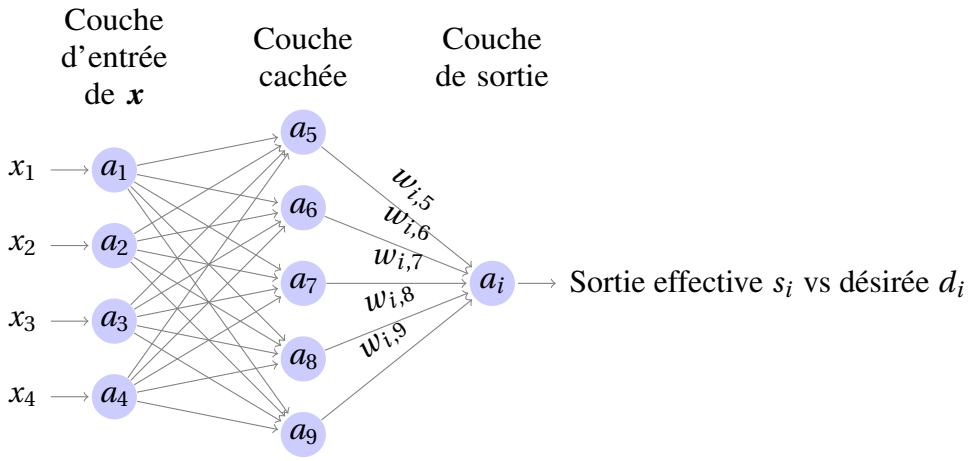
Le réseau est une boite noire avec une entrée de k neurones et une sortie de m neurones.

L'apprentissage est *supervisé* ou *non supervisé*.

Dans l'*apprentissage supervisé* le corpus $\mathscr{C}$ est un ensemble de couples $(\boldsymbol{x}, \boldsymbol{d})$ où $\boldsymbol{x}$ est le vecteur des k entrées, le *patron d'entrée*, présentées au réseau et $\boldsymbol{d}$ le vecteur des m valeurs désirées en sortie, le *patron de sortie*, que nous connaissons a priori.

L'entrée $\boldsymbol{x}$ est présentée au réseau. Un vecteur $\boldsymbol{s}$ de m valeurs est récupéré à la sortie du réseau. C'est en comparant le vecteur $\boldsymbol{s}$ et la valeur désirée $\boldsymbol{d}$ que sont corrigés les poids synaptiques. L'idée est de modifier la matrice des poids W de telle sorte qu'une fonction d'erreur E obtenue en comparant $\boldsymbol{d}$ et $\boldsymbol{s}$ soit minimisée. La matrice finale W des poids obtenue suite à l'apprentissage doit être bien adaptée pour la généralisation (i.e. possède de bonnes capacités d'interpolation) comme expliqué au paragraphe 3.4.

Ce mode d'apprentissage est utilisé pour la prédiction, l'identification, la classification, … (voir logiciels `R`, `Matlab`, …).



Dans l'*apprentissage non supervisé*, la sortie $\boldsymbol{d}$ n'est pas connue puisqu'il s'agit de regrouper les exemples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ formant le corpus $\mathscr{C}$ et présentées successivement en entrées au réseau. Il s'agit de modifier la matrice des poids synaptiques W afin qu'ils soient suffisamment regroupés et de s'assurer que la généralisation est correcte (voir paragraphe 3.4).

Ce type d'apprentissage s'applique pour l'auto-classification, par exemple. Le logiciel `SPAD` réalise de l'apprentissage non supervisé.

3.3. Fonctions d'erreur entre $\boldsymbol{d}$ et $\boldsymbol{s}$.

Soient $\boldsymbol{d} = (d_1, \ldots, d_m)$ la sortie désirée du réseau et $\boldsymbol{s} = (s_1, \ldots, s_m)$ la sortie effective.

Les plus usitées sont la fonction d'*erreur quadratique* :

$$\mathrm{E} = \frac{1}{2}\sum_{i=1}^{m}(d_i - s_i)^2 \tag{7}$$

et la fonction de *corrélation croisée* vers laquelle s'orientent de nouvelles applications car elle répond à l'inconvénient de la faible vitesse de convergence de la précédente :

$$\mathrm{E} = -\sum_{i=1}^{m}(d_i\, ln(s_i) + (1-d_i)\, ln(1-s_i)). \tag{8}$$

Seulement cette dernière est non adaptée aux données binaires.

Nous reviendrons sur la comparaison de ces fonctions d'erreur suite à l'étude du Perceptron Multi-Couche, le PMC, dont l'apprentissage est illustré avec l'erreur quadratique (voir Paragraphe 4.3 Chapitre 2).

3.4. Corpus d'apprentissage, de test et de validation.

Une fois opéré l'apprentissage sur un corpus dit d'*apprentissage*, il faut s'assurer que le réseau réagit correctement à la généralisation. C'est-à-dire qu'il réagit bien à des données qui n'appartiennent pas au corpus d'apprentissage. Donc, il faut disposer d'un autre corpus, appelé le *corpus test* ou *corpus de généralisation*. La généralisation est considérée comme correcte si le réseau se comporte correctement sur le corpus test. Par exemple, pour l'apprentissage non supervisé, la généralisation est considérée correcte si de nouveaux exemples, ceux du corpus test, donnés au réseau sont aussi regroupés.

Il est important de disposer d'un troisième corpus dit de *validation* qui interviendra en fin d'apprentissage.

La *méthode de validation croisée* (cross-validation) suit l'algorithme suivant (voir le logiciel SPAD pour la classification en non supervisé) :

(1) réaliser l'apprentissage avec le corpus d'apprentissage,

(2) suspendre régulièrement l'apprentissage pour passer au corpus test ;

(3) valider a posteriori avec le corpus de validation.

Recommandations pour le corpus d'apprentissage :
Le corpus d'apprentissage doit couvrir tous les cas et ne doit pas être trié selon un ordre significatif, qui pourrait induire le réseau en erreur. Il faut 5 à 10 individus pour ajuster chaque poids.

4. Comportements dynamiques - Fonction de Lyapunov

Le réseau peut converger vers des états stables (voir le réseau symétrique de Hopfield, Exemple 3.1) ou des états plus complexes. Les systèmes peuvent se classer en trois classes : *convergent, oscillatoires* et *chaotiques*.

Définition 4.1. Un point fixe est un *attracteur* s'il est entouré d'une région dans laquelle toutes les trajectoires *transitoires* convergent sur lui. Cette région est aussi appelée *bassin de l'attracteur* (c'est un *bassin d'attraction*).

Un système dynamique est dit *convergent* si chaque trajectoire approche un état d'équilibre. Plus généralement, un système dynamique est *quasiconvergent* si chaque orbite approche un ensemble d'états d'équilibre, sans converger vers un en particulier. Cela n'arrive que si l'ensemble d'équilibre est fini. Un système lisse (ou C^1) dissipatif (l'ensemble des états est le bassin d'au moins un attracteur) est approximable par des systèmes avec un nombre fini d'états d'équilibre.

Un système est dit *oscillatoire* si chaque ensemble limite est une orbite périodique (éventuellement stationnaire). Par exemple, le pendule sans friction ou les réseaux *neuronaux biologiques*. De même que pour les systèmes convergents, il se définit également les systèmes *quasioscillatoires*. Un tel système peut être modélisé en couplant adroitement des systèmes convergents. Les réseaux neuronaux oscillatoires ont été étudiés par Cowan et Ermentrout (1988), Li et Hopfield (1989), Crossberg et Elias (1975). Ils sont spécialement étudiés par des biologistes. Kopell et Ermentrout (1988) ont modélisé le système nerveux de la lamproie avec des réseaux neuronaux d'oscillateurs couplés.

Un système est dit *chaotique* s'il est difficile de faire une prédiction fiable à long terme. La faisabilité de l'utilisation de systèmes oscillatoires ou chaotiques à travers des réseaux neuronaux particuliers pour la reconnaissance (pattern recognition) est démontrées dans Baird, Troyer et Eeckman.

A part de très rares cas, les réseaux neuronaux sont en majorité convergents ou supposés l'être.

4.1. Convergence vers un point fixe.

Le critère est l'existence d'une *fonction de Lyapunov* :

Définition 4.2. Une fonction est dite de *Lyapunov* si elle est continue à valeurs réelles sur l'espace des états qui décroît strictement en temps le long de chaque trajectoire non stationnaire et qui possède une borne inférieure.

La fonction d'erreur que nous étudierons dans la *rétropropagation*, celles d'autres méthodes d'apprentissage ou l'entropie négative des systèmes mécaniques statistiques sont des fonctions de Lyapunov.

Un état a sera un point d'équilibre du système si c'est un point critique d'une fonction de Lyapunov $\mathscr{L}$ du système. C'est-à-dire que le gradient de $\mathscr{L}$ s'annule en a.

Exemple 4.3. Prenons un exemple issu du site `Mathworld` (http ://mathworld.wolfram.com/LyapunovFunction.html). Soit le système :

$$y' = z$$
$$z' = -y - 2z$$

et la fonction de Lyapunov $\mathscr{L}(y,z) = (y^2 + z^2)/2$ définie positive en tout point $(x,y) \neq (0,0)$; le long de la courbe, nous obtenons :

$$\mathscr{L}'(y,z) = <\text{grad}\mathscr{L}, (y',z') >= yz + z(-y-2z) = -2z^2$$

qui est décroissante dans toute région contenant l'origine,. La solution nulle est donc stable.

Les systèmes les plus naturels possédant une fonction $\mathscr{L}$ de Lyapunov sont les systèmes de gradients. De tels systèmes sont engendrés par des équations différentielles de la forme :

$$\frac{dx_i}{dt} = -\frac{\partial \mathscr{L}}{\partial x_i} \tag{9}$$

où $\mathscr{L}$ est supposée de classe C^1 et possède une borne inférieure. La fonction $\mathscr{L}$ est une fonction de Lyapunov du système décrit par l'évolution des x_i puisque le long de la courbe décrite par l'équation (9), nous avons : $\frac{\partial \mathscr{L}}{\partial t} = -\frac{\partial \mathscr{L}}{\partial x_i}\frac{dx_i}{dt} = -(\frac{d\mathscr{L}}{dxi})^2 \leq 0$.

Plus vectoriellement, prenons le système décrit par $d\boldsymbol{x}/dt = -\text{grad}\mathscr{L}$, où $\mathscr{L}$ est supposée de classe C^1. Le long de la courbe de solution $\boldsymbol{x}(t)$, nous avons :

$$\mathscr{L}'(\boldsymbol{x}(t)) = <\text{grad}\mathscr{L}, d\boldsymbol{x}/dt >= -|\text{grad}\mathscr{L}|^2 \leq 0 \quad .$$

Lyapunov énonça ce théorème fondamental :

THÉORÈME 4.4. *Soit* $\mathscr{L} : X \mapsto \mathbf{R}$ *une fonction de Lyapunov d'un système dynamique dissipatif* (X, Φ). *Alors :*

(i) Le système est quasiconvergent

(ii) Chaque point local minimum de $\mathscr{L}$ *est un équilibre local orbital*

(iii) Le système est convergent dans chacun des deux cas suivant :
(1) $grad\mathscr{L} = 0$ *en un nombre fini ou dénombrable d'états ; ou*
(2) le système est le gradient d'une fonction réelle analytique.

Mais il existe peu de moyens pour prouver l'existence d'une fonction de Lyapunov d'un système et ensuite de la calculer. Le théorème suivant de Cohen et Lyapunov (1983) est inclus dans de nombreux modèles de réseaux neuronaux :

THÉORÈME 4.5. *Pour tout système de la forme :*

$$\frac{dx_i}{dt} = a_i(x)[b_i(x) + \sum_j w_{i,j} g_j(x_j)]\,, \tag{10}$$

il existe une fonction de Lyapunov $\mathscr{L}$ si
(1) les fonctions a_i sont positives,
(2) les dérivées g'_j sont positives et
(3) la matrice $W = (w_{i,j})$ *est symétrique. Une fonction de Lyapunov du système est alors donnée par :*

$$\mathscr{L}(x) = \sum_i \int_0^{x_i} b_i(s) g'_i(s) ds - \frac{1}{2} \sum_{i,j} w_{i,j} g_j(x_j) g_i(x_i) \,. \tag{11}$$

Pour un réseau neuronal, les conditions suivantes sont connues pour être suffisantes à l'existence d'un point fixe :

- la matrice W des poids synaptiques est triangulaire avec une diagonale nulle (c'est un réseau nécessairement non récurrent) ;
- la matrice W est symétrique à diagonale positive comme dans les réseaux de Hopfield en asynchrone et les machines de Boltzmann (voir Section 6 et Exemple 3.1) ;
- $$\sum_{i,j} w_{i,j} < \frac{1}{\max_{i \in N} |f'_i|} ;$$
  donc, comme pour une sigmoïde exponentielle, il est judicieux de choisir une fonction d'activation f_i dont la dérivée est bornée ;
- la diagonale de la matrice Jacobienne de l'activation du réseau est dominante ; en général $J_{i,j} = \delta_{i,j} - w_{i,j}.f'_i(e_i)$ où $\delta_{i,j}$ est symbole de Kronecker.

Exemple 4.6. Dans l'exemple 3.1, avec Hopfield choisissons la fonction d'énergie :

$$E(t) = -\sum_{i=1}^{N} e_i(t) a_i(t) \,,$$

de dérivée partielle pour tout $i \in [1, N]$:

$$\frac{\partial E}{\partial a_i} = -e_i(t) \,.$$

Cette fonction est une fonction de Lyapunov du système :

$$\frac{da_i}{dt} = -a_i(t) + f(-\frac{\partial E(t)}{\partial a_i(t)}) \,,\ i \in [1, N] \,,$$

modélisé par le réseau en fonctionnement asynchrone puisqu'au temps t, un seul état i est modifié et que nous avons alors :

$$\frac{da_i}{dt} = a_i(t+1) - a_i(t) = -a_i(t) + f(e_i(t)) = -a_i(t) + f(-\frac{\partial E(t)}{\partial a_i(t)}) \quad .$$

4.2. Comportements dynamiques plus complexes.

Comme nous l'avons vu, il peut y avoir des cycles limites, des trajectoires chaotiques, des trajectoires transitoires (i.e. celles qui ne se stabilisent pas).

Ce qui est alors recherché est de contraindre le réseau à suivre une trajectoire donnée. Cela relève de la **prédiction et du contrôle de systèmes dynamiques**.

5. Erreur et apprentissage

Une *règle d'apprentissage* est une règle qui permet d'adapter un réseau à l'application concernée. Cette règle porte le plus souvent sur la matrice des poids synaptiques.

5.1. Apprentissage supervisé avec une fonction d'erreur.

L'objectif est de calculer la "meilleure" matrice des poids W. Soit un réseau de N neurones avec k neurones en entrée et m en sortie. Nous supposons disposer de

- $\boldsymbol{x}$, un *patron d'entrée* codé sous forme d'un vecteur $(x_1, \ldots, x_k)$,
- $\boldsymbol{s} = (s_1, \ldots, s_m)$, un *patron de sortie* calculé en présentant l'entrée $\boldsymbol{x}$ au réseau,
- si l'apprentissage est supervisé : $\boldsymbol{d} = (d_1, \ldots, d_m)$ un *patron de référence* (ou de sortie désirée) correspondant à la valeur que devrait prendre $\boldsymbol{s}$ comme valeur si le réseau était parfaitement adapté à l'application.

Soit $\mathrm{E}(\boldsymbol{s}, \boldsymbol{d})$, une fonction d'erreur entre la sortie effective $\boldsymbol{s}$ et la sortie désirée $\boldsymbol{d}$. Par exemple, l'erreur quadratique

$$\mathrm{E}(\boldsymbol{s}, \boldsymbol{d}) = \frac{1}{2} \sum_{i=1}^{m} (s_i - d_i)^2 \,. \tag{12}$$

L'apprentissage consiste en l'initialisation $\mathrm{W} = \mathrm{W}(0)$ de la matrice W des poids synaptiques et en l'exécution de plusieurs cycles d'apprentissages jusqu'à ce que la fonction d'erreur $\mathrm{E}(\boldsymbol{s}, \boldsymbol{d})$ atteigne un minimum. Soit $\mathrm{W} = \mathrm{W}(t)$ l'état de la matrice des poids à l'instant t, un *cycle (ou époque) d'apprentissage* consiste à

(1) présenter le patron d'entrée $\boldsymbol{x}$ au réseau et calculer $\boldsymbol{s}$ en propageant l'activation,

(2) calculer l'erreur $\mathrm{E}(\boldsymbol{s}, \boldsymbol{d})$ entre $\boldsymbol{s}$ et $\boldsymbol{d}$,

(3) en déduire la nouvelle matrice des poids $\mathrm{W}(t+1)$.

La formule établissant $\mathrm{W}(t+1)$ est la **règle d'apprentissage** des poids du réseau.

Puisque la modification des poids constitue un système dynamique (discret) le long duquel nous voulons faire décroître la fonction d'erreur jusqu'à son minimum (qu'on aimerait global ...), nous considérons le système dynamique décrit par l'équation (9) du paragraphe 4 où les x_i sont les poids $w_{i,j}$ et où la fonction $\mathscr{L}$ de Lyapunov est la fonction d'erreur E :

$$\frac{\partial w_{i,j}}{\partial t} = -\frac{\partial \mathrm{E}}{\partial w_{i,j}} \quad .$$

Comme E est la fonction de Lyapunov du système dynamique continu et que le système dynamique de correction des poids est discret, un coefficient correctif multiplicatif λ appelé **pas d'apprentissage** est inséré dans l'équation du système.

L'équation d'apprentissage, i.e. de la correction des poids, se présentera donc usuellement (presque toujours) sous la forme :

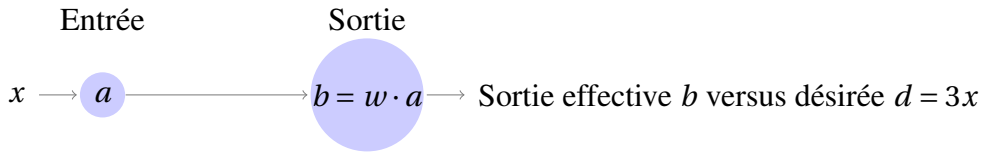
$$\Delta \mathrm{W} = \mathrm{W}(t+1) - \mathrm{W}(t) = -\lambda \frac{\partial \mathrm{E}}{\partial \mathrm{W}} \quad . \tag{13}$$

Ainsi le pas d'apprentissage λ de (13) permet d'adapter une équation d'un système dynamique dans laquelle la variable de temps t varie continuement dans $\mathbb{R}^+$ à un cas dans lequel t varie de manière discrète dans $\mathbb{N}$. L'exemple 5.1 illustre la nécessité du pas d'apprentissage même dans un cas des plus élémentaires.

Exemple 5.1. Soit un réseau modélisant l'équation $y = 3x$. L'architecture du réseau est constituée de deux neurones dont les activations respectives sont a, prenant la valeur d'entrée x, et b, celle du neurone de sortie. La matrice W des poids synaptiques est de dimension 2×2 :

$$\mathrm{W} = \begin{pmatrix} 0 & 0 \\ w & 0 \end{pmatrix}, \quad \text{avec } w \in \mathbf{R}$$

La fonction d'entrée du neurone b de sortie est $h(b) = w \cdot a$ et sa fonction d'activation est l'identité.



Nous disposons d'un jeu d'essai (un corpus $\mathscr{C}$, ici infini) : $(x,d) = (1,3),(2,6),(3,9),\ldots$ afin d'adapter la matrice des poids W (i.e. la valeur w) à l'application (i.e. apprendre) ; i.e. on cherche à modifier w afin d'approcher la valeur de la sortie effective de celle de la sortie désirée d après que la valeur x ait été présentée en entrée au réseau. Les valeurs successives de w aux temps $t = 0,1,...$ seront successivement notées $w(0), w(1), \ldots$; la trajectoire des poids formant ainsi un système dynamique discret de l'apprentissage des poids.

Comment les poids peuvent-ils être modifiés pour apprendre ? En fait l'apprentissage en supervisé (i.e. on connait la sortie désirée) consiste en ce qu'une fonction d'erreur $\mathrm{E}(t)$ donnée entre la sortie effective et la sortie désirée atteigne en $t = t_0$ un "minimum local' " (zéro si possible) le long de la trajectoire décrite par le système dynamique (discret) $w(t)$. Dans l'idéal, l'objectif est de parvenir à décrire un système dynamique $w(t)$ tel qu'à partir d'une valeur initiale $w(0)$, la trajectoire des poids atteigne en un temps fini $t = t_0$ un poids stable $w(t_0)$ satisfaisant $\mathrm{E}(t_0) = 0$.

Nous choisissons cette fonction d'erreur (le 1/2 de l'erreur quadratique est omis délibérément) :

$$\mathrm{E}(t) = (d - w(t) \cdot x)^2 \geq 0$$

Or, comme nous l'avons vu précédemment, une fonction $\mathscr{L}$ définie positive, donc $\mathscr{L} := \mathrm{E}$, est naturellement la fonction de Lyapunov du système dynamique continu (9) qui pour le cas discret se "traduit" par cette équation :

$$\frac{w(t+1) - w(t)}{(t+1) - t} = -\lambda \cdot \frac{d\mathrm{E}}{dw} \quad \text{où} \quad \frac{d\mathrm{E}}{dw} = -2x(d - w \cdot x)$$

où λ est le pas d'apprentissage. Donc la règle d'apprentissage décrivant la trajectoire des poids et que nous allons appliquer est la suivante :

$$w(t+1) = w(t) + 2x \cdot \lambda \cdot (d - w(t) \cdot x)\,. \tag{14}$$

La valeur de λ est choisie souvent assez petite pour discrétiser plus finement. Si $\lambda = 1$, il est possible d'osciller longtemps autour d'un point d'équilibre avant de l'atteindre ou pas du tout. Choisissons tout de même $\lambda = 1$ pour notre exemple.

Au départ ($t = 0$), choisissons $w = w(0) = 4$ et $(x, d) = (1, 3)$ et décrivons le début de la trajectoire suivie par $w(t)$ selon l'équation $w(t+1) = w(t) + 2x(d - w(t) \cdot x)$. Avec $x = 1$ en entrée le réseau ressort la valeur $w \cdot x = 4$ au lieu de la valeur désirée $d = 3$. D'où :

$$w(1) = w(0) + 2 \cdot 1 \cdot (3 - 4) = 2\,.$$

Maintenant avec $w = w(1) = 2$, étudions le comportement avec $(x, d) = (2, 6)$. Le réseau sort $w \cdot x = 4$ et donc

$$w(2) = w(1) + 2 \cdot 2 \cdot (6 - 4) = 2 + 8 = 10.$$

Puis avec $(x, d) = (3, 9)$ et $w(2) = 10$ les choses empirent. Le réseau sort $w \cdot x = 30$ et donc :

$$w(3) = w(2) + 2 \cdot 3 \cdot (9 - 30) = -116.$$

Nous constatons que le pas $\lambda = 1$ est trop important pour trouver l'état stable de la courbe des poids où l'erreur E prend son minimum. Ce qui est vrai en continue ne l'est pas nécessairement en discret.

Choisissons $\lambda = 1/2$. L'équation de changement des poids devient : $w(t+1) = w(t) + x(d - w(t) \cdot x)$ ou, exprimé, autrement :

$$w(t+1) = w(t) \cdot (1 - x^2) + d \cdot x \quad .$$

Avec $(x, d) = (1, k)$ dans le corpus (ici on cherche à apprendre la multiplication par k quelconque), pour toute valeur de $w(0)$, nous obtenons :

$$w(1) = k \ \text{, la valeur recherchée pour } w.$$

Au temps $t = 2$, pour tout couple (x, d) tel que $d = k \cdot x$, dans la règle d'apprentissage (14), nous aurons $d - w(1) \cdot x = 0$ et donc $w(2) = w(1)$ avec $\mathrm{E}(1) = \mathrm{E}(2) = 0$. La courbe des poids est arrivée dans un état stable en une étape avec E qui a atteint son minimum 0. Ainsi le réseau avec $w = k$ appris avec $\lambda = 1/2$ est parfaitement adapté à l'application qui est la multiplication par k.

Cet exemple montre clairement l'importance du choix de la valeur du pas de correction (apprentissage) λ. Ce que font les praticiens est de choisir λ assez petit pour descendre doucement vers l'état d'équilibre plutôt que d'osciller autour. Nous verrons qu'il est également possible d'apprendre le pas d'apprentissage.

5.2. La corrélation en Cascade. Due à Fahlman et Lebière (1990) cette méthode consiste à rajouter des neurones. ll utilise le QuickProp (Fahlman, 1988) (voir 1.2) et cherche à maximiser, pour un certain neurone supplémentaire, la covariance entre son activation et l'erreur des neurones de sortie.

5.3. Le neurochirurgien optimal (Optimal Brain Surgeon OBS). Elle est l'inverse de la corrélation en cascade (voir Le Cun et al., 1990 ; Hassibi et al., 1993).

L'idée est de retrancher des neurones à un réseau déjà entrainé en développant la fonction d'erreur E en série de Taylor jusqu'à l'ordre 2 (le premier ordre étant nul par hypothèse). Le problème de cette méthode est le calcul de la matrice inverse de la hessienne

$$H = \partial^2 E / \partial W^2.$$

6. Application : Analyse des données (Data Mining)

Cette partie du cours émane du cours de Monsieur Fabrice Rossi (Université Paris-IX Dauphine, http ://apiacoa.org//contact.html).

Les domaines concernés sont : la reconnaissance des formes ; la classification ; la prédiction et le contrôle de processus (finance, prédiction temporelle comme la consommation électrique, contrôle de processus dynamiques).

L'analyse des données classique consiste en des méthodes d'exploration des données (ACP, régression, ...). Elle s'étend à l'organisation des données. L'apprentissage supervisé s'applique à :

- la discrimination : affectation de nouvelles données à des groupes donnés ; exemples : pour un diagnostic médical, disposer de mesures biologiques (tension artérielles, numération sanguine, …) pour des personnes saines et des personnes malades ; reconnaître un code postal ...
- la régression : modélisation de relations fonctionnelles ; exemples : niveau d'ozone demain en fonction du niveau d'aujourd'hui et de mesures météo comme la vitesse du vent, cours d'une action , interpolation ...

Les réseaux neuronaux ne sont pas efficaces pour comprendre la relation observations/cible. L'objectif est d'estimer la cible.

L'apprentissage non supervisé s'applique à :

- la classification : découverte de groupes dans des données ; exemples : regroupement de profils de consommateurs, groupes d'individus pour l'analyse sociologique.

Ce cours étant destiné à des statisticiens éclairés nous ne nous étendrons pas plus sur cette partie.

Chapitre 2

Quelques modèles classiques

Dans ce chapitre, nous conservons les notations précédentes. Nous débuterons par des règles d'apprentissage usuelles puis nous décrirons des réseaux particuliers.

1. Règles d'apprentissages

Afin d'illustrer ce que peuvent être des règles d'apprentissage, nous en présentons ici de trois sortes : la première agissant sur les liens au paragraphe 1.1, la deuxième sur la matrice des poids au paragraphe 1.2 (voir aussi la règle DELTA, Paragraphe 2.4) et la troisième sur le pas d'apprentissage au paragraphe 1.3.

1.1. Principe de Hebb.

Ce principe permet de modifier les liens entre les neurones.

$$w_{i,j} = \lambda \overline{a_i a_j}$$

où $\overline{a_i a_j}$ est la corrélation entre a_i et a_j et $\lambda \in [0,1]$ est le paramètre de l'intensité d'apprentissage. Pour l'apprentissage dit *anti-hebbien*, nous avons $\lambda \in [-1,0]$ (analyse en composante principale).

L'idée est la suivante : si deux neurones connectés entre eux sont activés de la même manière (au même moment) alors la connexion qui les relie doit être renforcée. Sinon, elle doit rester inchangée ou être affaiblie.

Dans l'exemple du paragraphe 6, si $a_i = a_j$ alors il faut renforcer la connexion. Centrons avec $s_k = 2a_k - 1$. Si les deux neurones ont la même activation nous avons $\lambda s_i s_j = \lambda$ et sinon $\lambda s_i s_j = -\lambda$.

1.2. Le QuickProp.

Cette règle est due à Fahlman (1988). L'idée est d'augmenter le pas d'apprentissage lorsque les dérivées de la fonction d'erreur E gardent le même signe entre deux cycles d'apprentissage :

$$\Delta w_{i,j}^t = \frac{\mathrm{S}_{i,j}^t}{\mathrm{S}_{i,j}^{t-1} - \mathrm{S}_{i,j}^t} \Delta w_{i,j}^{t-1}$$

où

$$\mathrm{S}_{i,j}^t = \frac{\partial \mathrm{E}^t}{\partial w_{i,j}^t} \quad .$$

Dans les bonnes conditions les gains sont spectaculaires.

1.3. Règle Delta-Bar-Delta.

Il s'agit d'apprendre le pas d'apprentissage $\lambda_{i,j}$ tel que

$$\Delta w_{i,j} = -\lambda_{i,j} \frac{\partial \mathrm{E}}{\partial w_{i,j}} \quad .$$

Le règle est la suivante : loin du point d'équilibre le pas d'apprentissage doit être augmenté et inversement ; ce qui se traduit par :

(15)
$$\Delta\lambda_{i,j}^t = \begin{cases} \mu & \text{si} \quad \mathrm{S}_{i,j}^t \frac{\partial \mathrm{E}^t}{\partial w_{i,j}^t} > 0 \\ -\Phi\lambda_{i,j}^t & \text{si} \quad \mathrm{S}_{i,j}^t \frac{\partial \mathrm{E}^t}{\partial w_{i,j}^t} < 0 \\ 0 & \text{sinon} \end{cases}$$

où

$$\mathrm{S}_{i,j}^t = (1-\Theta) \frac{\partial \mathrm{E}^t}{\partial w_{i,j}} + \Theta . \mathrm{S}_{i,j}^{t-1}$$

et μ, Φ et Θ sont des constantes petites ; Par exemple, 0,05 ; 0,7 et 0,7. En général, $\lambda_{i,j}^0 = 0,1$.

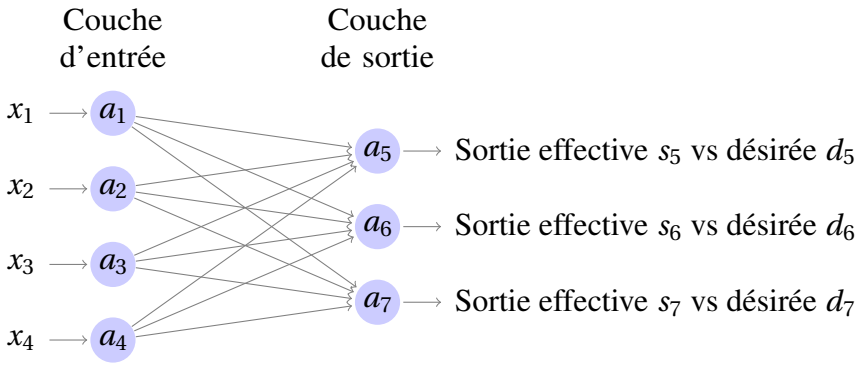
2. Le Perceptron de F. Rosenblatt (1958)

2.1. Description.

Rosenblatt proposa ce modèle dans l'intention de modéliser la perception humaine.

Ce réseau comporte deux couches : une d'entrée (neurones 1 à k) et une de sortie (neurones $k+1$ à N). La matrice des poids est donc de la forme :

$$\mathrm{W} = \begin{pmatrix} 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ w_{k+1,1} & \dots & w_{k+1,k} & 0 & 0 & \dots & 0 \\ w_{k+2,1} & \dots & w_{k+2,k} & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{\mathrm{N},1} & \dots & w_{\mathrm{N},k} & 0 & 0 & \dots & 0 \end{pmatrix}$$



La fonction d'activation est l'identité et la fonction d'entrée des neurones de la couche de sortie est donnée par les $\mathrm{N} - k$ dernières lignes du vecteur W.**a**, où **a** est le vecteur d'état.

Le fonctionnement est identique en synchrone et asynchrone. La propagation se réalise en deux temps. Au temps $t = 0$, nous avons $a_i := x_i$ pour $i \in [\![1, k]\!]$ et on peut supposer $a_{k+1} = \ldots = a_{\mathrm{N}} = 0$. Puis dans un second temps, $a_i := 0$ pour $i \in [\![1, k]\!]$ et $a_i = \sum_{i=1}^{k} w_{i,j} a_j$ pour $i \in [\![k+1, \mathrm{N}]\!]$

2.2. Que fait un tel réseau ?

Le perceptron est utilisé pour la classification. En fait, il permet de séparer linéairement l'espace à l'aide d'hyperplans.

Posons $p = \mathrm{N} - k$, le nombre de neurones en sortie et notons M la sous matrice $(w_{i,j})_{k+1 \le i \le \mathrm{N}; 1 \le j \le k}$ de dimension $p \times k$ de W constituée de ses colonnes 1 à k et de ses lignes $k+1$ à N.

Pour tout $(\boldsymbol{e}, \boldsymbol{d})$ du corpus, nous cherchons à approcher une matrice inconnue $\Lambda := (\lambda_{i,j})$ par M afin de minimiser l'erreur entre la sortie $\mathrm{M} \cdot \boldsymbol{e}$ du réseau et la sortie désirée $\boldsymbol{d}$ qui s'identifierait à $\Lambda \cdot \boldsymbol{e}$.

2.3. Limite du perceptron.

Toutes les fonctions ne sont pas *linéairement séparables*, c'est-à-dire, séparables par des hyperplans, comme le montre l'exemple suivant.

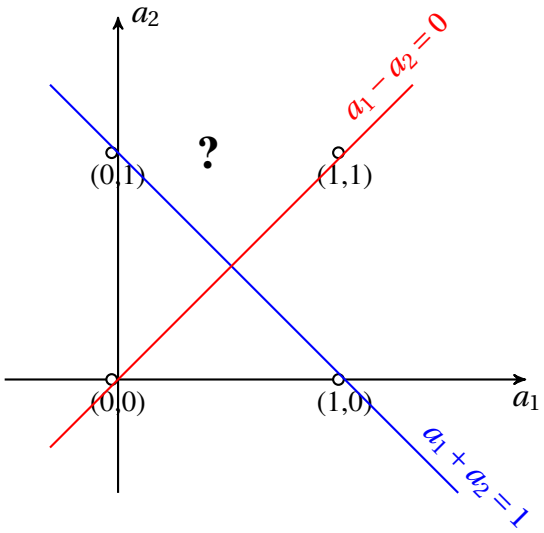
Exemple 2.1. Un exemple classique est celui du XOR (le ou exclusif) dont la table de vérité est la suivante :

XOR	0	1
0	0	1
1	1	0

Supposons que le XOR soit modélisable par un Perceptron. Nous avons un réseau avec 2 neurones en entrée d'activations respectives a_1 et a_2 et un de sortie d'activation a_3. Comme il n'y a qu'un neurone en sortie, nous cherchons les coefficients $w_{3,1}$ et $w_{3,2}$ de la droite $w_{3,1}a_1 + w_{3,2}a_2 = a_3$ qui modéliserait le XOR. Pour modéliser XOR, il faut simultanément avoir :

— la sortie $a_3 = 1$ (i.e. vrai) pour les entrées $(a_1, a_2) = (1,0)$ et $(0,1)$. L'équation de la droite serait alors $a_1 + a_2 = 1$ avec $w_{3,1} = w_{3,2} = 1$;

— la sortie $a_3 = 0$ (i.e. faux) pour les entrées $(0,0)$ et $(1,1)$. L'équation de la droite serait alors $a_1 - a_2 = 0$ avec $w_{3,2} = -1$. Comme, il y a contradiction entre $w_{3,2} = 1$ et $w_{3,2} = -1$, le XOR n'est pas modélisable par un Perceptron.



Cet exemple classique montre les limites du perceptron. L'idée fut donc d'utiliser des couches cachées. Voyons ci-après ce qu'il en est.

Comme la matrice W est très creuse et que nous fonctionnons par couches, nous considérons $\mathrm{A}_e^t = (a_1, \ldots, a_k)$ le vecteur des activations en entrée et $\mathrm{A}_s^t = (a_{k+1}, \ldots, a_{\mathrm{N}})$ celui des sorties. En posant

$$\mathrm{W}_{s,e} := (w_{i,j})_{k+1 \leq i \leq \mathrm{N}; 1 \leq j \leq k}$$

la sous-matrice de W constituée des colonnes 1 à k et des lignes $k+1$ à N (i.e. la matrice M introduite plus haut), nous obtenons :

$$\mathrm{A}_s = \mathrm{W}_{s,e}.\mathrm{A}_e \,.$$

De même, supposons qu'il existe une couche cachée C, de vecteur d'activation A_c, telle qu'il n'y ait pas de connection entre les neurones de cette couche, qu'elle reçoive des activations de la couche d'entrée avec les poids d'une matrice de poids $W_{c,e}$ et qu'elle active les neurones de sortie avec une matrice des poids $W_{s,c}$. Si on se contente d'un couche cachée avec une fonction d'activation toujours égale à l'identité pour tous les neurones alors le problème reste inchangé puisque :

$$A_s = W_{s,c}.A_c = W_{s,c}.W_{c,e}.A_e = W'.A_e\ ,$$

où W' est de nouveau une matrice des poids. Il faudra donc choisir judicieusement la fonction d'activation f de chaque neurone. En principe une unique couche cachée suffit à l'approximation des fonctions continues.

2.4. La règle d'apprentissage DELTA.

Le perceptron est utilisé pour la classification. La règle d'apprentissage du perceptron est la *règle DELTA*. Nous avons une couche d'entrée et une de sortie.

Soit la sortie s_i du neurone i de sortie et d_i sa sortie désirée. Alors la règle d'apprentissage du poids $w_{i,j}$ entre le neurone j (de la couche précédente) et le neurone i est la suivante :

$$\Delta w_{i,j} = \lambda \cdot a_j (d_i - s_i) \tag{16}$$

où $\lambda > 0$ est le pas d'apprentissage.

THÉORÈME 2.2 (Convergence de Rosenblatt, 1962). *Quelles que soient les entrées et la classification désirée, avec la règle DELTA, la convergence s'établit en un temps fini vers une matrice des poids* W *correcte si elle existe.*

Pour établir la règle DELTA d'apprentissage, nous nous référons au paragraphe 5.1, Chapitre 1.

Notons S l'ensemble des indices des neurones de sortie.

La règle DELTA d'apprentissage effectue une descente en gradient sur cette fonction d'erreur :

$$E = 1/2 \sum_{s \in S} (a_s - d_s)^2\ .$$

Comme E est une somme de carrés, le minimum sera unique. La correction des poids sera donnée par l'équation

$$\Delta w_{i,j} = -\lambda \frac{\partial E}{\partial w_{i,j}}$$

où λ est le pas d'apprentissage.

Nous cherchons à calculer $\frac{\partial E}{\partial w_{i,j}}$ où $i \in S$ est un neurone de sortie. Nous avons

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial h(i)} . \frac{\partial h(i)}{\partial w_{i,j}} \quad .$$

Afin de réutiliser ces calculs ultérieurement pour le PMC, oublions pour le moment que la fonction f d'activation est l'identité et considérons l'identité plus générale $a_i = f(h(i))$. Nous avons alors :

$$\frac{\partial \mathrm{E}}{\partial h(i)} = \frac{1}{2}\frac{\partial (a_i - d_i)^2}{\partial h(i)} = \frac{1}{2}\frac{\partial (a_i - d_i)^2}{\partial a_i} \cdot \frac{df(h(i))}{dh(i)}$$

d'où :

$$\frac{\partial \mathrm{E}}{\partial h(i)} = (a_i - d_i).f'(h(i)) \qquad \text{et} \tag{17}$$

$$\frac{\partial h(i)}{\partial w_{i,j}} = a_j\,, \tag{18}$$

puisque $h(i) = \sum w_{i,k} a_k$.

Nous avons ainsi établi que :

$$\frac{\partial \mathrm{E}}{\partial w_{i,j}} = a_j\,(a_i - d_i)\,f'(h(i)) \tag{19}$$

et donc la règle d'apprentissage pour toute fonction d'activation f est la suivante :

$$\Delta w_{i,j} = \lambda\, a_j\,(d_i - a_i)\,f'(h(i)) \tag{20}$$

où i est un neurone de sortie. On trouve la règle DELTA en posant $f = id$.

2.5. Algorithme de Windrow-Hoff.

Cet algorithme fut élaboré pour L'AdaLine par Windrow et Hoff (1960) (voir Paragraphe 3). Il applique la règle d'apprentissage DELTA après chaque passage d'un élément du corpus d'apprentissage. Cet algorithme s'est avéré plus efficace que celui qui consiste à accumuler toutes les modifications (via la règle DELTA) avec tous les éléments du corpus d'apprentissage pour ne modifier les poids qu'ensuite.

Dans l'algorithme suivant, nous nous focalisons sur un seul neurone i de la couche de sortie et cherchons à modifier la i-ème ligne $\mathbf{w}$ de la matrice des poids : $w_{i,j} = w_j$ où j parcourt les k neurones de la couche d'entrée. Nous supposons que le corpus d'apprentissage C est formé de couples $(\mathbf{x}, d)$ où d est la sortie désirée (du neurone i) lorsque $\mathbf{x}$ est le vecteur des k valeurs données en entrées au réseau.

Algorithme de **Windrow-Hoff**

ENTREES : Le vecteur **w** des `k` poids,
un corpus d'apprentissage `C`,
le pas d'apprentissage `lambda`.

SORTIE : La vecteur des poids **w** obtenu par la REGLE DELTA.

Pour tout `(x, d)` de `C` **Faire**
Pour `j=1` A `k` **Faire**
`s := w·x` ; la sortie effective
`w[j] := w[j] + lambda*(d-s)*x[j]`

Retourner `w`

3. Adaline

L'AdaLine (Adaptative Linear element) est due à Windrow et Hoff (1960).

Comme application en traitement du signal, nous pouvons citer les antennes adaptatives et les modems à haute vitesse (voir Windrow et Stearns en 1985).

La fonction d'entrée est affine :

$$h(i) = \sum_{j=1}^{N} w_{i,j} a_j - \Theta_i = \sum_{j=0}^{N} w_{i,j} a_j$$

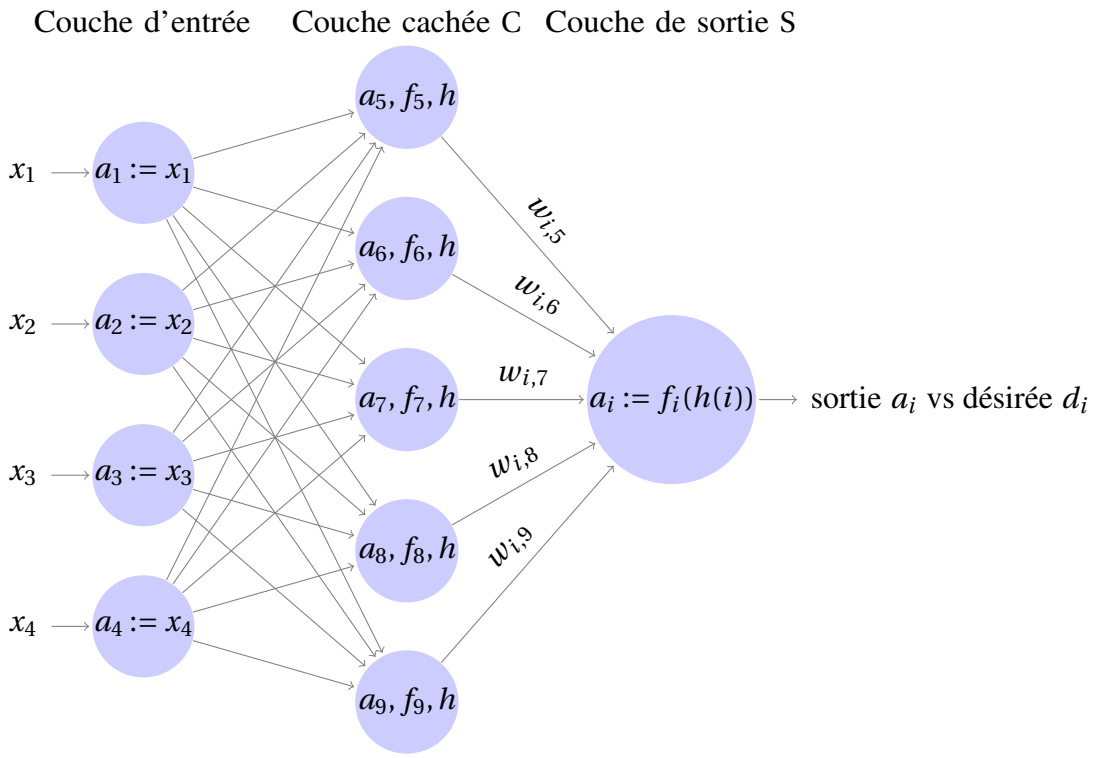
avec $w_{i,0} = \Theta_i$, $a_0 = -1$ et la fonction d'activation est l'identité. L'apprentissage se réalise par l'algorithme de Windrow et Hoff décrit pour le perceptron au paragraphe 2.5.

4. Le Perceptron Multi-Couches dit PMC

Les PMC sont des "approximateurs universels de fonctions". Un PMC, comportant une couche cachée avec une fonction de transfert non linéaire et une couche de sortie linéaire, suffit à approximer toute fonction continue à support borné (voir [**4**]). Ses applications sont nombreuses : reconnaissance des formes, la classification, le contrôle automatique de processus.

4.1. Architecture.

Le PMC se présente en 3 (ou plus) couches : la couche d'entrée, une couche cachée (ou des couches cachées) et la couche de sortie.



L'activation se propage de l'entrée vers la couche cachée puis de la couche cachée vers la couche de sortie.

La fonction d'entrée du neurone i est linéaire pour la couche cachée et la couche de sortie :

$$h(i) = \sum_{j=1}^{N} w_{i,j} a_j$$

et la fonction d'activation f_i pour la couche cachée est une sigmoïde afin que les couches cachées soient utiles (voir le paragraphe sur le perceptron) et que l'on puisse dériver. La couche de sortie est plutôt linéaire (i.e. $f_i = id$).

Pour la correction des poids synaptiques lors de l'apprentissage, le parcours se fera en sens inverse : correction des poids $w_{i,j}$ pour les neurones de sortie i et les neurones cachés j puis correction des poids $w_{i,j}$ pour les neurones cachés i et les neurones d'entrée j. Pour connaître l'erreur des couches cachées, il faut connaître celles des neurones auxquels elles sont reliées. Cette méthode de propagation des corrections des poids synaptiques est appelée la *rétropropagation*.

Comme il n'y a pas de boucle (réseau non récurrent), ces parcours sont réalisés en une passe et les temps sont alors proportionnels au nombre de liens synaptiques.

4.2. Apprentissage.

La règle de modification des poids reste

$$\Delta w_{i,j} = -\lambda \frac{\partial \mathrm{E}}{\partial w_{i,j}} \tag{21}$$

où j désigne donc un neurone de la couche précédente à celle du neurone i. La fonction d'erreur E calculée sur les neurones de la couche de sortie S est identique à celle du perceptron, c'est l'erreur quadratique :

$$\mathrm{E} = 1/2 \sum_{i \in \mathrm{S}} (a_i - d_i)^2 .$$

De sorte que, pour chaque neurone de sortie $i \in$ S, la règle dite *règle delta généralisée* d'apprentissage est donnée par (voir Identité (20) du paragraphe sur le perceptron) :

$$\Delta w_{i,j} = \lambda . a_j . (d_i - a_i) . f'(h(i)) . \tag{22}$$

Maintenant cherchons la règle d'apprentissage sur les neurones d'une couche cachée C. Pour appliquer la modification des poids (21), il n'y a qu'à calculer $\frac{\partial \mathrm{E}}{\partial w_{i,j}}$ où $i \in$ C en fonction de nos données. Nous procédons de la même manière que pour le calcul sur la couche de sortie décrit dans le paragraphe sur le perceptron :

$$\frac{\partial \mathrm{E}}{\partial w_{i,j}} = \frac{\partial \mathrm{E}}{\partial h(i)} . \frac{\partial h(i)}{\partial w_{i,j}} = a_j \frac{\partial \mathrm{E}}{\partial h(i)} \quad .$$

Nous devons donc calculer les dérivées $\frac{\partial \mathrm{E}}{\partial h(i)}$ de l'erreur sur les neurones i de la couche cachée C. Modifions la variable de sommation de notre fonction d'erreur calculée sur la couche de sortie afin de rendre la présentation plus claire :

$$\mathrm{E} \quad = \quad 1/2 \sum_{k \in \mathrm{S}} (a_k - d_k)^2$$

et remarquons que pour tout $k \in$ S :

$$1/2 \frac{\partial (a_k - d_k)^2}{\partial h(k)} = \frac{\partial \mathrm{E}}{\partial h(k)} \quad .$$

D'où puisque $a_i = f(h(i))$:

$$\frac{\partial \mathrm{E}}{\partial h(i)} \quad = \quad 1/2 \sum_{k \in \mathrm{S}} \frac{\partial (a_k - d_k)^2}{\partial h(i)} = \sum_{k \in \mathrm{S}} \frac{\partial \mathrm{E}}{\partial h(k)} . \frac{\partial h(k)}{\partial a_i} . \frac{df(h(i))}{dh(i)} \quad .$$

Comme d'après l'identité (17),

$$\frac{\partial \mathrm{E}}{\partial h(k)} = (a_k - d_k) f'(h(k)) \quad \text{pour tout } k \in \mathrm{S},$$

nous obtenons finalement :

$$\frac{\partial E}{\partial h(i)} = f'(h(i)) \sum_{k \in S} (a_k - d_k) f'(h(k)) w_{k,i} \quad .$$

car $h(k) = \sum_{j \in C} w_{k,j} a_j$.

Donc pour tout i dans la couche cachée C, nous avons :

(23)
$$\begin{aligned} \frac{\partial E}{\partial w_{i,j}} &= \frac{\partial E}{\partial h(i)} . \frac{\partial h(i)}{\partial w_{i,j}} \\ &= a_j f'(h(i)) \sum_{k \in S} w_{k,i} (a_k - d_k) f'(h(k)) \quad . \end{aligned}$$

et par conséquent :

(24)
$$\Delta w_{i,j} = \lambda . a_j . f'(h(i)) \sum_{k \in S} w_{k,i} (d_k - a_k) . f'(h(k)) \quad .$$

La démonstration permet de généraliser à plusieurs couches. Pour tout couple (i, j), où le neurone j appartient à la couche précédente de celle à laquelle appartient le neurone i :

(25)
$$\Delta w_{i,j} = \lambda . a_j \, err_i$$

où

(26)
$$err_i = (d_i - a_i) f'(h(i)) \quad \text{si } i \text{ est un neurone de sortie et}$$

(27)
$$err_i = f'(h(i)) \sum_k w_{k,i} \, err_k \quad \text{si } i \text{ est un neurone d'une couche cachée}$$

et où k parcourt les neurones de la couche suivante de celle du neurone i.

Nous pourrons appliquer ces formules à $f = f_e$ avec $f'_e(x) = f_e(x).(1 - f_e(x))$ pour la sigmoïde exponentielle et à $f = f_t$ avec $f'_t(x) = 1 - f_t(x)^2$ pour la sigmoïde tangentielle (voir Exemples 1.4 et 1.5 Paragraphe 1.5 Chapitre 1). Nous avons alors

(28)
$$f'_e(h(i)) = a_i.(1 - a_i) \text{ et } f'_t(h(i)) = 1 - a_i^2 \quad .$$

Note Lors de ce cours, sera expliqué le problème des minimas locaux. Le choix du nombre de neurones de la couche cachée sera traité lors de la séance sur machine.

4.3. Erreur quadratique vs corrélation croisée.

La fonction la plus utilisée est l'erreur quadratique. Pourtant elle possède un inconvénient majeur qui est d'avoir une dérivée qui s'annule en plusieurs autre valeur de a_i que la valeur désirée par le patron. Par exemple, pour la sigmoïde exponentielle comme fonction d'activation, en sortie nous obtenons la correction (voir Identité (22)) :

(29) $$\Delta w_{i,j} = \lambda.a_j.(d_i - a_i).a_i.(1 - a_i)$$

provenant de la dérivée de l'erreur qui s'annule en les trois valeurs 0,1,d_i de a_i. Or seule la dernière valeur pour laquelle $a_i = d_i$ présente un intérét pour l'apprentissage.

Lorsque les données en sortie ne sont pas binaires, dans la pratique la fonction de corrélation croisée l'emporte souvent sur la quadratique (voir aussi Paragraphe 3.3 Chapitre 1). En effet, la fonction de corrélation croisée sur la sortie du réseau :

(30) $$\mathrm{E} = -\sum_{i \in \mathrm{S}} (d_i ln(a_i) + (1 - d_i) ln(1 - a_i))$$

a pour dérivée partielle par rapport à a_i :

(31) $$\frac{\partial \mathrm{E}}{\partial a_i} = \frac{d_i - a_i}{a_i(a_i - 1)}$$

puisque

$$\frac{\partial \mathrm{E}}{\partial a_i} = -(\frac{d_i}{a_i} - \frac{1 - d_i}{1 - a_i}) = \frac{d_i(1 - a_i) - (1 - d_i)a_i}{a_i(a_i - 1)} \quad .$$

5. Réseaux Radial Basis Function dits RBF

Les réseaux RBF furent développés par Powell en 1985 et les premières utilisations réalisées par Broomhead et Lowe en 1988.

Ce modèle fait parti des réseaux de neurones supervisés et la mise en place de son premier algorithme d'apprentissage est devenu une alternative au PMC. Il est connu pour être plus rapide et souvent plus efficace que ce dernier. Ses paramètres sont plus aisés à ajuster que ceux du PMC. Il comporte moins de risque que le PMC de convergence vers un minimum local de la fonction d'erreur.

Les réseaux RBF perdent néanmoins de leur efficacité lorsque les données vivent dans des espaces de grande dimension (ci-dessous : d grand) ou sur des données très bruitées. Ils ont de moins bonnes capacité de généralisation que le PMC, surtout si le corpus ne recouvre pas tous les cas.

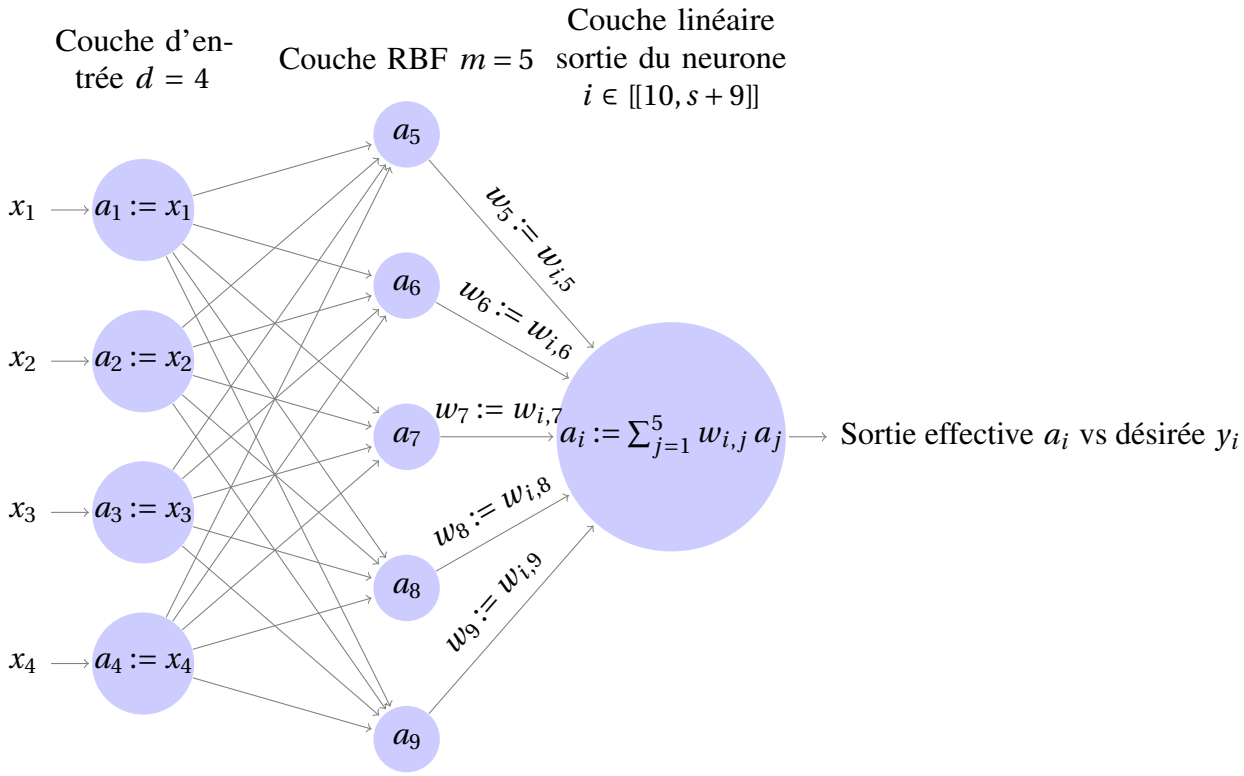
Si au début ils ont été conçus pour des applications de classification, nous pouvons les retrouver dans des domaines comme l'interpolation de fonctions, reconnaissance de la parole, prévision de signal, classification, etc

Architecture

L'architecture du réseau comporte trois couches :

- une couche d'entrée formée de d neurones avec, comme pour le PMC, la fonction d'activation identité,
- une couche cachée formée de m neurones RBF opérant avec des *fonctions radiales de base*,

— et une couche de sortie linéaire (ou affine) formée de s neurones.



Avertissement au lecteur : Afin d'alléger les notations et de les uniformiser avec les centres, plutôt que de noter a_{i+d}, $i \in [[1, m]]$, les activations pour la couche cachée et a_{j+d+m}, $j \in [[1, s]]$, (ou bien a_j, $j \in [[1+d+m, s+d+m]]$) celles de la couche de sortie (voir Dessin ci-dessus), par abus, nous les noterons désormais simplement a_i, $i \in [[1, m]]$ pour la couche cachée, et a_i, $i \in [[1, s]]$ pour la couche de sortie, tout en précisant la couche pour éviter les confusions. Nous ferons de même pour les poids.

Nous considérons des fonctions à noyaux locales qui apportent des réponses utiles pour un champ récepteur. Un tel champ est défini autour d'un point, appelé le noyau. La réponse du réseau doit être maximale au noyau et décroître monotonement avec la distance.

Pour chaque neurone i de la couche RBF ($i \in [[1, m]]$), notons $\mathbf{c}^i \in \mathbb{R}^d$ son **noyau** et pour tout $\mathbf{x} \in \mathbb{R}^d$, notons

$$\delta_i(\mathbf{x}) := \sqrt{\sum_{j=1}^{d} (x_j - c_j^i)^2} = ||\mathbf{x} - \mathbf{c}^i|| \,.$$

la distance de $\mathbf{x}$ au noyau $\mathbf{c}^i$.

Définition 5.1. Une ψ une fonction de $\mathbb{R}^d$ dans $\mathbb{R}$ est dite *radiale de base* ("radial basic function" en anglais) si elle est symétrique par rapport à un point de $\mathbb{R}^d$ pour la norme euclidienne, c'est-à-dire

qu'il existe un point $\mathbf{c}$ de $\mathbb{R}^d$:

$$\forall (\mathbf{x}, \mathbf{y}) \in \mathbb{R}^d \times \mathbb{R}^d \qquad \|\mathbf{x} - \mathbf{c}\| = \|\mathbf{y} - \mathbf{c}\| \Longrightarrow \psi(\mathbf{x}) = \psi(\mathbf{y}) \quad .$$

Un tel point $\mathbf{c}$ est appelé le *centre* ou le *noyau* de la fonction ψ.

La fonction radiale de base de $\mathbb{R}^d$ dans $\mathbb{R}$ la plus utilisée en réseau RBF provient des fonctions gaussiennes sur $\mathbb{R}$. A une facteur multiplicatif prés, elles prennent la forme de la fonction radiale de base suivante :

$$\psi_i(\mathbf{x}) := e^{-\frac{\delta_i(\mathbf{x})^2}{\sigma_i^2}} \quad , \quad \mathbf{x} \in \mathbb{R}^d, \tag{32}$$

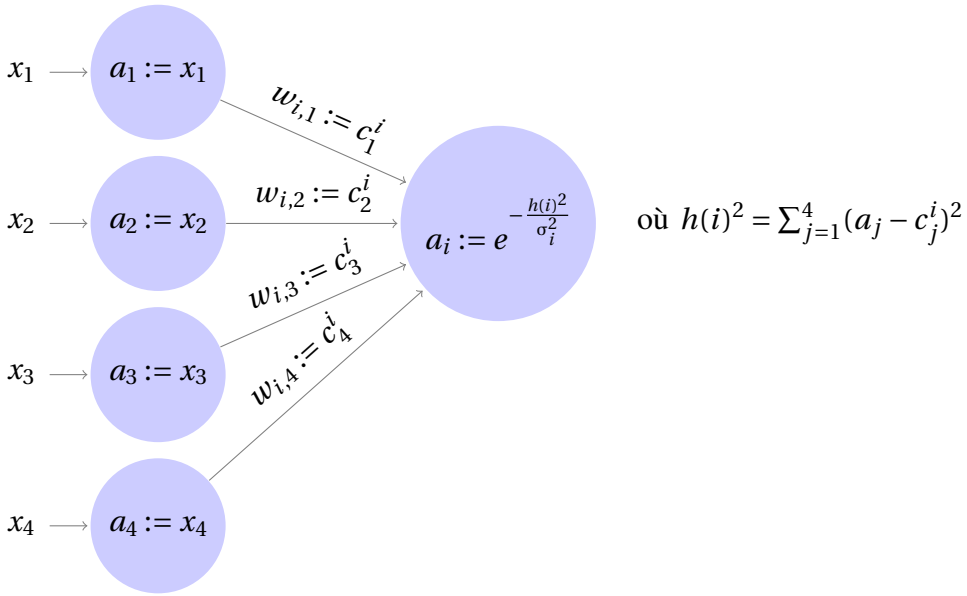
où σ_i est la taille du champ récepteur, la *largeur* de la fonction. Pour une entrée donnée, la sortie du neurone RBF est la hauteur de la gaussienne en ce point. La fonction gaussienne permet aux neurones de ne répondre qu'à une petite région de l'espace d'entrée, région sur laquelle la gaussienne est centrée. Elle permet d'estimer la densité de probabilité des données d'entrées.

Ainsi,comme nous l'avons vu dans l'exemple 1.6 du chapitre 1, nous pouvons considérer que le vecteur $\mathbf{x}$ de $\mathbb{R}^d$ est le vecteur $\mathbf{a}_e$ des activations $a_1, \ldots, a_d$ des d neurones de la couche d'entrée du réseau auxquels ces valeurs sont affectées et que $\boldsymbol{c}^i$ est le vecteur des valeurs des poids $w_{i,j} := c_j^i$, $j \in [\![1, d]\!]$ partant des d neurones de la couche d'entrée et arrivant au neurone i de la couche RBF. La fonction d'entrée $h(i)$ du neurone i nous est donnée par la distance δ_i avec $h(i) := \delta_i(\boldsymbol{a}_e)$:

$$h(i) = \sqrt{\sum_{j=1}^{d} (a_j - c_j^i)} = \left\| \boldsymbol{a}_e - \boldsymbol{c}^i \right\|$$

où $\boldsymbol{a}_e^t := (a_1, \ldots, a_d)$ est le transposé du vecteur d'activation de la couche d'entrée. et sa fonction d'activation est la gaussienne

$$f_i(x) = e^{-\frac{x^2}{\sigma_i^2}}, \quad x \in \mathbb{R} \quad .$$



Note (Rappel) : Dans la pratique, nous pourrons aussi choisir $h(i)^2 = \left\|\mathbf{a} - \boldsymbol{c}^i\right\|^2$ comme fonction d'entrée avec la fonction d'activation $g(x) = e^{-x/\sigma^2}$ (qui n'est pas une gaussienne). En effet, afin d'éviter de considérer le couple (f, h) où $h(i)$ est une racine carrée, nous pouvons choisir le couple (g, h^2), ce qui revient exactement au même puisque $a_i = f(h(i)) = g(h(i)^2)$.

Apprentissage

Nous supposons disposer d'un corpus $\mathscr{C}$ de cardinal n constitué de doublets

$$(\mathbf{x}^r, \mathbf{y}^r) \in \mathbb{R}^d \times \mathbb{R}^s, \quad r = 1, \dots n \quad .$$

Nous devons adapter notre réseau à ce corpus en rentrant successivement comme pour le PMC les d valeurs de chaque vecteur $\mathbf{x}^r$ et en comparant la sortie aux s valeurs attendues du vecteur $\mathbf{y}^r$ pour appliquer la règle d'apprentissage choisie sur les poids.

après avoir vu l'architecture générale du réseau, nous avons encore à règler les paramètres variables dans notre réseau qui sont :

1. le nombre m de neurones RBF dans la couche cachée,
2. la position des m centres des gaussiennes ; i.e. des poids entre les d neurones de la couche d'entrée et les m de la couche RBF,
3. la largeur σ_i de chaque gaussienne,
4. les poids des connexions entre les m neurones RBF et les s neurones de sortie (apprentissage proprement dit).

1. et 2. *Nombre m de neurones RBF dans la couche cachée et position des centres*

Rappelons que les centres sont en fait les poids de la couche d'entrée vers la couche RBF.

Si le nombre n d'exemples du corpus n'est pas trop élevé, on prend autant de neurones RBF que d'exemples, i.e. $m = n$,les gaussiennes sont alors centrées sur les exemples donnés en entrée ; c'est-à-dire que $\mathbf{c}^i := \mathbf{x}^i$ où $\mathbf{x}^i$ est extrait du i-ème vecteur $(\mathbf{x}^i, \mathbf{y}^i)$ du corpus .

Dans le cas contraire, nous prenons $m << n$ et déterminer m devient un véritable problème. Il n'existe pas vraiment de méthodes pour proposer une bonne valeur. Une fois une décision prise, il va falloir calculer les centres. Pour cela différentes options s'offrent à nous :

— un choix aléatoire des centres parmi les exemples avec un risque de mauvais choix,
— utiliser des méthodess plus performantes comme l'algorithme des K-moyennes, la quantification vectorielle (LVQ),
— d'autres proposent aussi d'utiliser les cartes de Kohonen.

Pour la classification, les centres sont choisis avec l'algorithme des K-moyennes. Cet algorithme sépare les n vecteurs d'entrées $\mathbf{x}^i$ du corpus en m différents "clusters", chacun représenté par un centre. L'algorithme des K-moyennes, avec ici K = m est le suivant :

Algorithme des K-**moyennes**
Entrees : Les n vecteurs $\boldsymbol{x}^r$, $r = 1, \ldots, n$ provenant du corpus $\mathscr{C}$
le nombre de centres m, avec $m < n$
une borne M > 1.
Sortie : Les centres $\boldsymbol{c}^1, \ldots, \boldsymbol{c}^m$.
compteur := 1
Pour chaque cluster C_i, $i \in [\![1, m]\!]$
choisir pour le entre $\boldsymbol{c}^i$ une valeur au hasard parmi les $\boldsymbol{x}^r$
Tant que il n'y a pas convergence **Et** compteur < M **Faire**
Attribuer a chaque cluster les vecteurs $\boldsymbol{x}^r$ les plus proches du centre
Pour chaque cluster C_i, m_i := Moyenne($\boldsymbol{x}^r \mid \boldsymbol{x}^r \in C_i$) **Et** $\boldsymbol{c}^i := m_i$
compteur := compteur +1
Retourner $\boldsymbol{c}^1, \ldots, \boldsymbol{c}^m$

L'inconvénient de cette méthodes est la nécessité de la connaissance a priori du nombre m de clusters, ce qui n'est pas le cas avec les cartes de Kohonen.

3. *Largeur des gaussiennes*

Une fois les centres "appris", nous pouvons déterminer les largeurs des gaussiennes qui en dépendent.

Les 2 règles connues pour affecter des valeurs à ces largeurs sont :

$$\sigma = \frac{d}{\sqrt{m}} \quad \text{où } d = \max_{1 \le i,j \le m} \left\| \mathbf{c}^i - \mathbf{c}^j \right\|$$

qui affecte une variance identique à toutes les gaussiennes, et

$$\sigma_j = \frac{1}{n\sqrt{8}} \sum_{i=1}^{n} \left\| \mathbf{x}^i - \mathbf{c}^j \right\| \quad .$$

4. *Poids de connexions entre les neurones RBF et ceux de sortie*

Maintenant que nous avons déterminé les centres et les largeurs des gaussiennes, nous pouvons passer au choix des poids reliant les neurones RBF et ceux de sortie.

Fixons un neurone de sortie, disons le i-ème, avec $i \in [\![1, s]\!]$.

Soit le vecteur $\mathbf{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_m \end{pmatrix}$ des poids arrivant à notre neurone i de la couche de sortie (i.e. $w_j := w_{i,j}$, $i \in [\![1, s]\!]$). Ce sont ces valeurs que le réseau doit apprendre (pour chaque neurone de sortie) afin de s'adapter à l'application, c'est-à-dire au corpus.

Soit $(\mathbf{x}, \mathbf{y}) \in \mathscr{C}$, un doublet du corpus. Lorsque $\mathbf{x}$ est donné en entrée, l'activation du i-ème neurone en sortie prend pour valeur

$$a_i := (\psi_1(\mathbf{x}), \ldots, \psi_m(\mathbf{x})).\mathbf{w}$$

et c'est la i-ème composante y_i du vecteur $\mathbf{y}$ qui est attendue.

Regardons ce que cela donne toujours sur le i-ème neurone de sortie et en prenant non pas un seul mais la totalité des n exemples du corpus.

Notons $\mathrm{Y}_i = \begin{pmatrix} y_i^1 \\ \vdots \\ y_i^n \end{pmatrix}$ le vecteur des i-èmes coordonnées des n vecteurs $\mathbf{y}^r$ des doublets $(\mathbf{x}^r, \mathbf{y}^r)$ du corpus $\mathscr{C} \subset \mathbb{R}^d \times \mathbb{R}^s$ (nous généralisons cette notation aux s autres coordonnées).

Soit encore la matrice $n \times m$ représentant la partie RBF des n simulations (expériences) sur le réseau à partir des n exemples du corpus :

$$\mathrm{A} \quad = \quad (\psi_j(\mathbf{x}^r))_{1 \le r \le n, 1 \le j \le m} = \begin{pmatrix} \psi_1(\mathbf{x^1}) & \ldots & \psi_m(\mathbf{x}^1) \\ \vdots & \vdots & \vdots \\ \psi_1(\mathbf{x^n}) & \ldots & \psi_m(\mathbf{x}^n) \end{pmatrix} \quad ;$$

i.e. pour l'entrée $\mathbf{x^r}$ de la r-ième expérience, $\psi_1(\mathbf{x^r})$ est l'activation du neurone 1 de la couche cachée, …, $\psi_m(\mathbf{x^r})$ est celle de m-ième ; la r-ième ligne est le transposée du vecteur d'activation de la couche cachée.

Les n expériences produisent en sortie le vecteur A.$\mathbf{w}$ sur le neurone i qu'il s'agit de comparer au vecteur Y_i attendu. Pour trouver le vecteur $\mathbf{w}$ des poids partant de la couche RBF et arrivant au neurone i, il s'agirait donc de résoudre l'équation

$$\mathrm{A}\cdot\mathbf{w} = \mathrm{Y}_i\,;$$

ce qui revient à inverser la matrice A. La matrice A n'étant pas nécessairement inversible, il va falloir passer par l'apprentissage sur les neurones de sortie décrite pour les PMC.

Remarquons qu'il s'agit de déterminer colonne par colonne la matrice des poids W qui satisfait l'équation matricielle :

$$\mathrm{A}\cdot\mathrm{W} = (\mathrm{Y}_1,\ldots,\mathrm{Y}_n)$$

où chaque Y_j, faisant référence au j-ième neurone en sortie, est défini de la même manière que Y_i.

Puisque la dernière couche est linéaire, il s'agit donc d'appliquer la règle DELTA dans l'algorithme de Windrow-Hoff vu au paragraphe 2.5 à adapter à la couche RBF. Prenons encore $(\mathbf{x},\mathbf{y}) \in \mathscr{C}$. La correction du poids $w_j = w_{i,j}$ partant du neurone j de la couche RFB et arrivant au neurone i de sortie est

$$w_j(t+1) = w_j(t) + \lambda\,(y_i - a_i)\,a_j$$

où $a_i = (\psi_1(\mathbf{x}),\ldots,\psi_m(\mathbf{x}))$.$\mathbf{w}$ et $a_j = \psi_j(\mathbf{x})$ sont les valeurs calculées des activations respectives des neurones i et j calculées par le réseau avec $\mathbf{x}$ en entrée associé au vecteur désiré $\mathbf{y}$ dans le corpus.

Cas $d = s = 1$.

Nous cherchons à approcher une fonction F de R dans R.

Nous avons $\mathbf{x}^r = x^r \in \mathbb{R}$ et $\mathbf{y}^r = y^r = \mathrm{F}(x^r) \in \mathrm{R}$ pour $r \in [\![1,n]\!]$, et $\mathbf{c}^j = c^j \in \mathbb{R}$ pour $j = 1,\ldots,m$. Nous notons $w_1,\ldots,w_m$ les poids des m neurones de la couche RBF vers l'unique neurone de sortie.

Pour tout couple $(x,y) \in \mathscr{C} \subset \mathbb{R}^2$ tel que $y = \mathrm{F}(x)$, la sortie du réseau est

$$\hat{\mathrm{F}}(x) = \sum_{j=1}^{m} w_j \psi_j(x)$$

où $\psi_j(x) = e^{-\frac{(x-c^j)^2}{\sigma_j^2}} = g(\frac{x-c^j}{\sigma_j})$ avec $g(x) = e^{-x^2}$ (nous modifions légèrement nos notations pour $d = s = 1$).

Notons que dans le cas particulier $n = m$, $c^j := x^j$, $\sigma_j = \sigma$ et $w_j = \frac{y^j}{\sum_{j=1}^{m}\psi_j(x)}$ pour $j = 1,\ldots,m = n$. Nous retrouvons alors l'estimateur de type Nadaraya-Watson :

$$\hat{F}(x) = \frac{\sum_{j=1}^{n} y^j g(\frac{x-x^j}{\sigma})}{\sum_{j=1}^{n} g(\frac{x-x^j}{\sigma})} \quad .$$

6. Connexions symétriques

Nous nous intéressons ici au travail de Hopfield (1982) pour modéliser les “verres de spin de Ising ” (physique statistique). Le lecteur pourra aussi s’intéresser au travail de Ackley, Seijnouski et Hinton (1985) sur la machine de Boltzmann.

Nous avons déjà abordé les réseaux symétriques dans l’exemple 3.1 du chapitre 1.

Le modèle ne comporte pas de couche cachée et s’applique aux mémoires associatives ou à la résolution avec contraintes.

Le réseau est supposé comporter N neurones. Pour chaque neurone $i \in [1,N]$, l’activation a_i représente le spin de l’atome i et sa valeur est 0 ou 1. Pour chaque $i \in [1,N]$, la relation qui relie s_i la valeur du spin (ou le moment magnétique) et sa représentation par a_i est donnée par

$$s_i = 2a_i - 1 \quad ;$$

et s_i prend les valeurs -1 et 1 lï¿½ où a_i prend respectivement les valeurs 0 et 1. Un seuil Θ_i est attribué au neurone i.

La matrice des poids synaptiques W est symétrique. Chacun de ses coefficients $w_{i,j}$ (avec $i, j \in [1,N]$) représente l’effet de l’atome i sur l’atome j avec $w_{i,i} = 0$ et aussi, de part la symétrie, $w_{i,j} = w_{j,i}$.

La contribution de l’atome i au champ est représentée par la fonction d’entrée linéaire $h(i) = \sum_{j\in[1,N]} w_{i,j} a_j$.

Les fonctions d’activation de tous les neurones sont toutes identiques à la fonction d’activation binaire f qui s’applique sur $h(i)$ définie par :

$$f(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{sinon} \end{cases}$$

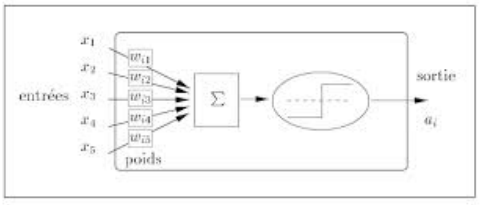


FIG. 2.1 – *Un neurone formel.*

Le réseau peut se retrouver dans 2^N états $(s_1^p, \ldots, s_N^p)$ avec s_i^p = -1 ou 1.

6.1. La règle d'apprentissage de Hebb.

La règle d'apprentissage de Hebb (1949) (voir Paragraphe 1.1) s'applique pour aboutir à un état stable pourvu que les patrons du corpus P ne soient pas biaisés et soient décorrélés. C'est-à-dire que pour tous patrons p et q du corpus P :

$$\sum_{i=1}^{\mathrm{N}} s_i^p = 0 \text{ et, pour } p \neq q, \sum_{i=1}^{\mathrm{N}} s_i^p s_i^q = 0 \,. \tag{33}$$

La règle de Hebb pour les réseaux symétriques est :

$$w_{i,j} = \frac{1}{\mathrm{N}} \sum_{p \in \mathrm{P}} s_i^p s_j^p \,.$$

Cette règle s'applique de deux manières : (1) pour calculer directement les poids et (2) par pas successif avec la règle $w_{i,i} = 0$ et :

$$\Delta w_{i,j}(t) = \frac{1}{\mathrm{N}} s_i(t) s_j(t) \text{ avec } i, j \in [1, \mathrm{N}] \,, i \neq j \,.$$

Montrons que lorsque la stabilité sera atteinte alors $s_i.h(i) \geq 0$ pour tout $i \in [1, \mathrm{N}]$. Avec la règle de Hebb nous avons :

$$h(i) = \frac{1}{\mathrm{N}} \sum_{j=1}^{\mathrm{N}} (\sum_{p \in \mathrm{P}} s_i^p s_j^p) a_j \,.$$

Supposons que le réseau soit dans un état appris q : $a_j := a_j^q$ pour $j \in [\![1, \mathrm{N}]\!]$ (de même $s_i := s_i^q$). Puisque $2a_j = s_j + 1$, nous avons

$$\begin{aligned} 2\mathrm{N}.s_i.h(i) &= s_i^q \sum_{j=1}^{\mathrm{N}} (\sum_{p \in \mathrm{P}} s_i^p s_j^p).(s_j^q + 1) \\ &= \sum_{j=1}^{\mathrm{N}} \sum_{p \in \mathrm{P}} s_i^q s_i^p s_j^p.(s_j^q + 1) \\ &= \sum_{p \in \mathrm{P}} (s_i^q s_i^p \sum_{j=1}^{\mathrm{N}} s_j^p.(s_j^q + 1)) \quad . \end{aligned}$$

D'où, d'après (33), :

$$2\mathrm{N}.s_i.h(i) = s_i^q s_i^q \sum_{j=1}^{\mathrm{N}} s_j^q.s_j^q = (s_i^q)^2 \sum_{j=1}^{\mathrm{N}} (s_j^q)^2 > 0 \qquad .$$

puisque $\forall i \; \forall p \quad s_i^p \neq 0$.

7. Réseaux à compétition

Il en existe de trois types importants : LVQ (Kohonen, 1984), SOM (Kohonen) et le réseau ART de Grossberg. Ils interviennent en analyse des données et en classification.

Pour qu'une couche soit de compétition, elle doit répondre à trois critères :

1. les neurones répondent différement aux diverses entrées
2. les fonctions d'activation sont bornées
3. il existe un mécanisme de concurrence entre les neurones réalisé par des connexions inhibitrices.

Les poids inhibiteurs sont égaux pour une bonne compétition.

L'utilisation est la suivante :

— présenter le patron d'entrée et récupérer la sortie,
— compétition entre les neurones de sortie,
— les plus activés sont les vainqueurs.

En dehors de deux exceptions, dont SOM, ce sont les vainqueurs qui sont entraînés et en général l'apprentissage est non supervisé (règle de Hebb, par exemple).

La capacité de discrimination dépend de la distribution des données à apprendre et à interpréter. Si elles ne se regroupent pas en nuage, la méthode ne marchera pas.

Nous expliquons le principe de ces réseaux sur un réseau simple constitué de deux couches : une d'entrée E et une de compétition S.

Soit $\{x_j \mid j \in \mathrm{E}\}$ un ensemble de valeurs présentées à cette couche d'entrée qui constitue le patron (passif) :

$$a_j := x_j \qquad \text{pour } j \in \mathrm{E}\,.$$

Pour i un neurone de la couche de compétition, l'activation pondérée est donnée par

$$e_i = h(i) = \sum_{j \in \mathrm{E}} w_{i,j} a_j = \mid \boldsymbol{x} \mid . \mid \mathrm{W}_i \mid . cos(\theta_i)$$

où θ_i est l'angle entre $\boldsymbol{x}$, le vecteur d'entrée, et W_i, le vecteur des poids arrivant au neurone i. En normalisant les données avec $\mid \boldsymbol{x} \mid = \mid \mathrm{W}_i \mid = 1$ pour tout $i \in \mathrm{S}$, nous obtenons $e_i = cos(\theta_i)$.

Si la fonction d'activation est monotone croissante alors le gagnant est $a_c = max_{i \in \mathrm{S}}(a_i)$.

Le neurone vainqueur agit comme un détecteur de certains types d'entrées. Une des règles due a Grossberg (1976) est la suivante : il s'agit de redistribuer les poids W_c d'entrée du vainqueur c de façon à le rapprocher de chaque entrée x_j, $j \in \mathrm{E}$: pour $i \neq c$ $\Delta w_{i,j} = 0$ et

$$\Delta w_{c,j} = \lambda(\frac{x_j}{\sum_{k \in \mathrm{E}} x_k} - w_{c,j}) \quad .$$

L'effet est de réduire l'angle entre W_c et $\boldsymbol{x}$. Si la somme $\sum_{j \in \mathrm{E}} w_{c,j}$ des poids d'entrée du neurone c est 1, elle reste à 1. En effet, dans ce cas :

$$\sum_{j\in \mathrm{E}} \Delta w_{c,j} = \sum_{j\in \mathrm{E}} \lambda(\frac{x_j}{\sum_{k\in \mathrm{E}} x_k} - w_{c,j}) = \lambda(\frac{\sum_{k\in \mathrm{E}} x_k}{\sum_{k\in \mathrm{E}} x_k} - \sum_{j\in \mathrm{E}} w_{c,j}) = 0 \quad .$$

8. La Machine de Boltzmann Restreinte, RBM

Une RBM (”Restricted Boltzmann Machine”) est un réseau neuronal stochastique génératif destiné à apprendre une distribution de probabilité sur ses entrées.

Architecture :
C’est un réseau composé de couches ”visibles‘” et ”cachées”. Il est symétrique (la propagation est dirigée dans un sens ou dans l’autre avec le même poids), entièrement connecté sous la restriction suivante : nous ne permettons pas à toutes les couches d’être ainsi inter-connectées (sinon ce serait une Machine de Bolzmann non restreinte).

Fonctions d’entrée h et d’activation f :
Nous considérons le RBM à unités binaires sous une distribution de Bernouilli : $f(x)$:=Prob(X = x) = $p^x(1-p)^{1-x}$, pour $x \in \{0,1\}$. La fonction d’entrée possédant un seuil : au-dessus du seuil $h(i) = 1$, en dessous, $h(i) = 0$.

Le corpus : $\mathscr{C}$.

Apprentissage : non supervisé.

Nous choisissons 2 couches : 1 visible et 1 cachée.

Nous présentons $\boldsymbol{x} \in \mathscr{C}$ à la couche visible. Soit $\boldsymbol{s} = \Psi_1(\boldsymbol{x})$ la sortie de la couche cachée. Nous renvoyons $\boldsymbol{s}$ vers la couche visible : $\boldsymbol{x}_1 := \Psi_2(\boldsymbol{s}) = \Psi_2 o \Psi_1(\boldsymbol{x})$. Puis $\boldsymbol{x}_1$ est renvoyé vers la couche cachée pour obtenir $\boldsymbol{s}_1 := \Psi_1(\boldsymbol{x}_1)$. Soit $\boldsymbol{w}$ le vecteur de poids entre les deux couches (nous rappelons que les connexions sont symétriques). La correction des poids est donnée par la règle d’apprentissage suivante : :

$$\Delta(\boldsymbol{w}(t)) = \lambda(< \boldsymbol{x}, \boldsymbol{s}^{'} > - < \boldsymbol{x}_1, \boldsymbol{s}_1^{'} >)$$

où λ est le pas d’apprentissage. Cet algorithme d’apprentissage non supervisé est dit à *divergence contrastive*.

Chapitre 3

Corrélation en cascades et neurochirurgien optimal (OBS)

Comme nous l'avons évoqué au chapitre 1, la corrélation en cascades est due à Fahlman et Lebière (1990). Cette méthodes consiste à réduire l'erreur en rajoutant des neurones. Le neurochirurgien optimal est l'inverse de la corrélation en cascade (voir Le Cun et al., 1990 ; Hassibi et al., 1993). L'idée est de retrancher des neurones à un réseau déjà entrainé.

1. Corrélation en cascades

Elle utilise le QuickProp (Fahlman, 1988) (voir 1.2 chapitre 2) et cherche à maximiser, pour un certain neurone supplémentaire, la covariance entre son activation et l'erreur .

On choisit le neurone à ajouter dans une famille de candidats. Le critère pour sélectionner ce candidat est la maximisation de la covariance, sur le corpus d'apprentissage, entre la sortie du neurone candidat (i.e. son activation) et la fonction d'erreur des neurones de sortie. Soit $\boldsymbol{x}$ un individu du corpus d'apprentissage et :

$$\mathrm{C} = \sum_{\boldsymbol{x}} \sum_{j} (\mathrm{E}_{\boldsymbol{x},j} - \overline{\mathrm{E}_j})(a^{\boldsymbol{x}} - \overline{a})$$

— $\mathrm{E}_{\boldsymbol{x},j}$ est l'erreur en sortie du neurone j pour l'individu $\boldsymbol{x}$,
— $a^{\boldsymbol{x}}$ est la valeur d'activation du neurone candidat pour l'individu $\boldsymbol{x}$,
— $\overline{\mathrm{E}_j}$ et $\overline{a}$ sont les moyennes des valeurs précédentes quand $\boldsymbol{x}$ parcourt tout le corpus.

En modifiant les poids arrivant au nouveau neurone k de la même manière que pour la rétro-propagation, on peut ensuite maximiser C avec la règle

$$\Delta w_{k,j} = -\lambda \frac{\partial \mathrm{C}}{\partial w_{k,j}} \quad .$$

L'opération peut ensuite être répétée en rajoutant un autre neurone dans une couche cachée.

2. Le neurochirurgien optimal (Optimal Brain Surgeon, OBS)

Le neurochirurgien optimal (ou OBS) est l'inverse de la corrélation en cascade (voir Le Cun et al., 1990 ; Hassibi et al., 1993). Outre un gain de vitesse pour le fonctionnement du réseau, il permet d'éviter les risques de sur-apprentissage (augmentation des performances en généralisation) en réduisant la complexité du réseau de neurones par la suppression de certaines connections synaptiques. L'algorithme OBS minimise la sensibilité d'un critére d'erreur sous la contrainte de nullité des paramètres ce qui correspond à la suppression de ce paramètre.

Supposons qu'un réseau soit entrainé jusqu'à un minimum de la fonction d'erreur.

Considérant le système dynamique dont les points ont pour coordonnées les poids, nous représentons les poids sous forme d'un vecteur $\boldsymbol{w}$ et non plus de la matrice W, en ordonnant par exemple ses lignes une à une les unes derrière les autres dans $\boldsymbol{w}$ (i.e. le q-ième élément w_q de $\boldsymbol{w}$ est un élément $w_{i,j}$ de W).

On développe la fonction d'erreur E en série de Taylor jusqu'à l'ordre 2, le premier ordre étant nul par hypothèse. Le problème de cette méthode est le calcul de la matrice inverse de la hessienne

$$\mathrm{H} = \partial^2 \mathrm{E}/\partial \boldsymbol{w}^2 = (\frac{\partial^2 \mathrm{E}}{\partial w_{i,j} \partial w_{k,l}})_{1 \leq i,k \leq m, 1 \leq j,l \leq d} = (\frac{\partial^2 \mathrm{E}}{\partial w_q \partial w_p})_{1 \leq q,p \leq d.m}$$

dont la dimension est élevée : $(d.m)^2$ où d et m sont les nombres de neurones respectifs des couches concernées.

Soit le développement de l'erreur E en série de Taylor :

$$\partial \mathrm{E} = \partial \boldsymbol{w}.\frac{\partial \mathrm{E}}{\partial \boldsymbol{w}} + \frac{1}{2}{}^t\partial \boldsymbol{w}.\mathrm{H}.\partial \boldsymbol{w} + \mathrm{O}(\|\partial \boldsymbol{w}\|^3)\,.$$

En négligeant l'ordre 3 et en supposant nulle $\frac{\partial \mathrm{E}}{\partial \boldsymbol{w}}$ (puisque le réseau est bien entrainé), nous avons à étudier :

$$\partial \mathrm{E} = \frac{1}{2}{}^t\partial \boldsymbol{w}.\mathrm{H}.\partial \boldsymbol{w}\,.$$

Revenons à notre réseau. Eliminer un lien $w_q = w_{i,j}$ de j vers i signifie que $w_q(t+1) = 0$; c'est-à-dire que $0 = w_q(t) + (w_q(t+1) - w_q(t)) = w_q(t) + \Delta w_q$. Soit ${}^t e_q.(\boldsymbol{w} + \partial \boldsymbol{w}) = 0$, où ${}^t e_q = (0,\ldots,0,1,0,\ldots 0)$ est le transposé du vecteur canonique e_q.

La suppression du lien q doit conduire à un accroissement minimal de l'erreur E. Il nous faut donc trouver q et modifier les poids des autres liens telle que l'erreur soit la plus petite possible. Nous nous retrouvons avec le problème de minimisation suivant :

$$\min_q [\min_{\partial \boldsymbol{w}} (\frac{1}{2}{}^t\partial \boldsymbol{w}.\mathrm{H}.\partial \boldsymbol{w}\,;\, {}^t e_q.(\boldsymbol{w} + \partial \boldsymbol{w}))].$$

Soit le lagrangien :

$$\mathrm{L} := \frac{1}{2}{}^t\partial \boldsymbol{w}.\mathrm{H}.\partial \boldsymbol{w} + \lambda({}^t e_q.(\boldsymbol{w} + \partial \boldsymbol{w}))) \quad .$$

On calcule ses dérivées partielles $\frac{\partial \mathrm{L}}{\partial w_q}$ et $\frac{\partial \mathrm{L}}{\partial \lambda}$. La minimisation de ce Lagrangien conduit à la variation du vecteur des paramètres :

$$\partial w = -\frac{w_q}{\mathrm{H}^{-1}_{qq}} \mathrm{H}^{-1}.e_q$$

où H^{-1}_{qq} est le q-ième terme diagonal de H^{-1}. Et on obtient que l'augmentation de l'erreur si on supprime w_q est :

$$\mathrm{L}_q = \frac{1}{2}\frac{w_q^2}{\mathrm{H}^{-1}_{q,q}} \quad .$$

Si le lien q qui minimise L_q le minimise "assez", on retirera ce lien. Cette méthodes évite de recommencer l'apprentissage. L'erreur se distribuera sur les autres liens. Seulement, il faut inverser la matrice H de grande dimension et la méthodes n'a pas de test d'arrêt.

Chapitre 4

La Statistique et le PMC

1. Lien avec la statistique

Supposons que $X_1,\ldots,X_n$ soient des valeurs observées et que l'on cherche $\phi(X_1,\ldots,X_n)$ telle que

$$E(X_{n+1}|X_1,\ldots,X_n) = \phi(X_1,\ldots,X_n)\,.$$

Toute fonction $\phi(x_1,\ldots,x_n)$ suffisamment régulière peut être approchée par une fonction de la forme :

$$g(\sum_{i=1}^{n} a_i x_i + b_i)\,.$$

Ce qu'il faut chercher est l'estimation statistique $\hat{g}(X_1,\ldots,X_n)$ de g s'exprimant sous la forme :

$$\hat{g} = g(\sum_{i=1}^{n} \hat{a_i} x_i + \hat{b_i})$$

où, pour $i \in [1,n]$, les valeurs $\hat{a_i}$ et $\hat{b_i}$ sont respectivement des valeurs approchées des a_i et b_i. (Voir [**2**] et [**3**] pour une vitesse d'approximation).

Ce type de problème classique de la statistique est résolu dans des cas très généraux. L'intérêt des réseaux neuronaux intervient dans les cas de grandes dimensions.

2. Statistique et PMC

Nous avons $x_1,\ldots,x_n$ des variables observées et nous cherchons x_{n+1}. Pour $i = 1,\ldots,n+1$, notons X_i la variable aléatoire de la i-ième valeur. Posons $X = (X_1,\ldots,X_n)$ et $Y = (X_{n+1})$. Pour approcher Y, nous cherchons à construire un réseau qui approxime au mieux l'espérance $E[Y|X]$ par une fonction $f_W(X)$, où W est une matrice de poids synaptiques.

La couche d'entrée comportera n neurones (pour X) et la couche de sortie 1 neurone (pour $f_W(X)$).

Qui dit estimer, sous-entend choisir une fonction d'erreur $\pi(Y, f_W(X))$ entre la sortie obtenue $f_W(x)$ avec $x = (x_1,\ldots,x_n)$ et celle désirée $Y = (x_{n+1})$.

Une fois cette fonction d'erreur choisie, nous désirons qu'elle soit aussi petite que possible. Donc nous cherchons f_W qui minimise au mieux l'espérance $E[\pi(Y, f_W(X))]$ de la fonction d'erreur π.

La question est de savoir si approcher Y permet de trouver E[Y|X]. Cela dépend de la fonction d'erreur choisie.

Exemple 2.1. Choisissons $\pi(Y, f_W(X)) = (Y - f_W(X))^2$. Avec cette fonction d'erreur, nous calculons de manière classique

$$E[(Y - f_W(X))^2|X] = E[(Y - E[Y|X])^2|X] + E[(E[Y|X] - f_W(X))^2|X] .$$

(l'erreur totale est la somme de l'erreur de prédiction et de l'erreur statistique de prédiction). Donc minimiser l'erreur entre la sortie du réseau $f_W(X)$ et la valeur désirée Y revient à minimiser celle entre $f_W(X)$ et l'espérance E[Y|X], ce qu'on cherche à estimer.

D'autres fonctions d'erreur entre la sortie du réseau $f_W(X)$ et la valeur désirée Y partagent cette propriété d'*estimation optimale*. Par exemple, la *Cross Entropy* souvent employée dans les PMC :

$$Y log(f_W(X)) + (1 - Y) log(1 - f_W(X)) .$$

Comme nous l'avons étudié dans le PMC, nous savons que les poids optimaux sont obtenus en minimisant l'espérance d'erreur entre les calculs du réseau et les valeurs désirées obtenues à partir d'un corpus.

Chapitre 5

Apprentissage Automatique : nouvelle génération

Les principaux algorithmes de réseaux profonds (“deep learning“) sont

DNN Les réseaux de neurones profonds (Deep Neural Networks). Ce sont les réseaux PMC (Perceptron Multi-Couches) avec plusieurs de couches cachées que nous avons étudiés.

CNN Les réseaux de neurones convolutionnels (Convolutional Neural Networks). Ils sont utilisés pour la reconnaissance d'images. Le problème est divisé en zones reflétant des sous-problèmes locaux. Localement, pour chaque zone, un noyau (cluster) de neurones est créé.

DBN La machine de Boltzmann profonde (Deep Belief Network). Les algorithmes fonctionnent suivant une première phase non supervisée, suivie de l'entrainement classique PMC supervisé. La première étape non-supervisée facilite l'apprentissage supervisé sur tout le réseau.

La nouvelle génération de réseaux DBN s'intéresse à l'apprentissage automatique (machine learning). Pour simplifier, disons que l'idée est de construire le réseau automatiquement par empilement à partir de briques de bases RBM (”Restricted Boltzmann Machine”), PMC etc.

Il s'agira d'entraîner individuellement chaque brique selon la méthode d'apprentissage qui lui est propre : la couche de sortie d'une brique de base est ou bien une cachée ou bien celle de sortie du réseau complet. L'objectif de cette démarche est de conduire à une ossature, architecture (dont les poids) du réseau adaptée au plus proche de l'application avant d'entrainer tout le réseau considéré globalement comme un PMC.

Imaginons un réseau constitué par d'abord une RBM puis d'un PMC devant apprendre un corpus $\mathscr{C}$ en supervisé : la première brique est une RBM et ses sorties sont dirigées vers un PMC. Soit $(\boldsymbol{x}, \boldsymbol{d})$ du corpus d'apprentissage $\mathscr{C}$. L'apprentissage est d'abord réalisé sur la RBM uniquement avec les entrées $\boldsymbol{x}$ du corpus $\mathscr{C}$ (i.e. en non supervisé et en oubliant le PMC). Puis, soit $(\boldsymbol{x}, \boldsymbol{d})$ du corpus d'apprentissage $\mathscr{C}$. La correction des poids du PMC est l'apprentissage par la rétropropagation de l'erreur avec $\boldsymbol{d}$ comme vecteur de sortie désirée et les entrées qui lui viennent de la sortie de la RBM ayant reçu $\boldsymbol{x}$ en entrée. L'apprentissage par rétropropagation du PMC s'arrêtera donc à la couche de sortie de la RBM. Une fois le réseau ainsi construit, il s'agira d'entraîner tout le réseau tel un PMC et ne pas s'arrêter à la sortie de la RBM dans la rétropropagation de l'erreur.

Nous voyons que dans la phase construction, pour utiliser des types de réseaux comme briques de base sans que leurs sorties respectives soient communes à celle du réseau, il faut que leur apprentissage soit non supervisé. Par exemple, il est possible d'empiler des RBM et l'algorithme d'apprentissage est le suivant : apprendre la première RBM, puis apprendre la seconde avec les sorties apprises de la première, et ainsi de suite. Et l'apprentissage final des poids est ajusté globalement en voyant le réseau comme un PMC. De tels réseaux sont dits de *croyances profondes*.

Conclusion et liens utiles

Les réseaux de neurones sont bons pour la prédiction et l'estimation seulement quand : les entrées et les sorties sont bien comprises, l'expérience est disponible pour un nombre "suffisant" d'exemples à utiliser pour entraîner le réseau. Ils ne peuvent faire mieux que les exemples qui lui sont fournis. Etant statiques, ils doivent être régulièrement mis à jour avec les nouveaux exemples.

Ils sont remarquables quant à leur aptitude à modéliser des structures complexes et des données irrégulières. En particulier, pour les relations, interactions, non linéaires entre les variables. Ils offrent une robustesse satisfaisante aux données bruitées Ils s'appliquent à une grande variété de problèmes.

Ils requièrent un pré-traitement sur les données aberrantes auxquelles ils sont sensibles. Pour les PMC, les données doivent être normalisées (i.e. appartenir à $[\![0,1]\!]$).

Leurs faiblesses sont le caractère non explicite des résultats, lors de la convergence qui ne se réalise pas toujours vers la meilleure solution globale, l'apprentissage qui requiert au préalable un choix peu guidé de l'architecture (nombre de couches, nombre de neurones par couche) et du taux d'apprentissage. Ils sont également sensibles à un trop grand nombre de variables discriminantes.

Liens utiles

Le lecteur est invité à écouter le cours de Yann LeCun (au Collège de France) et pourra ainsi s'intéresser aux réseaux dit *convolutifs* dont il est l'inventeur :
http ://deeplearning.net/tutorial/lenet.html.

Il est également invité à lire une traduction de l'article original et limpide de Brandon Rohrer (Senior Data Scientist à Facebook), "How Convolutional Neural Networks Work" : "Comment les réseaux de neurones à convolution fonctionnent" :

```
https ://medium.com/@CharlesCrouspeyre/
 comment-les-rÃ©seaux-de-neurones-Ã -convolution-fonctionnent-b288519dbcf8
```

Pour votre projet

Voici l'adresse du serveur sage de l'UPMC (sous Jupiter) :

https ://sage.math.upmc.fr/hub/login

Pour se connecter, il faut utiliser les identifiants UPMC avec peut-être "pN" comme login où N est votre numéro d'étudiant (c'est le CAS qui gère cela).

Il vient en complément du Cloud de SageMath : https ://cocalc.com.

Les logiciels

Traité en cours

1. **Sous R : la fonction** `nnet` **pour le PMC, les librairies** `neuralnet`**,** `RSNNS` **qui en particulier implante le réseau RBF**

2. **Spad pour l'apprentissage non supervisé**

3. **La procédure** `Proc Neural` **pour le perceptron sous** `SAS`

4. **Le logiciel** `Tanagra`

5. `SNNS`

6. `Weka`

7. `Scilab, SageMath, Maxima`

Bibliographie

[1] J.A. Anderson et E. Rosenfeld Eds. , **Neurocomputing, Fondations of research**, MIT Press Cambridge, Massachusetts, Vol. I et II, 1989.
(contient d nombreux articles de référence).

[2] A.R. Baron Universal approximation bounds for superposition of a sigmoïde function IEEE Trans. on Inf. Theory, vol. 39, N. 3, 930-945 (Rap. interne 56, Stat. Dep. Univ. of Illinois, 1991)

[3] A.R. Baron Approximation and Estimation Bounds for Artificial Neural Networks, Machine Learning, vol. 14, 115-133.

[4] Cybenko G. (1989) Approximation by Superposition of a sigmoïde function, Mathematics and control, Signals, and Systems, **2**, 303-314.

[5] Fahlman S.E. (1988) Faster learning variations on back-propagation : an empirical study, Processing of Connexionist Models. Summer School, Morgan Kaufmann, CA, 38-51

[6] Fahlman S.E and C. Lebière, The cascade correlation learning architecture, Neural Information Processing Systems, 2, pp. 524-532, 1990.

[7] Funahashi K, On the approximate realisation of continuous mapping by neural networks, Neural Networks, 2, pp. 183-192, 1989.

[8] Hassibi B. and D.G. Stork, Second order derivatives for network pruning : optimal brain surgeon, Advances in Neural Information Processing Systems, S.H. Hanson, J.D. Cowan and C.L. Gilles (Eds.), Morgan Kaufmann, San Mateo, CA, 5, pp. 164-171, 1993.

[9] Jodouin (livre dont une grande partie de ce cours est tiré)

[10] Jollife I.T., Principal Component Analysis, Springer, 1986.

[11] J.J. Hopfield (1982) Neural networks and physical system with emergent collective computational Abilities, Proc. Natl. Acac. Sci. USA 79 2554-2558.

[12] D.O. Hebb, **The organization of behavior**, Wiley, 1949.

[13] T. Kohonen (1974), An adaptative associative memory principle, IEEE Transaction on Computers, 444-445.

[14] Kwok T.Y. and D.Y. Yeung, Constructive Algorithms for structure learning in feedforward neural networks for regression problems, IEEE Trans. on Neural Network, 8(3), pp. 630-645, 1997.

[15] Ljung L., System identification : theory for the users, Prentice-Hall, Englewood Cliffs, N.J., 1987.

[16] , W.S. McColloch et W. Pitts (1943), A logical Calculus of the ideas immanent in nervous activity, Bull. Math. Biophysics 5, 115-133.

[17] F. Rosenblatt (1957), The perceptron, A perceiving and recognizing automation, Cornell Aeronautical Laboratory Report 85-460-1.

[18] Ruck D.W., S.K. Rogers and M. Kabrisky, Feature selection using a multilayer perceptron, Neural Network Computing, 2(2), pp. 40-48, 1990.

[19] M.C. Mozer et D.E. Rumelhard, Lawrence Erlbaum, **Mathematical perspectives on neural network**, ed. P. Smolensky Associates, (1996).

[20] Tarr G., Multilayered feedforward networks for image segmentation, PhD Air Force Inst. Technol. Whright-Patterson AFB, 1991.