Chapitre 2

Le neurone de Mc Culloch et Pitts

Le neurone le plus simple est le neurone de McCulloch et Pitts, c'est le premier neurone qui est apparue (en 1943). Il est décrit comme suit :

N variables $(x_1, x_2, \dots, x_N)$ constituent les entrées.

Les valeurs de ces variables sont respectivement multipliées par des constantes $w_1, w_2, \dots, w_N$, qui sont appelées les *poids*.

La somme pondérée $a = w_1x_1, w_2x_2, \dots, w_Nx_N$, qui est appelée *activation*, est comparée à une valeur donnée θ, qui est appelée *seuil*.

Si la valeur de l'activation est inférieure à celle du seuil, le neurone reste inerte et produit la valeur 0.

Si la valeur de l'activation est supérieure à celle du seuil, le neurone devient actif et produit la valeur 1.

Ainsi, le neurone de McCulloch et Pitts ne peut fournir que deux réponses possibles, 0 ou 1.

En raison de cela nous pouvons dire que la fonction de transfert de ce neurone, qui transforme la valeur de l'activation en valeur de sortie, est la fonction à seuil.

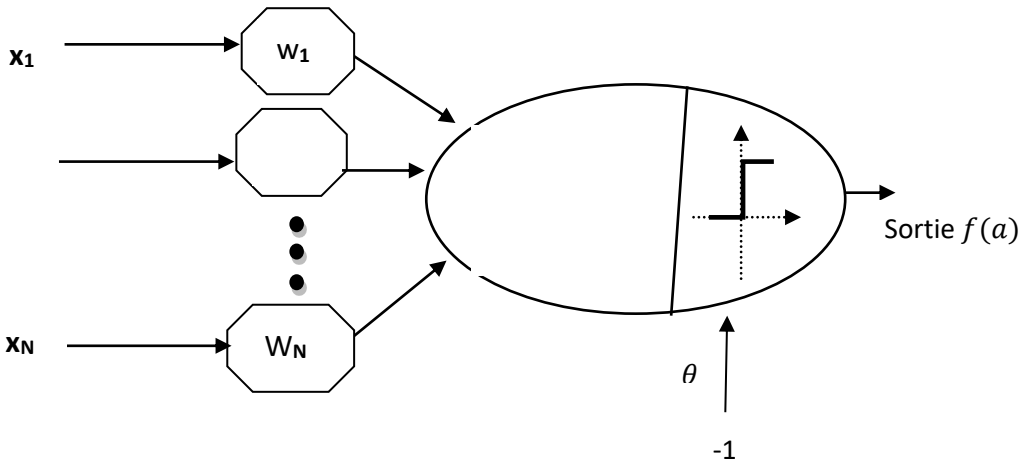


Figure 2.1 : Le neurone de McCulloch et Pitts

La valeur de l'activation est $a = \sum_i^N w_i x_i$

Si $a \geq \theta$ alors $sortie\ du\ neurone = f(a) = 1$

Si $a < \theta$ alors $sortie\ du\ neurone = f(a) = 0$

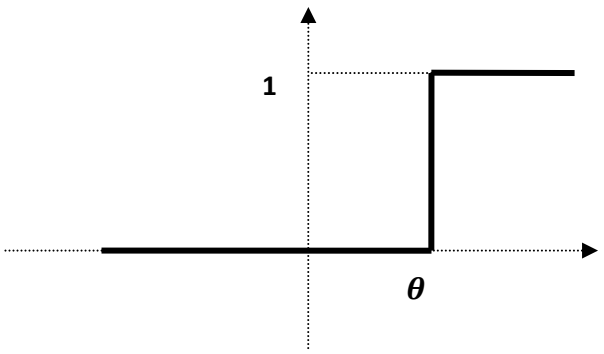


Figure 2.2 : La fonction à seuil

1- Universalité des neurones de McCulloch et Pitts

Le neurone le plus simple est celui de McCulloch et Pitts.

Nous allons montrer que des réseaux de tels neurones ont des possibilités de **calcul universel**.

Définition :

On dit qu'un réseau a des possibilités de calcul universel lorsqu'il peut émuler toute machine de traitement de l'information *numérique déterministe finie*.

On doit démontrer qu'un réseau de neurones de McCulloch-Pitts correctement construit a des possibilités de calcul universel.

Est-ce que les unités logiques de base des machines digitales peuvent toutes être construites avec des neurones de McCulloch-Pitts?

Si oui et par conséquent :

Tous les théorèmes de l'informatique qui s'appliquent à de telles machines digitales s'appliqueront tout aussi bien aux réseaux de neurones de McCulloch-Pitts.

2. Réduction aux opérations binaires à sortie unique

Nous allons en premier démontrer comment les machines digitales peuvent être réduites à une collection de machines plus simples binaires à sortie unique :

- Des machines digitales finies ne peuvent manipuler par définition que seulement des représentations de nombres réels en **précision finie** c'est-à-dire,

$$\pi \rightarrow 3.14159265358979323846264 3$$

- Toute représentation en précision finie d'un nombre réel peut, à son tour, être exprimée en termes de nombres entiers :

$$\pi \rightarrow \left(\underbrace{31415926535897932384626433}_{chiffres} ; \underbrace{}_{point\ décimal} \right)$$

- Tout nombre entier peut, à son tour, être exprimé comme une chaine (dite **booléenne**) de nombres binaires appartenant à l'ensemble $\{0\ ,\ 1\ \}$.

Par exemple,

$$\begin{aligned} 1239 &= 1.2^{10} + 0.2^9 + 0.2^8 + 1.2^7 + 1.2^6 + 0.2^5 + 1.2^4 + 0.2^3 \\ &\quad + 1.2^2 + 1.2^1 + 1.2^0 \\ &= 1024 + 128 + 64 + 16 + 4 + 2 + 1 = (10011010111) \end{aligned}$$

Exemple : $57_{(10)} = ?_{(2)}$

Exemple : $637_{(10)} = ?_{(2)}$

D'où : $57_{(10)}=111001_{(2)}$ et $637_{(10)}=1001111101_{(2)}$

Exemple : $0{,}6875_{(10)} = ?_{(2)}$

$0{,}6875 \times 2 = 1{,}375$ partie entière = **1**

$0{,}375 \times 2 = 0{,}75$ partie entière = **0**

$0{,}75 \times 2 = 1{,}5$ partie entière = **1**

$0{,}5 \times 2 = 1{,}0$ partie entière = **1**

d'où $0{,}6875_{(10)} = \mathbf{0{,}1011}_{(2)}$

Exemple : $0{,}8125_{(10)} = ?_{(2)}$

$0{,}8125 \times 2 = 1{,}625$ partie entière = **1**

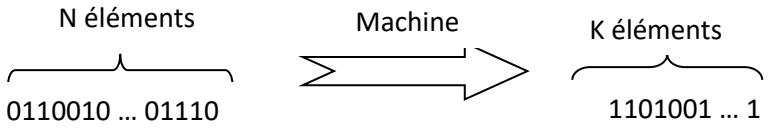
$0{,}625 \times 2 = 1{,}25$ partie entière = **1**

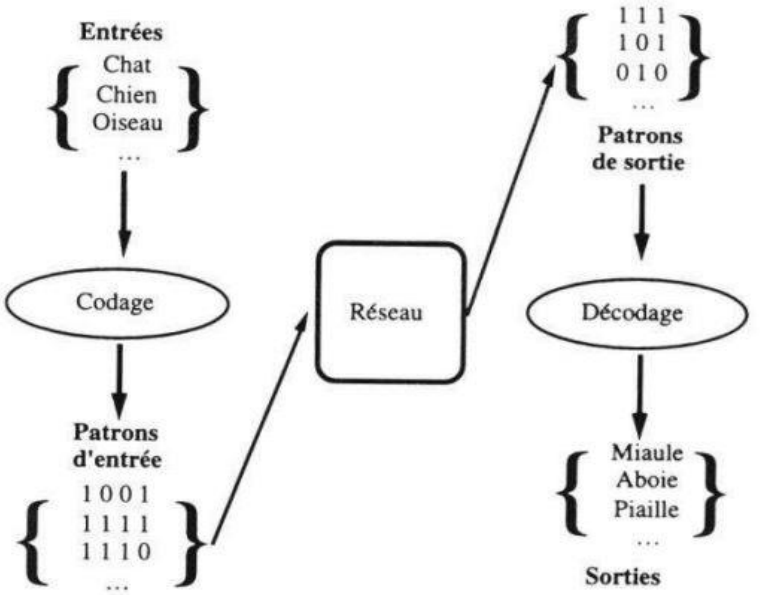
$0{,}25 \times 2 = 0{,}5$ partie entière = **0**

$0{,}5 \times 2 = 1{,}0$ partie entière = **1**

d'où $0{,}8125_{(10)} = \mathbf{0{,}1101}_{(2)}$

- Il s'ensuit que toute machine digitale finie peut être vue comme un dispositif qui projette (applique) des vecteurs binaires d'entrée de dimension finie $S \in \Omega \subseteq \{0\ ,1\}^N$ (représentant des caractères, des nombres, des lettres tapées sur le clavier, des images de caméras, etc.) sur des vecteurs binaires de sortie de dimension finie $S' \in \{0\ ,1\}^K$ (caractères, nombres, texte de réponse sur un écran, signal de contrôle pour un autre équipement, etc.).





Les unités de la couche d'entrée sont spécialisées pour recevoir l'input; les états de celles de la couche de sortie vont être interprétés comme la réponse du réseau.

- Par définition, des machines déterministes répondront toujours de la même manière pour des données d'entrée identiques.
- Ainsi toute machine digitale déterministe finie peut être entièrement spécifiée par la spécification du vecteur sortie $S'(S)$ pour tout vecteur d'entrée possible $S \in \Omega$ c'est-à-dire, par la spécification de l'application M :

$$M: \quad \Omega \to \{0\ ,1\}^K \quad MS = S'(S) \ \ \forall S \in \Omega$$

- Finalement, nous pouvons toujours construire K sous machines indépendantes M_l $(l = 1, \dots, K)$ chacune s'occupe de l'une des K composantes de sortie :

$$M_l: \quad \Omega \rightarrow \{0\,,1\} \quad M_l S = S_l'(S) \quad \forall S \in \Omega$$

Nous concluons qu'il est possible que nous pouvons nous concentrer seulement sur la question de voir s'il est possible d'effectuer toute application à sortie unique :

$$M: \quad \Omega \subseteq \{0\,,1\}^N \quad \rightarrow \quad \{0\,,1\}$$

avec un réseau de neurones de McCulloch-Pitts.

Nous répondrons à cette question par l'affirmative en deux étapes :

- ✓ Nous allons d'abord montrer que *toute fonction booléenne* à sortie unique de N variables peut être exprimée en termes de combinaisons de trois opérations logiques effectuées sur les entrées (ces trois opérations logiques peuvent même être réduites à une seule).
- ✓ Nous montrons ensuite que toutes ces trois opérations élémentaires peuvent être produites par *les neurones de McCulloch-Pitts.*

3. Réduction à trois opérations élémentaires

Dans le but d'arriver à cette réduction, nous allons introduire un partitionnement de l'ensemble Ω des vecteurs d'entrée en deux sous-ensembles, dépendant de la valeur de sortie voulue :

$$\Omega = \Omega^{+} \cup \Omega^{-}$$

avec

$$\Omega^{+} = \{S \in \Omega \ / \ MS = 1\}$$
$$\Omega^{-} = \{S \in \Omega \ / \ MS = 0\}$$

Nous numérotons maintenant les éléments de Ω^{+}.

Le nombre d'éléments de Ω^{+} , noté P, ne peut évidemment pas être plus grand que 2^{N} , qui est le nombre total des vecteurs de $\{0\,,1\}^{N}$

$$\Omega^{+} = \{S^{1}, S^{2}, \dots, S^{P-1}, S^{P}\} \qquad S^{\mu} \in \{0\,,1\}^{N}$$

L'évaluation de M revient à identifier si oui ou non un vecteur d'entrée appartient à Ω^{+}.

Cette identification peut être faite en utilisant seulement les trois opérations logiques : $\wedge\ (AND)$,$\vee\ (OR)$ et $\dashv\ (NOT)$

qui sont définies par leurs tables :

$\wedge: \{\mathbf{0}\,,\mathbf{1}\}^{\mathbf{2}} \rightarrow \{\mathbf{0}\,,\mathbf{1}\}$

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

$\vee: \{\mathbf{0}\,,\mathbf{1}\}^{\mathbf{2}} \rightarrow \{\mathbf{0}\,,\mathbf{1}\}$

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

$\dashv: \{\mathbf{0}\,,\mathbf{1}\} \rightarrow \{\mathbf{0}\,,\mathbf{1}\}$

x	$\dashv x$
0	1
1	0

Nous pouvons utiliser ces opérations de base pour construire l'opérateur SAME(x,y), qui nous dit quand deux variables binaires ont la même valeur :

$$SAME(x, y) = (x \wedge y) \vee ((\dashv x) \wedge (\dashv y))$$

Les définitions de AND et OR peuvent être étendues pour couvrir des variables ayant plus de deux arguments :

$$x_1 \wedge x_2 \wedge \ldots \wedge x_{L-1} \wedge x_L = x_1 \wedge (x_2 \wedge (\ldots \wedge (x_{L-1} \wedge x_L) \ldots))$$

$$x_1 \vee x_2 \vee \ldots \vee x_{L-1} \vee x_L = x_1 \vee (x_2 \vee (\ldots \vee (x_{L-1} \vee x_L) \ldots))$$

La vérification du fait que le vecteur d'entrée S est identique à l'un quelconque des vecteurs S^{μ} de l'ensemble Ω^{+} est alors faite par l'évaluation de

$$\begin{aligned} MS = (&SAME(S_1, S_1^1) \wedge (SAME(S_2, S_2^1) \wedge \ldots \wedge (SAME(S_N, S_N^1) \\ &\vee (SAME(S_1, S_1^2) \wedge (SAME(S_2, S_2^2) \wedge \ldots \wedge (SAME(S_N, S_N^2) \\ &\qquad\qquad \vdots \\ &\vee (SAME(S_1, S_1^P) \wedge (SAME(S_2, S_2^P) \wedge \ldots \wedge (SAME(S_N, S_N^P) \end{aligned}$$

Dans la mesure où cette dernière expression peut être entièrement construite avec les trois opérations de base $\{\wedge, \vee, \dashv\}$.

M: $\quad \Omega \subseteq \{0, 1\}^N \quad \rightarrow \quad \{0, 1\}$

L'opération de toute machine digitale déterministe finie, peut être réduite à une combinaison de ces trois opérations de base.

Remarque :

Nous pouvons encore réduire l'ensemble des opérations requises, du fait que $\vee$ (OR) peut être écrit en termes de $\dashv$ (NOT) et $\wedge$ (AND) de la manière suivante :

$$x \vee y = \dashv ((\dashv x) \wedge (\dashv y))$$

x	y	$x \vee y$	$\dashv x$	$\dashv y$	$(\dashv x) \wedge (\dashv y)$	$\dashv ((\dashv x) \wedge (\dashv y))$
0	0	0	1	1	1	0
0	1	1	1	0	0	1
1	0	1	0	1	0	1
1	1	1	0	0	0	1

En fait nous pouvons encore réduire l'ensemble des opérations, du fait que les opérations $\{\wedge, \dashv\}$peuvent, à leur tour, être écrites avec la seule opération NAND (NOT – AND). Autrement dit, lorsque nous avons deux propositions x et y, l'opération $NAND(x, y)$ résulte de la composition successive des deux opérations AND puis NOT. C'est-à-dire : $NAND(x, y) = \dashv (x \wedge y)$

Comme $\dashv x = NAND(x, x)$ et $x \wedge y = NAND(NAND(x, y), NAND(x, y))$

Alors :

x	y	$NAND(x,y)$	$NAND(NAND(x,y),NAND(x,y))$
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

x	$NAND(x,x)$
0	1
1	0

4. Opérations élémentaires avec les neurones de McCulloch –Pitts

Il nous reste à montrer que chacune des trois opérations logiques élémentaires $\{\wedge, \vee, \neg\}$ peuvent être réalisées avec nos simples neurones de McCulloch-Pitts

$$S(x) = \theta\left(\sum_{k=1}^{N} w_k x_k - U\right)$$

Remarque

> θ n'est pas une valeur mais c'est une fonction (indicatrice de la positivité), qui prend la valeur 1 lorsque l'entrée est positive ou nulle ; et qui prend la valeur 0 lorsque l'entrée est négative.
>
> Ainsi, nous avons :

$$\begin{cases} \theta\left(\sum_{k=1}^{N} w_k x_k - U\right) = 1 & si \left(\sum_{k=1}^{N} w_k x_k - U\right) \geq 0 \\ \theta\left(\sum_{k=1}^{N} w_k x_k - U\right) = 0 & si \left(\sum_{k=1}^{N} w_k x_k - U\right) < 0 \end{cases}$$

Il suffit de choisir les valeurs appropriées pour les paramètres $\{w_k, U\}$. Ceci sera fait par construction.

Remarquons que, dans le but de prouver l'universalité, nous avons seulement à construire les opérations NAND et NOR avec les neurones de McCulloch-Pitts ; cependant à titre d'illustration nous construisons aussi les opérations $\{\wedge, \vee, \dashv\}$:

x	y	$x \wedge y$	$1.x + 1.y - (3/2)$	$\theta(x + y - 3/2)$
0	0	0	-3/2	0
0	1	0	-1/2	0
1	0	0	-1/2	0
1	1	1	½	1

x	y	$x \vee y$	$1.x + 1.y - (1/2)$	$\theta(x + y - 1/2)$
0	0	0	-1/2	0
0	1	1	½	1
1	0	1	½	1
1	1	1	3/2	1

x	$\dashv x$	$(-1).x-(-1/2)$	$\theta(-1.x+1/2)$
0	1	½	1
1	0	-1/2	0

x	y	$NAND(x,y)$	$-x-y+3/2$	$\theta(-x-y+3/2)$
0	0	1	3/2	1
0	1	1	½	1
1	0	1	½	1
1	1	0	-1/2	0

x	y	$NOR(x,y)$	$-x-y+1/2$	$\theta(-x-y+1/2)$
0	0	1	½	1
0	1	0	-½	0
1	0	0	-½	0
1	1	0	-3/2	0

5. Séparabilité linéaire

Toutes les **opérations logiques élémentaires** que nous avons rencontrées dans le chapitre précédent peuvent non seulement être réalisées avec des neurones de McCulloch-Pitts, **mais même aussi avec un seul neurone** de McCulloch-Pitts.

La question qui se pose naturellement est :

Si toutes les opérations $\{0,1\}^N \rightarrow \{0,1\}$ peuvent être exécutées avec un seul neurone de McCulloch-Pitts (?).

Pour $N = 1$, c'est-à-dire, le cas trivial où il n'y a qu'une seule variable d'entrée $x \in \{0,1\}$, nous pouvons simplement vérifier toutes les opérations possibles $M : \{0,1\} \rightarrow \{0,1\}$

(qui sont au nombre de quatre, que nous allons noter Ma, Mb, Mc, et Md), et nous pouvons vérifier que nous pouvons construire un neurone de McCulloch-Pitts équivalent :

$$S(x) = \theta(w.x - U)$$

x	$M_a(x)$	$\theta(-1)$	$M_b(x)$	$\theta(-x + 1/2)$	$M_c(x)$	$\theta(x - 1/2)$	$M_d(x)$	$\theta(1)$
0	0	0	1	1	0	0	1	1
1	0	0	0	0	1	1	1	1

Cependant, pour $N > 1$ la réponse est non.

Contre-exemple pour $N = 2$:

Ce contre-exemple peut par la suite être utilisé pour générer des contre exemples pour tout $N \geq 2$, c'est l'opération XOR (Le OR exclusif) :

x	y	$XOR(x,y)$
0	0	0
0	1	1
1	0	1
1	1	0

Proposition

Il n'existe pas de nombres réels $\{w_x, w_y, U\}$ tels que

$$\theta(w_x.x + w_y.y - U) = XOR(x,y)$$

$$\forall (x,y) \in \{0,1\}^2$$

Démonstration

Par l'absurde. Supposons que la proposition est fausse, ce qui implique :

$(x,y) = (0,0)$: $-U < 0 \quad \Rightarrow \quad U > 0$

$(x,y) = (0,1)$: $w_y - U > 0 \quad \Rightarrow \quad w_y > U$

$(x,y) = (1,0)$: $w_x - U > 0 \quad \Rightarrow \quad w_x > U$

$(x,y) = (1,1)$: $w_x + w_y - U < 0 \quad \Rightarrow \quad w_x + w_y < U$

Cette contradiction montre que la proposition précédente ne peut jamais être fausse, ce qui achève la démonstration.

Pour N > 2 nous pouvons construire une opération similaire M juste en appliquant l'opération XOR aux deux premières des N variables :

$$M: \{0\,,1\}^N \rightarrow \{0\,,1\} \qquad M(x_1, \ldots, x_N) = XOR(x_1, x_2)$$

A laquelle la même démonstration s'applique comme pour le cas $N = 2$.

Comme résultat nous verrons qu'en effet pour tout $N \geq 2$ il existe des opérations qui ne peuvent pas être exécutées par un **seul** neurone de McCulloch-Pitts.

Nous pouvons également donner une image géométrique de la situation.

L'ensemble de toutes les opérations possibles $M: \{0,1\}^N \rightarrow \{0,1\}$ revient à l'ensemble des moyens possibles de remplir la colonne de droite par des zéros et des uns :

x_1	x_2	$\cdots$	$\cdots$	x_{N-1}	x_N	$M(x)$
0	0	$\cdots$	$\cdots$	0	0	$*$
1	0	$\cdots$	$\cdots$	0	0	$*$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	1	$\cdots$	$\cdots$	1	0	$*$
1	1	$\cdots$	$\cdots$	1	1	$*$

Il y a 2^N entrées dans cette colonne, avec deux valeurs possibles pour chacune d'elle.

Ainsi le nombre d'opérations $M: \{0,1\}^N \to \{0,1\}$ est 2^{2^N} .

L'ensemble $\{0,1\}^N$ des vecteurs binaires d'entrée est l'ensemble des coins de l'hyper-cube unité dans $\mathbb{R}^N$.

Un neurone de McCulloch-Pitts , en exécutant l'opération

$$S: \{0,1\}^N \to \{0,1\}$$

$$\mathrm{S(x)} = \theta\left(\sum_{\mathrm{k=1}}^{\mathrm{N}} \mathrm{w_k x_k} - \mathrm{U}\right)$$

divise l'espace $\mathbb{R}^N$ en deux sous ensembles qui sont séparés par un hyper-plan

$$\sum_{\mathrm{k=1}}^{\mathrm{N}} \mathrm{w_k x_k} = \mathrm{U}$$

L'information fournie par le résultat de l'opération du neurone de McCulloch-Pitts, sur une donnée d'entrée $x \in \{0,1\}^N$, est simplement dans quel sous ensemble le coin x est localisé.

$S(x) = 1$ si x est dans le sous ensemble : $\sum_{\mathrm{k=1}}^{\mathrm{N}} \mathrm{w_k x_k} > U$

$S(x) = 0$ si x est dans le sous ensemble : $\sum_{\mathrm{k=1}}^{\mathrm{N}} \mathrm{w_k x_k} < U$

Remarquons que cette division de $\{0,1\}^N$ en sous–ensembles, qui sont ici séparés par un plan, est identique à la division que nous avons déjà faite en répartissant en Ω^+et Ω^-.

Ω^+ est l'ensemble de tous les coins $x \in \{0,1\}^N$ de l'hyper-cube pour lesquels $* = 1$.

Ω^- est l'ensemble de tous les coins $x \in \{0,1\}^N$ de l'hyper-cube pour lesquels $* = 0$.

Dans la mesure où chacune de ces $2^{2^{\mathrm{N}}}$ opérations M est caractérisée uniquement par son ensemble associé Ω^+ , chacune d'elle peut être représentée par le dessin de l'hyper-cube dans $\mathbb{R}^N$ dans lequel les coins

appartenant à Ω^+ sont colorés en noir, et les coins appartenant à Ω^-sont colorés en blanc.

Pour $N = 1$ l'hyper-cube se réduit simplement à un segment avec $2^1 = 2$ 'coins'.

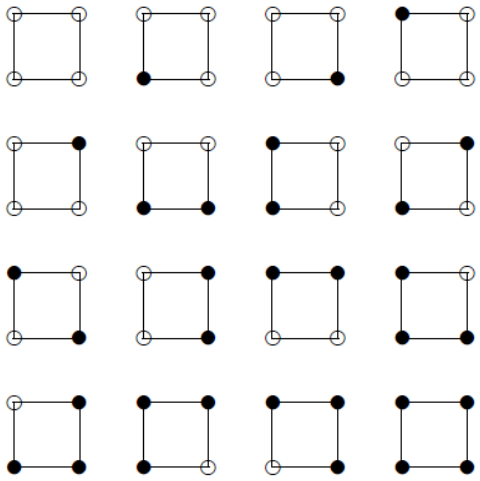
Il y a $2^{2^1} = 4$ opérations : $\{0,1\} \to \{0,1\}$, c'est-à-dire quatre façons de colorer les deux coins :

$N = 1$: $x \in \{0,1\}$ •: $x \in \Omega^+$, $M(x) = 1$ ○: $x \in \Omega^-$, $M(x) = 0$

Pour N = 2 l'hyper-cube est un carré (avec $2^2 = 4$ coins).

Il y a $2^{2^2} = 16$ opérations $M: \{0,1\}^2 \to \{0,1\}$, c'est-à-dire, seize façons de colorer les quatre coins :

$N = 2$: $x \in \{0,1\}^2$ •: $x \in \Omega^+$, $M(x) = 1$ ○: $x \in \Omega^-$, $M(x) = 0$



Pour N = 3 l'hyper-cube est un cube (avec huit coins).

Il y a $2^{2^3} = 256$ opérations $M: \{0,1\}^3 \to \{0,1\}$, c'est-à-dire, 256 façons de colorer les huit coins, etc.

De toutes ces opérations, les neurones de McCulloch-Pitts ne peuvent exécuter seulement que celles pour lesquelles les coins noirs de Ω^+ peuvent être séparés des coins blancs de Ω^-par un seul plan

$$\boldsymbol{x}.\boldsymbol{W} = U.$$

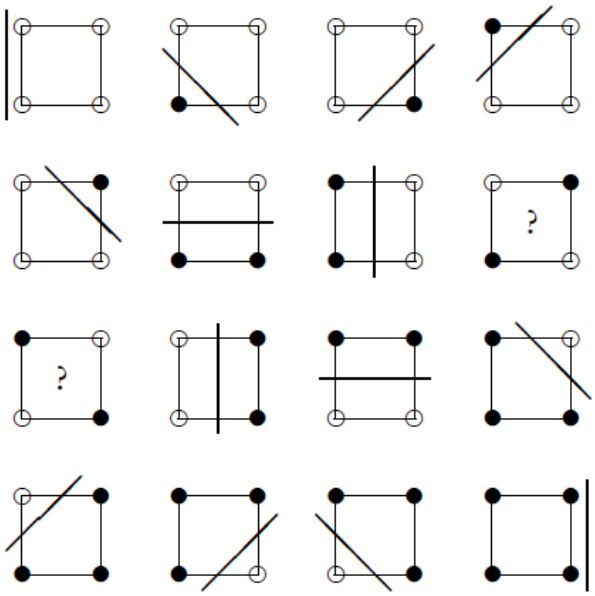
De telles opérations sont dites ☺ *linéairement séparables* ☺

Il est clair, de voir pour les exemples

N = 1 et N = 2 lesquelles des opérations sont linéairement séparables.

N =1

N = 2



Pour N = 1 nous retrouvons notre résultat précédent qui dit que toutes les opérations sont linéairement séparables (i.e. elles peuvent être exécutées par un neurone de McCulloch-Pitts convenablement construit).

Pour N= 2 nous trouvons que parmi les seize opérations possibles, les deux suivantes ne sont pas linéairement séparables :

x	y	$M_a(x,y)$		x	y	$M_b(x,y)$
0	0	0		0	0	1
0	1	1		0	1	0
1	0	1		1	0	0
1	1	0		1	1	1

Et nous reconnaissons en elles les deux opérations XOR et $\dashv XOR$.

6. Réseaux multicouche

Nous avons vu qu'un seul neurone de McCulloch-Pitts ne peut réaliser toutes les applications

$$M: \{0,1\}^N \rightarrow \{0,1\}.$$

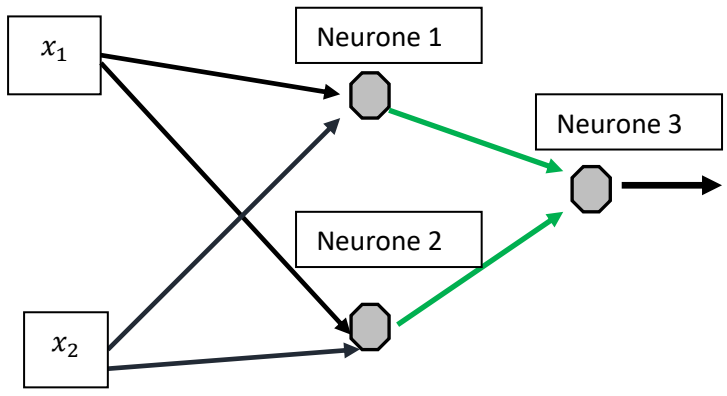
Nous allons maintenant démontrer que toute application M peut être au moins réalisée par un réseau de ces mêmes neurones à propagation avant et à deux couches, avec seulement un seul neurone dans la deuxième couche.

Première méthode :

Soit le réseau de neurones suivant où :

Le neurone 1 effectue l'opération OU
Le neurone 2 effectue l'opération ET

entrée x1	entrée x2	sortie
0	0	0
0	1	1
1	0	1
1	1	0



- Il s'agit de déterminer les poids et les seuils des trois neurones pour que ce réseau effectue l'opération XOR.
- Les trois neurones étant des neurones de McCulloch et Pitts.

Opération OU

x	y	$x \vee y$	$x + y - 1/2$	$\theta(x + y - 1/2)$
0	0	0	-1/2	0
0	1	1	½	1
1	0	1	½	1
1	1	1	3/2	1

$$w_x = 1;\ w_y = 1; U = \frac{1}{2}$$

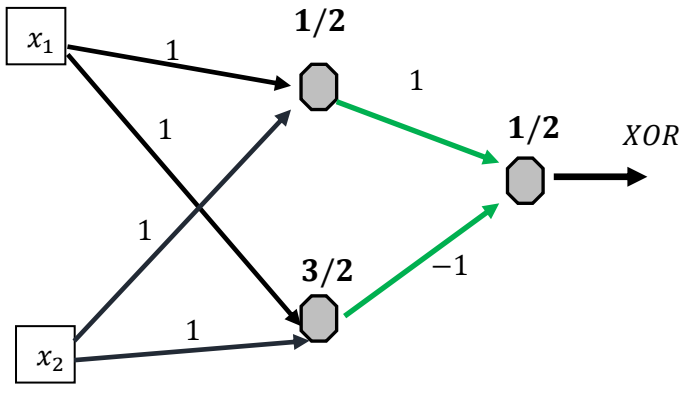
Opération ET

x	y	$x \wedge y$	$x + y - 3/2$	$\theta(x + y - 3/2)$
0	0	0	-3/2	0
0	1	0	-1/2	0
1	0	0	-1/2	0
1	1	1	1/2	1

$$w_x = 1;\ w_y = 1; U = \frac{3}{2}$$

Pour le troisième neurone :

$$w_{ou} = 1;\ w_{et} = -1; U = \frac{1}{2}$$



Deuxième méthode (qui conduit à une méthode générale)

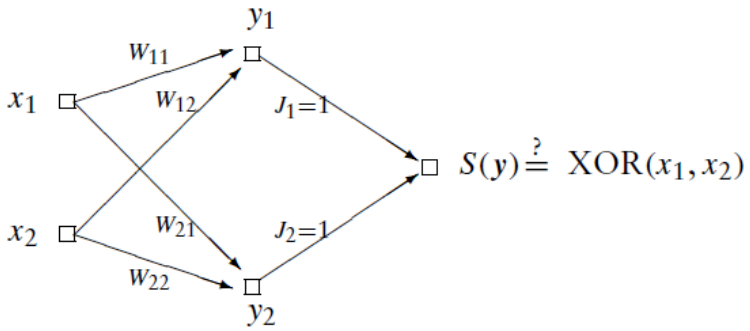
Ce sera en fait une réalisation par le moyen neuronal de la recherche dans la table des éléments de l'ensemble Ω^+ , comme cela a été vu lors de la démonstration de l'universalité des neurones de McCulloch-Pitts.

En tant qu'exemple simple et explicite, nous allons illustrer comment un perceptron multicouche peut être construit pour exécuter une tâche non linéairement séparable telle que l'opération XOR.

Ici l'ensemble des vecteurs Ω^+, pour lesquels l'opération XOR doit donner une valeur de sortie égale à 1, est :

$$\Omega^+ = \{x^1, x^2\} \quad \text{avec } x^1 = (1,0), \quad x^2 = (0,1)$$

Notre solution est basée sur une construction qui utilise notre connaissance de Ω^+, et qui est construite de sorte que pour vecteur x^{i} de Ω^+, il y a précisément un neurone ***i*** dans la couche cachée qui produit +1 pour ce vecteur (et seulement pour ce vecteur).



Les poids w_{ij} et les seuils V_i des neurones cachés qui aboutissent à cela se déduisent du fait que pour $\mathrm{x}^{\mathrm{i}} \in \Omega^+$ et $x \in \{0,1\}^2$ la somme

$$\sum_{j=1}^{2} (2x_j^i - 1)(2x_j - 1) = \sum_{j=1}^{2} 2(2x_j^i - 1)\, x_j - \sum_{j=1}^{2} (2x_j^i - 1)$$

sera égale à +2 si $x = x^i$, et sera inférieure ou égale à 0 sinon.

De ceci nous pouvons voir que les poids et les seuils convenables des neurones cachés pour obtenir l'action désirée

$$y_i(x) = \theta\left(\sum_{j=1}^{2} w_{ij}x_j - V_i\right) = \delta_{x,x^i}$$

Sont $w_{ij} = 2\left(2x_j^i - 1\right)$ et $V_i = 1 + \sum_{j=1}^{2}\left(2x_j^i - 1\right)$.

Pour les deux vecteurs de Ω^+, ceci implique :

$$W_{11} = 2(2x_1^1 - 1) = +2 \qquad W_{12} = 2(2x_2^1 - 1) = -2$$
$$W_{21} = 2(2x_1^2 - 1) = -2 \qquad W_{22} = 2(2x_2^2 - 1) = +2 \qquad V_1 = V_2 = 1$$

Ayant fixé les valeurs des paramètres sur ces valeurs précédentes, nous pouvons vérifier que l'opération XOR est maintenant correctement exécutée :

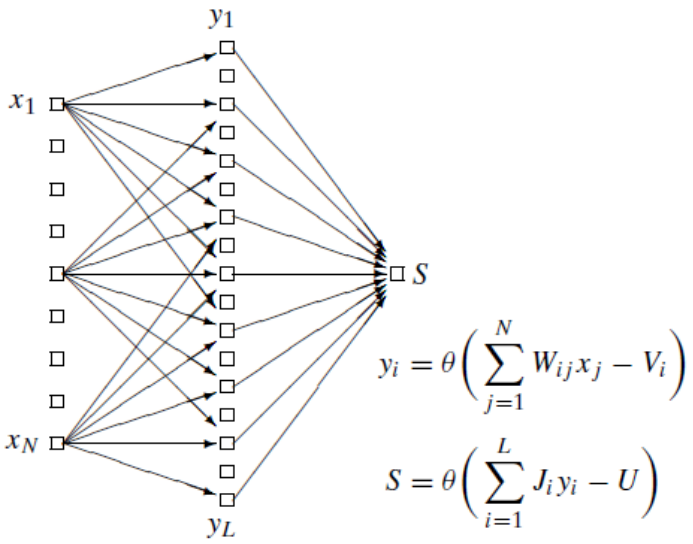
$$\begin{aligned}
\boldsymbol{x} = (0,0): &\quad S(\boldsymbol{y}) = \theta(y_1 + y_2 - 1/2) = \theta(0 + 0 - 1/2) = 0\\
\boldsymbol{x} = (1,0): &\quad S(\boldsymbol{y}) = \theta(y_1 + y_2 - 1/2) = \theta(1 + 0 - 1/2) = 1\\
\boldsymbol{x} = (0,1): &\quad S(\boldsymbol{y}) = \theta(y_1 + y_2 - 1/2) = \theta(0 + 1 - 1/2) = 1\\
\boldsymbol{x} = (1,1): &\quad S(\boldsymbol{y}) = \theta(y_1 + y_2 - 1/2) = \theta(0 + 0 - 1/2) = 0
\end{aligned}$$

Nous allons maintenant donner la construction générale, qui fonctionne pour n'importe quelle dimension N et pour n'importe quelle opération : $\{0,1\}^N \rightarrow \{0,1\}$.

Soit L le nombre des éléments de $\Omega^+ = \{x \in \{0,1\}^N / M(x) = 1\}$, construisons les neurones y_i de la couche cachée de sorte que chacun se spécialise sur l'un des vecteurs de Ω^+ et détermine si le vecteur d'entrée est égal ou pas à ce vecteur prototype.

Définissons le bloc constitutif G :

$$x, x^* \in \{0,1\}^N: \quad G(x^*; x) = \theta\left(\sum_{i=1}^{N}(2x_i - 1)(2x_i^* - 1) - N + 1\right)$$



Proposition

$G(x^*; x) = 1$ si et seulement si $x = x^*$.

Démonstration

Condition suffisante (si) :

$$G(x;x) = \theta\left(\sum_{i=1}^{N}(2x_i - 1)^2 - N + 1\right) = \theta\left(\sum_{i=1}^{N} 1 - N + 1\right) = 1$$

Condition nécessaire (seulement si) : Nous allons montrer que $G(x^*; x) = 0$ lorsque $x \neq x^*$.

$$\forall x_i, x_i^* \in \{0,1\}: \quad (2x_i - 1)(2x_i^* - 1) = \begin{cases} 1, & \text{if } x_i = x_i^* \\ -1, & \text{if } x_i \neq x_i^* \end{cases}$$

Par conséquent :

$\sum_{i=1}^{N}(2x_i - 1)(2x_i^* - 1) \; = \; N - 2 \times (nombre\ des\ indices\ i\ avec\ x_i \neq x_i^*)$

Et donc

$$x_i \neq x_i^* \Rightarrow G(x^*; x) = 0$$

CQAD.

L'expression $G(x^*; x)$, interprétée comme une opération sur la variable x (pour un x^*fixé), est clairement de la forme de McCulloch-Pitts.

$$G(x^*; x) = \theta\left(2\sum_{i=1}^{N}(2x_i^* - 1)x_i - 2\sum_{i=1}^{N}x_i^* + 1\right)$$

Nous pouvons maintenant construire notre réseau à deux couches avec ces blocs, en assignant à chaque vecteur $x^* \in \Omega^+$ un neurone caché de la forme $G(x^*; x)$:

$$\Omega^+ = \{x \in \{0,1\}^N \mid M(x) = 1\} = \{x^1, x^2, \ldots, x^{L-1}, x^L\}$$

$$\forall i \in \{1, \ldots, L\}: \qquad y_i : \{0,1\}^N \to \{0,1\} \quad y_i = G(x^i; x)$$

Le rôle du neurone de sortie S est de détecter simplement si l'un quelconque des neurones de la couche cachée est dans l'état 1.

Notre résultat final est le réseau suivant :

$$y_i(x) = \theta\left(\sum_{j=1}^{N} W_{ij}x_j - V_i\right) \qquad S(y) = \theta\left(\sum_{i=1}^{L} y_i - \frac{1}{2}\right)$$

$$W_{ij} = 2(2x_j^i - 1) \qquad V_i = 2\sum_{j=1}^{N} x_j^i - 1$$

Cette construction va exactement exécuter l'opération M.

L'état de chaque neurone cachée (N°i) indique si l'élément correspondant (N°i) de Ω^+ est égal au vecteur d'entrée x ou pas ($(y_i = 1)$ou $(y_i = 0)$).

Par la suite, le neurone de sortie S nous dit s'il y a ou pas un neurone dans la couche cachée qui est dans l'état +1 ($S = 1$ ou $S = 0$). C'est-à-dire si x est ou non dans Ω^+.

Ceci prouve donc qu'un réseau à deux couches à propagation avant peut reproduire n'importe quelle application $M: \{0,1\}^N \rightarrow \{0,1\}$.

Néanmoins ceci est une démonstration théorique pour prouver la possibilité, mais dans le fait le nombre de neurones croît exponentiellement avec N et ceci ne sera pas commode dans la pratique.

7. Le perceptron

Ce qui suit introduit la notion d'apprentissage dans son contexte général. Cette notion n'est pas définitivement définie. Ceci se fera dans le chapitre 3.

Cependant les démonstrations qui vont suivre peuvent être assimilées sans entrer dans les détails des paradigmes d'apprentissage, des règles et d'algorithmes d'apprentissage.

Le neurone de McCulloch et Pitts doté de la capacité d'apprentissage s'appelle Perceptron.

Nous nous tournons maintenant à la question de voir comment modéliser et comprendre le processus d'apprentissage dans le contexte simple du neurone de McCulloch-Pitts $\theta(J.x - U)$; ceci signifie l'adaptation des poids $\{J_i\}$ et du seuil U, dans le but d'améliorer l'exactitude dans l'exécution d'une tâche donnée

$$M: \Omega \subseteq \{0,1\}^N \to \{0,1\}.$$

Supposons que nous ne connaissons pas M explicitement ; nous avons seulement quelques exemples, fournis par un 'maître', qui sont constitués de 'questions' (vecteurs d'entrée $\mathrm{x} \subseteq \{0,1\}^N$) et des 'réponses' correspondantes (les valeurs associées $M(x)$).

Le point de départ d'une session d'entraînement est un neurone dont les paramètres $\{J_i\}$ et U sont, disons, choisis aléatoirement. Une session d'entraînement dite 'en ligne' consiste dans l'itération de la procédure suivante :

Etape 1 : prendre d'une manière aléatoire une question $x \in \Omega$

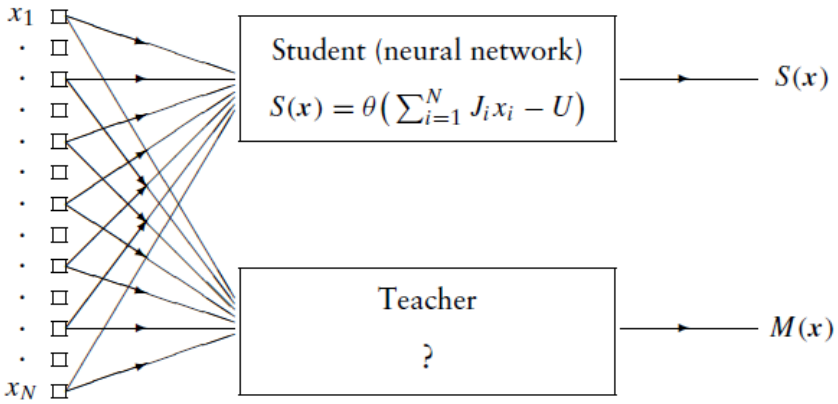
Etape 2 : vérifier si le maître et l'apprenant s'accordent sur la réponse :

$M(x) = S(x)$: Ne rien faire, retourner à 1

$M(x) \neq S(x)$: Modifier les paramètres, et retourner à 1.

Modification des paramètres :

$$J \to J + \Delta J(J, U; x, M(x))$$
$$U \to U + \Delta U(J, U; x, M(x))$$



Le but ultime est de finir avec un neurone qui a appris à exécuter la tâche M parfaitement, c'est-à-dire d'avoir $M(x) = S(x) = \theta(J.x - U),\ \forall x \in \Omega$ (qui, naturellement, n'est possible que si la tâche M est elle-même séparable). Le problème auquel nous faisons face est comment choisir la recette appropriée $(\Delta J, \Delta U)$ pour la mise à jour des paramètres du neurone qui aboutissent à cela.

✓ **Définition :** Un perceptron est un neurone de McCulloch-Pitts, dont l'apprentissage se fait en ligne, suivant la règle suivante pour la mise à jour des paramètres.

Règle d'apprentissage du perceptron :

$$\begin{cases} M(x) = 0, \;\; S(x) = 1: \;\; \Delta J = -x, \;\; \Delta U = 1 \\ M(x) = 1, \;\; S(x) = 0: \;\; \Delta J = x, \;\; \Delta U = -1 \end{cases}$$

Devoir Maison

Entrainer avec la règle d'apprentissage du perceptron l'ensemble suivant puis donner la conclusion générale :

x_t	$M(x)$
[0,0]	0
[0,1]	1
[1,0]	0
[1,1]	1

Les valeurs initiales : $J \leftarrow [0,0]$; $\mathrm{U} = 0.5$

A rendre le Mercredi 12 Décembre 2018

- Sous un certain angle, cette recette est plutôt transparente. Si $S(x) = 1$ mais qui devrait être égale à 0, l'effet de la modification est de décroître le champ local $h = J.x - U$ de sorte que la prochaine fois où la question x apparaît le perceptron aurait tendance à donner la solution (correcte) $S(x) = 0$.
- Réciproquement, si $S(x) = 0$ mais devrait être égale à 1, la modification cause une croissance du champ local de sorte que le perceptron aurait tendance à donner la solution (correcte) $S(x) = 1$ dans le futur.

Le théorème de la convergence du perceptron

L'élégante et puissante propriété de la recette spécifique est l'existence du théorème associé suivant de la convergence du perceptron.

Proposition

Si la tâche M est linéairement séparable, alors la procédure précédente converge en un nombre fini d'étapes de modifications vers une configuration stationnaire, où $\forall x \in \Omega: M(x) = S(x)$

Démonstration

Nous allons d'abord simplifier nos équations en introduisant une variable d'entrée artificielle, qui est simplement une constante : $x_0 = 1$. Ceci nous permettra, conjointement à l'identification $J_0 = -U$, d'écrire le perceptron et sa règle d'apprentissage dans sa forme compacte

$$S(x) = \theta(\boldsymbol{J} \cdot \boldsymbol{x}) \qquad \begin{aligned} \boldsymbol{x} &= (x_0, x_1, \ldots, x_N) \in \{0,1\}^{N+1}, \\ \boldsymbol{J} &= (J_0, J_1, \ldots, J_N) \in \mathbb{R}^{N+1} \end{aligned}$$

Règle d'apprentissage :

$$\begin{aligned} M(x) = 0,\ S(x) = 1: &\quad \Delta \boldsymbol{J} = -\boldsymbol{x} \\ M(x) = 1,\ S(x) = 0: &\quad \Delta \boldsymbol{J} = \boldsymbol{x} \end{aligned} \quad \Leftrightarrow \Delta \boldsymbol{J} = [2M(x) - 1]\boldsymbol{x}$$

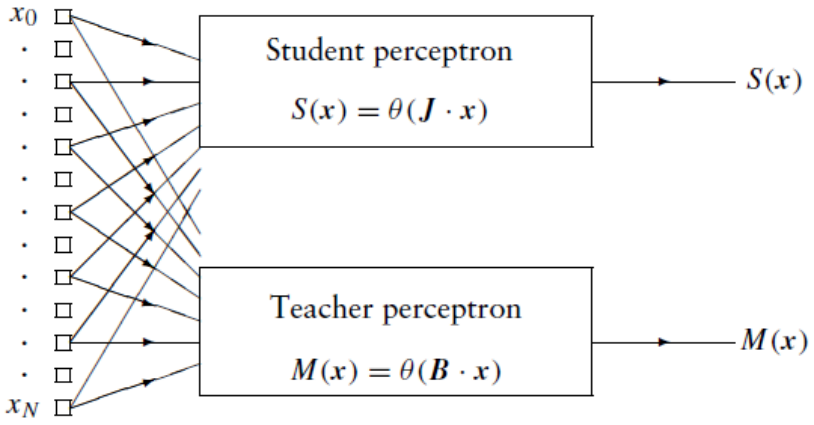
Le nouvel ensemble Ω est maintenant constitué de tous les vecteurs $(x_0, x_1, \ldots, x_N)$ avec $x_0 = 1$ et où $(x_0, x_1, \ldots, x_N)$ sont les composantes des vecteurs de l'ensemble d'origine.

✓ *La première étape de la démonstration* est d'écrire la séparabilité linéaire de l'opération M sous la forme suivante :

$\exists \boldsymbol{B} \in \mathbb{R}^{N+1}$ tel que $\forall \boldsymbol{x} \in \Omega: \quad M(\boldsymbol{x}) = \theta(\boldsymbol{B} \cdot \boldsymbol{x})$

Et du fait que nous savons que $M(x)$ est soit égal à 0 soit égal à 1 (d'une manière déterministe) et donc que $B.x$ ne peut jamais être égal à 0, alors :

$\exists \delta > 0$:tel que $\forall \boldsymbol{x} \in \Omega: \quad |\boldsymbol{B} \cdot \boldsymbol{x}| > \delta$



✓ B peut être fixée arbitrairement sans affecter l'opération associée M, nous pouvons choisir $|B| = 1.$

✓ Toute étape de modification, où $J \to J' = J + \Delta J$, doit vérifier les deux inégalités suivantes :

$$\begin{aligned} \boldsymbol{J}' \cdot \boldsymbol{B} &= \boldsymbol{J} \cdot \boldsymbol{B} + [2M(\boldsymbol{x}) - 1]\boldsymbol{x} \cdot \boldsymbol{B} \\ &= \boldsymbol{J} \cdot \boldsymbol{B} + [2\theta(\boldsymbol{x} \cdot \boldsymbol{B}) - 1]\boldsymbol{x} \cdot \boldsymbol{B} \\ &= \boldsymbol{J} \cdot \boldsymbol{B} + |\boldsymbol{x} \cdot \boldsymbol{B}| \\ &\geq \boldsymbol{J} \cdot \boldsymbol{B} + \delta \end{aligned}$$

et

$$\begin{aligned} |\boldsymbol{J}'|^2 &= \{\boldsymbol{J} + [2M(\boldsymbol{x}) - 1]\boldsymbol{x}\}^2 \\ &= \boldsymbol{J}^2 + 2\ \mathrm{sgn}(\boldsymbol{B} \cdot \boldsymbol{x})\boldsymbol{J} \cdot \boldsymbol{x} + \boldsymbol{x}^2 \\ &\leq \boldsymbol{J}^2 - 2\ \mathrm{sgn}(\boldsymbol{J} \cdot \boldsymbol{x})\boldsymbol{J} \cdot \boldsymbol{x} + N + 1 \\ &\leq \boldsymbol{J}^2 + N + 1 \end{aligned}$$

Il faut remarquer ici que nécessairement $sgn(B.x) = -sgn(J.x)$ car $S(x)$ et $M(x)$ ne sont pas en accord, c'est pour cela qu'il y a modification.

Dans le cas où il y a accord il n'y a pas de modification du tout et $\Delta J = 0$.

Après n étapes de modification de ce genre, la répétition des inégalités précédentes donne :

$$\boldsymbol{J}(n) \cdot \boldsymbol{B} \geq \boldsymbol{J}(0) \cdot \boldsymbol{B} + n\delta \qquad |\boldsymbol{J}(n)|^2 \leq |\boldsymbol{J}(0)|^2 + n(N+1)$$

✓ Définissons maintenant :

$$\omega = \frac{\boldsymbol{J} \cdot \boldsymbol{B}}{|\boldsymbol{J}|}$$

En vertu de l'inégalité de Schwarz $|x \cdot y| \leq |x||y|$ nous avons toujours $|\omega| \leq |\boldsymbol{B}| = 1.$

Mais par ailleurs, après n modifications, il découle que nous devons avoir

$$\omega(n) = \frac{\boldsymbol{J}(n) \cdot \boldsymbol{B}}{|\boldsymbol{J}(n)|} \geq \frac{\boldsymbol{J}(0) \cdot \boldsymbol{B} + n\delta}{\sqrt{|\boldsymbol{J}(0)|^2 + n(N+1)}}$$

A partir de ceci, nous concluons qu'il ne peut y avoir que seulement un nombre fini n d'étapes de modifications. L'algorithme doit s'arrêter à une certaine étape, sinon il y aurait contradiction avec $|\omega(n) \leq 1|$:

$$\lim_{n\to\infty} \omega(n) \geq \lim_{n\to\infty} \frac{\boldsymbol{J}(0) \cdot \boldsymbol{B} + n\delta}{\sqrt{|\boldsymbol{J}(0)|^2 + n(N+1)}} = \infty$$

Si d'autres modifications ne peuvent avoir lieu, le système doit par définition être dans une configuration où $M(x) = S(x),\ \forall x \in \Omega$.

- Le théorème de la convergence du perceptron est un résultat très fort, dans la mesure où il ne dépend que de la séparabilité linéaire de la tâche à apprendre M.
- Une borne supérieure n_{max} pour le nombre n d'étapes de modification requises pour la convergence peut être obtenue en vérifiant à quelle étape (dans le pire des cas) nous entrons en contradiction avec $|\omega(n) \leq 1|$

$$\frac{[\boldsymbol{J}(0) \cdot \boldsymbol{B} + n_{\max}\delta]^2}{|\boldsymbol{J}(0)|^2 + n_{\max}(N+1)} = 1$$

Pour des poids initiaux nuls, c'est-à-dire, pour $J(0) = (0, \dots, 0)$, nous obtenons :

$$n_{\max} = \frac{N+1}{\delta^2}$$

- Bien que ceci n'a pas d'intérêt pratique, du fait que nous ne connaissons pas δ :
  - ✓ Plus δ est petit, et plus donc est grand le nombre de pas d'adaptation dont le perceptron a besoin pour obtenir la meilleure performance.
- Il nous est possible aussi de prendre les vecteurs x dans un ordre fixé, du fait que le théorème de la convergence du perceptron n'exige pas qu'ils soient choisis d'une manière aléatoire.
- Dans ce cas aussi, la présentation d'une question x sera en nombre limité.

8. Synthèse des étapes du raisonnement pour démontrer qu'un réseau de McCulloch et Pitts est capable d'exécuter n'importe quelle application

- Après avoir compris qu'un seul neurone ne peut pas exécuter n'importe quelle opération (l'opération XOR en est un exemple).
- En fait, lorsque $N = 2$, il y a en tout seize opération possibles, et les deux opérations non exécutable sont XOR et non XOR.
- Losque $N > 2$, il y a donc aussi des opérations non exécutables. Il suffit de montrer un exemple. C'est celui déduit de l'opération XOR.
- Reste maintenant à savoir si **un réseau** (cette fois-ci) est capable d'exécuter n'importe quelle opération.
- Il faut pour cela, d'abord cerner et dénombrer toutes les opérations possibles $M: \{0,1\}^N \to \{0,1\}$
- Et démonter que le réseau est capable de les exécuter toutes.
- L'idée est de concevoir une méthode qui nous permet d'exécuter l'opération XOR.
- D'utiliser cette même méthode pour construire un réseau capable d'exécuter toute sorte d'opération.
- L'idée est de concevoir un réseau comportant autant de neurones que de vecteurs de Ω^+.
- Chacun de ces neurones est spécialisé exclusivement sur chacun des vecteurs de Ω^+ (lui et uniquement lui).
- De sorte que pour tout élément de Ω^+ il y a un neurone et un seul qui s'active et ainsi, le réseau donne la valeur 1.
- Pour les vecteurs n'appartenant pas à Ω^+, il n'y a aucun neurone qui s'active et ainsi, le réseau donne la valeur 0.
- De cette manière l'application en question est parfaitement reconstruite.

Comme cette application est quelconque, la démonstration est valable pour toute application $M: \{0,1\}^N \to \{0,1\}$.