Mémoires Associatives

mmc <marc.corsini@u-bordeaux2.fr>

11 octobre 2013

Table des matières

Avertissement : ce document est en cours de rédaction. En particulier l'orthographe et la forme sont parfois douteuses. L'objectif du document est de servir de support de cours au module « Mémoire et Apprentissage » du Master 1 de Sciences Cognitives. Le terme de réseau utilisé fait référence à la notion de réseau de neurones formels.

Chapitre 1

Mémoires associatives

Les réseaux communément appelés *mémoires associatives* ont pour objectif d'apprendre à associer des vecteurs d'entrée à des vecteurs de sortie. Mais si le réseau a appris à associer un vecteur $\mathbf{x}$ à un vecteur $\mathbf{y}$, on attend que tout vecteur $\mathbf{z}$ qui diffère peu de $\mathbf{x}$ devra associer le vecteur $\mathbf{y}$. En d'autres termes deux objets qui diffèrent peu doivent se comporter de manière similaire. L'intérêt est que si l'on fournit une entrée bruitée (par rapport à celle apprise par le réseau), on pourra malgré tout récupérer une sortie correcte.

Pour réaliser ces mémoires, on pourra soit utiliser des réseaux non bouclés, soit des réseaux récurrents, bien que ces derniers donnent de meilleurs résultats. Dans ce chapitre, nous nous focalisons sur les réseaux récurrents les plus simples c'est-à-dire que la sortie du réseau est réinjectée comme entrée au système jusqu'à ce qu'il n'y ait plus de modifications (à une erreur ϵ prés). On dit que l'on cherche un état *stable* du système. Dans ce qui suit, nous allons considérer trois types de mémoires associatives :

1. Les réseaux *hétéro-associatifs* qui ont pour but d'associer à m vecteurs $\mathbf{x}^i$ de dimension n, m vecteurs $\mathbf{y}^i$ de dimension k. C'est-à-dire que l'on cherche à réaliser, par apprentissage une application qui à $\mathbf{x}^i \mapsto \mathbf{y}^i$. Mais, on veut de plus que si on connait un vecteur $\tilde{\mathbf{x}}$ tel que $\| \tilde{\mathbf{x}} - \mathbf{x}^i \|^2 < \epsilon$ alors $\tilde{\mathbf{x}} \mapsto y^i$.
2. Les réseaux *auto-associatifs* qui sont des cas particuliers de réseaux hétéro-associatifs (puisqu'il suffit de prendre $\forall\, i, \mathbf{x}^i = \mathbf{y}^i$). Leur but est de pouvoir corriger des données bruitées, afin de retrouver les données apprises.
3. Les réseaux de *reconnaissance de patrons*. Ces réseaux doivent retrouver le « nom » du patron, c'est-à-dire qu'à une donnée multi-dimensionnelle ils doivent associer un scalaire i qui sera interprété comme le nom recherché.

1.1 Architecture

Ces trois types de réseaux peuvent être implémenté à l'aide d'un réseau mono-couche. Notons $w_{i,j}$ le poids de la connexion allant de la cellule d'entrée i vers la cellule de sortie j, et soit W la matrice des poids de connexions, dans le cas général (hétéro-associatif) la matrice appartient à $\mathcal{M}_{n\times k}$. Si on applique le vecteur ligne $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ au réseau, on peut calculer le vecteur d'activation de sortie $\mathbf{s} = (s_1, \ldots, s_k)$:

$$\mathbf{s} = \mathbf{x}.W$$

On peut ensuite appliquer la fonction de transfert sur chaque cellule de sortie $f(\mathbf{s}) = \mathbf{y}$. Si la fonction de transfert est l'identité, on a juste défini un associateur linéaire dont la sortie $\mathbf{y} = \mathbf{x}W$. On peut généraliser la situation au cas où l'on dispose de m vecteurs à associer. Il suffit de construire la matrice $X \in \mathcal{M}_{m\times n}$ dont la ligne i est constituée par le vecteur $\mathbf{x}^i$; de manière similaire on construit la matrice $Y \in \mathcal{M}_{m\times k}$ dont la $i$ème ligne correspond au vecteur $\mathbf{y}^i$. On cherche donc la matrice W qui vérifie :

$$XW = Y$$

Si, de plus on a $m = n$ et que la matrice X est inversible, ce qui n'est pas toujours le cas. La solution pour le cas hétéro-associatif est :

$$W = X^{-1}Y$$

Si maintenant on suppose que l'on travaille sur un réseau récurrent, cela signifie que la sortie est réinjectée dans le système (sous réserve que l'on s'intéresse à un auto-associateur qui fonctionne de manière synchrone) ; dans ce cadre on cherche à déterminer un point fixe ξ qui vérifie :

$$\xi W = \lambda.\xi$$

Cela revient à chercher un vecteur propre de la matrice W associé à la valeur propre λ.

1.2 Apprentissage

L'objectif est d'utiliser les mémoires associatives comme des systèmes dynamiques, dont les états stables (on dit aussi attracteurs) sont les patrons que l'on veut stocker. Malheureusement dans le cas des associateurs linéaires on s'aperçoit qu'un seul attracteur est possible [6, p. 313-314]. C'est pour résoudre ce problème que l'on est conduit à utiliser des fonctions de transferts non linéaires pour obtenir des systèmes dynamiques non-linéaires. L'une des méthodes les plus simples consistent à prendre des vecteurs dont les valeurs sont dans l'ensemble $\{-1, 1\}$ et de choisir comme fonction de transfert une fonction signe comme :

$$\text{sgn}(x) \quad = \quad \begin{cases} 1 & \text{si} \quad x \geq 0 \\ -1 & \text{si} \quad x < 0 \end{cases} \tag{1.1}$$

Prendre des valeurs bi-polaires au lieu de valeurs binaires $\{0, 1\}$ permet d'augmenter la probabilité d'avoir des vecteurs orthogonaux.

1.2.1 Apprentissage Hebbien

Supposons un réseau avec une seule couche de connexions, et k cellules de sortie, dont la fonction de transfert est la fonction signe définie par l'équation 1.1. On cherche à trouver les poids des connexions permettant de faire correspondre un vecteur $\mathbf{x}$ de dimension n à un vecteur $\mathbf{y}$ de dimension k. La règle d'apprentissage la plus simple que l'on puisse utiliser est celle proposée par D. Hebb en 1949 qui exprime la variation d'une connexion $w_{i,j}$ sous la forme $\Delta w_{i,j} = \gamma x_i y_j$, où γ est le coefficient d'apprentissage. La matrice $W = \mathbf{x} \otimes \mathbf{y}$, permet de faire correspondre le vecteur $\mathbf{x} \neq \mathbf{0}$ au vecteur $\mathbf{y}$

$$\mathbf{x}^T W = \mathbf{y}(\mathbf{x}.\mathbf{x})$$

Si $\mathbf{x} \neq \mathbf{0}$ alors $(\mathbf{x}.\mathbf{x}) > 0$ et comme $\mathbf{y}$ est bi-polaire, $\text{sgn}(\mathbf{x}^T W) = \text{sgn}(\mathbf{y}) = \mathbf{y}$.

Si l'on souhaite réaliser m associations, il suffit d'appliquer l'apprentissage hebbien sur chaque paire, et effectuer ensuite la somme des matrices résultantes :

$$W = \sum_{i=1}^{m} W^i = \sum_{i=1}^{m} (\mathbf{x}^i \otimes \mathbf{y}^i)$$

Si l'on présente l'un des vecteurs $\mathbf{x}^l$ on obtient :

$$\mathbf{x}^{l^T} W = \mathbf{y}^l(\mathbf{x}^l.\mathbf{x}^l) + \sum_{\mu \neq l} \mathbf{y}^\mu(\mathbf{x}^\mu.\mathbf{x}^l)$$

Le terme croisé $\sum_{\mu \neq l} \mathbf{y}^\mu(\mathbf{x}^\mu.\mathbf{x}^l)$ s'il est proche de $\mathbf{0}$ garantit que l'on retrouve le vecteur $\mathbf{y}^l$. Le terme croisé aura cette propriété si les vecteurs sont choisis de façon aléatoire et en petit nombre.

Afin d'illustrer notre propos, nous avons repris l'expérience proposée par [6, p. 318]. On appelle distance de Hamming entre deux vecteurs (binaires ou bi-polaires) $\mathbf{a}$ et $\mathbf{b}$ le nombre de composantes qui diffèrent [1].

On considère un réseau auto-associatif, et des vecteurs $\mathbf{x}^i$ de taille n. On génère aléatoirement m vecteurs bi-polaires. On construit de manière incrémentale m auto-associateurs indexé de 1 à m ; le $k^{ème}$ auto-associateur contient les k premiers vecteurs aléatoires. À chaque mémoire, on présente l'ensemble des vecteurs $\tilde{\mathbf{x}}^i$ qui vérifie $\mathcal{H}(\tilde{\mathbf{x}}^i, \mathbf{x}^i) < \frac{n}{2}$ et on regarde quelle proportion de vecteur $\tilde{\mathbf{x}}^i$ retrouve $\mathbf{x}^i$ en une itération. L'expérience est faite 1000 fois.

La figure 1.2 reprend l'expérience précédente mais en effectuant au plus cinq itérations

1. $\mathcal{H}(\mathbf{a}, \mathbf{b}) = \frac{1}{2}(n - \sum_{i=1}^{n} a_i b_i)$ pour les vecteurs bi-polaires et $\sum_{i=1}^{n}(1 - a_i)b_i + (1 - b_i)a_i$ pour les vecteurs binaires.

$\mathcal{H}$	1	2	3	4	5	6	7	8	9	10
0	100	100	100	100	100	100	100	100	100	100
1	100	100	90	85	60	60	54.3	56.2	45.5	33
2	100	86.7	64.4	57.2	40	31.8	22.5	23.1	17	13.3
3	100	50	38.6	25.4	13.5	8.3	4.8	5.9	3.1	2.4
4	100	0	9.7	7.4	4.5	2.7	0.9	0.8	0.3	0.2

FIGURE 1.1 – Pourcentage de bons rappels après une itération

$\mathcal{H}$	1	2	3	4	5	6	7
1	100	100	90	85	60	60	54.3
2	100	100	72.6	71.1	41.8	34.8	27.9
3	100	80	48.6	47.5	18.3	10.8	8.5
4	100	42.8	21.9	22.3	6.8	3.7	2

FIGURE 1.2 – Pourcentage de bons rappels après 5 itérations

1.2.2 Pseudo inverse

Dans le cas général on cherche la matrice W vérifiant

$$XW = Y$$

En algèbre linéaire on sait que l'expression $\| XW - Y \|^2$ est minimale pour $W = X^+Y$, avec X^+ la pseudo inverse de X. Si X^{-1} existe on a $X^{-1} = X^+$. En d'autres termes la pseudo inverse est la meilleure approximation de l'inverse d'une matrice. La pseudo inverse vérifie les points suivants :

1. $XX^+X = X$
2. $X^+XX^+ = X^+$
3. X^+X et XX^+ sont symétriques

Par ailleurs, on peut démontrer que la pseudo inverse d'une matrice existe toujours et qu'elle est unique.

Chapitre 2

Réseaux de Hopfield

Pour en savoir plus sur le sujet, je vous renvoie aux livres suivants : [3, chap. 2] ; [5, chap. 2] ; [4, chap. 3] à [1, chap. 5]. Dans cette partie nous aborderons les réseaux de Hopfield sous plusieurs angles :

1. La dynamique : étant donné un réseau, et un état, comment le système évolue-t-il ?
2. Les états stables : étant donné un réseau, qu'a-t-il appris, à quoi peut-il servir ?
3. L'apprentissage : étant donné un ensemble de prototypes, comment les mémoriser pour qu'ils soient états stables du système ?
4. La fonction d'énergie : comment, étant donné un problème d'optimisation, peut-on résoudre ce problème à l'aide d'un réseau de Hopfield ?

2.1 Intention

Introduit en 1982 par J. Hopfield, ces réseaux fonctionnent comme des mémoires associatives (adressables par le contenu). L'idée sous-jacente est de stocker p prototypes (patrons) X^μ, de telle sorte que lorsque l'on présente un nouvel objet Y, le système réponde en produisant l'objet X^k qui ressemble le plus à Y. Dans ce qui suit, les prototypes seront indexés par $\mu = 1, \ldots, p$ et le réseau sera constitué de N cellules indexées par $i = 1, \ldots N$. Dans ce chapitre nous ne considèrerons que des cellules bivaluées à valeurs dans $\{-1, 1\}$ [1], on parle alors d'information bi-polaire. Bien entendu, nous pourrions obtenir le même type de comportement si nous utilisions un système basé sur la distance de Hamming qui calcule le nombre de bits distincts entre deux objets :

$$\sum_i [X_i^\mu (1 - Y_i) + Y_i (1 - X_i^\mu)]$$

L'inconvénient de cette méthode est d'être coûteuse si tant est que le nombre de candidats est important, d'où l'idée d'un système à base de neurones formels permettant de calculer automatiquement cet indice.

2.2 Le modèle

La dynamique du système est décrite par l'évolution des valeurs de sorties des cellules au cours du temps, si on note $s_i(t)$ la valeur en sortie de la cellule i au temps t, on considère :

$$s_i(t+1) = \text{Sign}(\sum_j w_{ij} s_j(t) - \theta_i)$$

où θ_i désigne le seuil associé à la cellule i. La fonction **Sign** est définie par :

$$\text{Sign}(x) \quad = \quad \begin{cases} 1 & \text{si } x \geq 0 \ ; \\ -1 & \text{si } x < 0 \ ; \end{cases}$$

Cette définition n'est pas la seule que l'on rencontre, par exemple, certains auteurs prônent l'utilisation suivante :

1. le passage de $\{0, 1\}$ à $\{-1, 1\}$ se fait par la transformation t(x) = 2x -1.

$$\text{Sign}(x) \quad = \quad \begin{cases} 1 & \text{si } x > 0 \text{ ;} \\ -1 & \text{si } x \leq 0 \text{ ;} \end{cases}$$

D'autres auteurs, préfèrent utiliser une situation intermédiaire :

$$o_j = \text{Sign}(a_j) \quad = \quad \begin{cases} 1 & \text{si } a_j > 0 \text{ ;} \\ -1 & \text{si } a_j < 0 \text{ ;} \\ \text{Ne change pas} & \text{si } a_j = 0; \end{cases}$$

Dans la suite de ce chapitre, nous ignorerons en général le seuil θ_i afin de simplifier les expressions, et sauf mention explicite nous considérerons la première version de la fonction **Sign**. La mise à jour des sorties des cellules peut se faire suivant deux modes :

- à chaque pas de temps, toutes les cellules mettent leur sortie à jour simultanément. On parle alors d'une stratégie *synchrone*.
- Le mode asynchrone :
  1. à chaque pas de temps, ne mettre à jour qu'une seule cellule, en fonction d'un ordonnancement pré-établi.
  2. à chaque pas de temps la mise à jour d'une cellule se fait avec une certaine probabilité, tout en respectant l'hypothèse qu'au bout d'un certain nombre de pas de temps, la probabilité qu'une cellule ait fait une mise à jour soit de 1.

La stratégie *synchrone* est plus adaptée à la simulation avec une centralisation du contrôle, la seconde est plus appropriée au comportement "hardware".

Une fois la dynamique du système fixée, il nous reste encore à étudier son fonctionnement. Le mécanisme mis en jeu est relativement simple, et la simulation par logiciel est triviale ; il s'agit de présenter un prototype au système, sa taille (son nombre de composants) correspondant très exactement au nombre de cellules du réseau, la valeur du protoype va fixer l'état des sorties des cellules à $t = 0$. à partir de là, la dynamique du système entre en jeu, on a démontré que si l'on considérait un réseau de Hopfield asynchrone, le système atteignait un état stable ; si par contre on choisissait une mise à jour synchrone, le système pouvait aboutir soit à un état stable du système, soit se mettre à osciller entre deux états.

Exemple 1

Considérons le système de Hopfield décrit par la matrice des poids suivante :

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

et supposons que l'on présente en entrée le patron : $\overrightarrow{X}^T = (1 \ -1 \ -1)$. Nous allons étudier comment évolue le système dans trois situations les deux premières sont asynchrones, la troisième est synchrone. Nous fixons à priori les ordres $(1, 2, 3)$ et $(3, 2, 1)$. L'opérateur $\rightsquigarrow$ servira à indiquer un changement d'état, si cet opérateur est indexé par i (ex. $\rightsquigarrow_2$), alors seule la cellule i (ex. 2) sera mise à jour.

1. Les mises à jour s'effectuent successivement pour les cellules 1, 2, puis 3.

$$\begin{array}{llllllll} (1\ -1\ -1) & \rightsquigarrow_1 & (\mathbf{-1}\ -1\ -1) & \rightsquigarrow_2 & (-1\ \mathbf{1}\ -1) & \rightsquigarrow_3 & (-1\ 1\ \mathbf{-1}) & \rightsquigarrow_1 \\ (\mathbf{1}\ 1\ -1) & \rightsquigarrow_2 & (1\ \mathbf{1}\ -1) & \rightsquigarrow_3 & (1\ 1\ \mathbf{1}) & \rightsquigarrow_1 & (\mathbf{1}\ 1\ 1) & \rightsquigarrow_2 \\ (1\ \mathbf{1}\ 1) & \rightsquigarrow_3 & (1\ 1\ \mathbf{1}) & & & & & \end{array}$$

2. Les mises à jour s'effectuent dans l'ordre 3, 2 puis 1.

$$(1\ -1\ -1) \rightsquigarrow_3 (1\ -1\ \mathbf{1}) \rightsquigarrow_2 (1\ \mathbf{1}\ 1) \rightsquigarrow_1 (\mathbf{1}\ 1\ 1) \rightsquigarrow_3 (1\ 1\ \mathbf{1}) \rightsquigarrow_2 (1\ \mathbf{1}\ 1)$$

3. Mise à jour synchrone :

$$(1\ -1\ -1) \rightsquigarrow (-1\ 1\ 1) \rightsquigarrow (1\ -1\ -1)$$

Sur cet exemple on constate que dans les deux cas asynchrones, le système évolue vers le même état stable, mais que dans le cas synchrone, le système oscille entre deux états. Si on change légèrement la matrice (laisser en exercice), qu'en est-il de l'évolution du système. On partira, bien entendu du même état initial $(1\ -1\ -1)^T$.

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

2.2.1 Bilan

Plusieurs mise à jour sont possibles, synchrones vs asynchrones. Le premier problème qui se pose est de pouvoir montrer que la série de mises à jour se termine. J. Hopfield a montré que pour une matrice de connexions symétrique, ces mises à jour s'arrêteront au bout d'un certain moment. On parle de convergence vers un attracteur, malheureusement cet attracteur n'est pas forcément optimal, et surtout, en fonction de la mise à jour il pourra être différent. Par exemple pour la matrice de poids suivantes :

$$\begin{pmatrix} 0 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 \\ 0 & -2 & 0 & 0 \\ -2 & 0 & 0 & 0 \end{pmatrix}$$

En partant de l'état initial $(-1\ -1\ -1\ -1)^T$, si l'ordre de mise à jour est 2 puis 1. L'attracteur sera $(1\ 1\ -1\ -1)^T$. Alors que si l'ordre est 2, puis 4. L'attracteur sera $(-1\ 1\ -1\ 1)^T$.

2.3 Apprentissage

Nous allons maintenant étudier comment obtenir les poids des connexions $w_{i,j}$ pour que les états stables du système soient, autant que possible, des états appris par le réseau. Pour cela nous allons procéder en deux étapes.

2.3.1 Un seul prototype

Dans le cas où l'on ne souhaite stocker qu'un seul prototype (et le retrouver) Il suffit que l'on ait :

$$\text{Sign}(\sum_j w_{ij} X_j) = X_i, \forall\ i$$

Il est facile de voir que cette propriété est vérifiée dès que w_{ij} est proportionnel à $X_i X_j$, puisque l'on a $X_j X_j = 1$.

Qui plus est, si moins de la moitié des valeurs du vecteur initial S, diffère de X, le système se stabilisera sur X. Tout se passe comme si les X_i étaient des attracteurs du système. En fait, il existe deux attracteurs qui sont X et −X.

Exemple 2
Supposons que l'on dispose du patron suivant $X = (1\ -1\ 1\ -1)^T$, on obtient le réseau de Hopfield suivant :

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

Considérons maintenant la donnée $Y = (1\ 1\ 1\ -1)^T$ qui ne diffère de X que d'une valeur. Le produit $Y^T H = (1\ -3\ 1\ -1)$, qui après application de la fonction de transfert donne X^T, ce vecteur étant différent du vecteur initial Y, on réitère le processus. C'est-à-dire que l'on calcule $X^T H = (3\ -3\ 3\ -3)$ qui après application de la fonction de transfert donne X^T. Le calcul s'arrête, puisque l'on a atteint un état stable.

La propriété peut être établie formellement en développant chaque terme, en considérant S_i la sortie de la cellule i, et Y_j l'entrée de la cellule j :

$$\begin{aligned} S_i &= \text{Sign}(\sum_{j=1}^{n} w_{ij} Y_j) \\ S_i &= \text{Sign}(\sum_{j=1}^{n} X_i X_j Y_j) \\ S_i &= \text{Sign}(X_i \sum_{j=1}^{n} X_j Y_j) \\ S_i &= \text{Sign}(X_i \sum_{j=1}^{p} X_j X_j - \sum_{j=p+1}^{n} X_j X_j) \\ S_i &= \text{Sign}(X_i[p-(n-p)]) \\ S_i &= X_i \text{ si } 2p > n \end{aligned}$$

2.3.2 Plusieurs prototypes

Dans le cas où l'on a plusieurs prototypes à stocker, la réponse la plus simple à ce problème est de considérer les poids des connexions comme la superposition des différents patrons :

$$w_{ij} = \frac{1}{p} \sum_{\mu=1}^{p} X_i^\mu X_j^\mu$$

Dans les exemples traités ici, on ignorera le facteur $\frac{1}{p}$. Cette règle est communément appelée règle de Hebb généralisée du fait de sa ressemblance avec l'hypothèse faite par D. Hebb (1949) sur la manière dont la force synaptique des connexions entre neurones du cerveau évolue en fonction de la corrélation de comportement entre deux neurones connectés.

On peut démontrer que la capacité de stockage d'un réseau de N cellules est d'environ $p_{\max} \approx 0.15N$ [3] si les prototypes sont aléatoires et que l'on souhaite une erreur de récupération inférieure à 0.01 Nous verrons comment lever cette limitation dans la section 2.5.

2.4 Fonction d'énergie

L'un des apports majeurs de Hopfield fut l'idée d'introduire une fonction d'énergie dans la théorie des réseaux de neurones formels. La fonction choisie par J. Hopfield (dans le cas asynchrone) est :

$$H(X) = -\frac{1}{2} \sum_i \sum_j w_{ij} X_i X_j$$

On peut généraliser cette équation dans le cas où l'on a des seuils associés à chaque cellule :

$$H(X) = -\frac{1}{2} \sum_i \sum_j w_{ij} X_i X_j - \sum_j \theta_j X_j$$

Ce qui, sous forme matricielle s'exprime par :

$$H(X) = -\frac{1}{2} X^T W X - X^T \mathbf{\Theta}$$

où $\mathbf{\Theta}$ est le vecteur des seuils θ_i. Cette équation peut se réécrire sous la forme :

$$H(X) = C - \sum_{(i,j)} w_{ij} X_i X_j$$

où $(i, j) = \forall i, j,\ i < j$, le terme constant, C correspond à la somme des connexions $w_{ii},\ \forall i$.

Nous allons montrer que cette fonction décroit lorsque le système évolue dynamiquement. De fait, un état stable est un état qui correspond à un minimum de la fonction d'énergie. L'appellation de fonction d'énergie vient du parallèle avec les systèmes magnétiques, ceci étant on peut étendre cela aux problèmes d'optimisation où l'on cherche à minimiser la fonction de coût (fonction objective).

Lorsque le système évolue d'un état X vers un état Y, il le fait en accord avec $Y_i = \text{Sign}(\sum_j w_{ij} X_j)$, rappelons que la fonction Sign est définie par :

$$\text{Sign}(x) \quad = \quad \begin{cases} 1 & \text{si} \quad x \geq 0 \\ -1 & \text{sinon} \end{cases}$$

Si on considère une cellule en particulier i on a deux situations : soit $Y_i = X_i$, soit $Y_i = -X_i$.

Pour simplifier, on suppose dans ce qui suit que seule la cellule i_0 change d'état. Calculons la différence $H(X) - H(Y)$

$$
\begin{aligned}
&= [C - \sum_{(i,j)} w_{ij} X_i X_j] - [C - \sum_{(i,j)} w_{ij} Y_i Y_j] \\
&= -\sum_j w_{i_0 j} X_{i_0} X_j + \sum_j w_{i_0 j} Y_{i_0} Y_j \\
&= (-X_{i_0} + Y_{i_0}) \sum_j w_{i_0 j} X_j \\
&= 2 Y_{i_0} \sum_j w_{i_0 j} X_j \\
&> 0
\end{aligned}
$$

La ligne 1 est la retranscription directe de la différence entre les deux valeurs de la fonction d'énergie. La ligne 2 s'obtient en remarquant que si $j \neq i_0$ alors on a $X_j = Y_j$ donc les seuls termes résiduels sont ceux où i_0 intervient. Le passage à la ligne 3, s'obtient par factorisation en remarquant que $Y_j = X_j$, la ligne 4 est due au fait que, par hypothèse, $X_{i_0} = -Y_{i_0}$. Enfin, la dernière s'obtient en remarquant que $Y_{i_0} = \text{Sign}(\sum_j w_{i_0 j} X_j)$, les deux termes étant de même signe, le produit est strictement positif.

Remarque 2.4.1
Si plusieurs cellules avaient changé d'état on aurait obtenu comme différence d'énergie entre les deux états :

$$2 \sum_{i=i_0}^{i_k} Y_i \sum_j w_{ij} X_j$$

Chacun des termes étant positif, la somme est positive.

L'argument précédent ne tient pas lorsque l'on se place dans une mise à jour synchrone. En effet, dans ce cas, tous les élements du réseau sont mis à jour en parallèle, la contribution d'une seule cellule à la fonction d'énergie n'est plus possible à prendre en compte de façon isolée. Dans ce cas, il est plus approprié de considérer la fonction définie par J. Bruck et J. Goodman dans [2] :

$$E_L(t) = -\sum_{i,j} w_{ij} s_i(t) s_j(t-1)$$

qui dépend de l'état du système à deux instants consécutifs de son évolution. On peut démontrer que, si les poids des connexions sont symétriques cette fonction décroît au cours du temps. On constate cependant que l'on atteint soit un état stable, soit un cycle de longueur 2.

Si on note $h_i(t) = \sum_j w_{ij} s_j(t-1)$, et comme $s_i(t) = \text{Sign}(h_i(t))$; on aboutit à :

$$
\begin{aligned}
E_L(t+1) &= -\sum_i \text{Sign}(h_i(t)) h_i(t) \\
&= -\sum_i \text{abs}(h_i(t)) \\
&\leq -\sum_j h_j(t) s_j(t-1) = E_L(t)
\end{aligned}
$$

avec, par ailleurs :

$$E_L(t+1) - E_L(t) = -\sum_{i,j} w_{ij} s_i(t) [s_j(t+1) - s_j(t-1)]$$

Pour terminer, il est possible de retrouver la règle de Hebb en partant de la notion de fonction d'énergie. En effet, prenons le cas d'un seul prototype. On souhaite que l'énergie du système soit minimale lorsque la réponse du réseau est proche de la valeur du prototype. On peut alors définir la fonction d'énergie suivante :

$$H = -\frac{1}{2} (\sum_i S_i X_i)^2$$

Dans le cas où l'on a plusieurs patrons il suffit de modifier légèrement la fonction :

$$H = -\frac{1}{2} \sum_{\mu=1}^{p} (\sum_i S_i X_i^\mu)^2$$

Ce qui, en développant donne :

$$H = -\frac{1}{2}\sum_{\mu=1}^{p}(\sum_{i} S_i X_i^{\mu})(\sum_{j} S_j X_j^{\mu}) = -\frac{1}{2}\sum_{i,j}(\sum_{\mu=1}^{\mu} X_i^{\mu} X_j^{p}) S_i S_j$$

Ce qui est la même expression que celle que nous avions choisie lorsque w_{ij} était construit en utilisant la règle de Hebb.

Exercice 1
Considérez le réseau de l'exemple 1 et calculez les différentes valeur de la fonction d'énergie.

2.4.1 Application

Nous montrons, au travers d'un exemple simple de partionnement d'un graphe, comment on peut recoder le problème afin de le résoudre au moyen d'un réseau de Hopfield.
Supposons donné un graphe non orienté $G = \langle X, E\rangle$, avec card(X) = N = 2n. On souhaite répartir les sommets de X en deux ensembles X_1 et X_2 (de cardinalité n), tel que $X_1 \cap X_2 = \emptyset$, et $X = X_1 \cup X_2$, tout en minimisant le nombre de connexions entre les deux ensembles. Pour cela nous définissons $c_{ij} = 1$ s'il existe une arête entre x_i et x_j. De la même façon, on associe à chaque sommet une variable s_i qui vaut 1 si $x_i \in X_1$ et -1 sinon. Le problème de partitionnement du graphe revient à minimiser la quantité $L = -\sum_{(i,j)} c_{ij} s_i s_j$, tout en ayant $\sum_i s_i = 0$. Ce qui nous conduit à choisir $H = L + \mu(\sum_i s_i)^2$, avec $0 \leq \mu \leq 1$. En développant on trouve :

$$\begin{aligned} H &= L + \mu[\sum_i s_i^2 + 2\sum_{(i,j)} s_i s_j] \\ &= \mu N - \sum_{(i,j)}(c_{ij} - 2\mu) \end{aligned}$$

Résoudre ce problème revient à prendre :

$$\begin{aligned} w_{ii} &= \mu \\ w_{ij} &= c_{ij} - 2\mu \end{aligned}$$

2.5 Retour sur Hopfield synchrone

En fait comme les réseaux de Hopfield sont un cas particulier de mémoires associatives, on pourra se reporter au chapitre 1 pour voir ce qu'il en est dans un cas plus général. On trouvera aussi comment en s'appuyant sur des propriétés d'algèbre linéaire, on peut sensiblement améliorer les performances d'un réseau de Hopfield synchrone.

2.6 Extensions du modèle

Il existe plusieurs extensions du modèle de Hopfield, l'une d'entre-elles est connue sous le nom de *machine de Boltzman*. On trouve par ailleurs une extension (simple) au cas continu, en remplaçant la fonction signe du cas discret par la tangeante hyperbolique qui est à valeur continue dans l'intervalle [-1,1].

Bibliographie

[1] Hervé Abdi. *Les réseaux de neurones*. Sciences et technologies de la connaissance. PUG, 1994.

[2] J Bruck and J. W. Goodman. A Generalized Convergence Theorem for Neural Networks and Its Application in Combinatorial Optimization. *IEEE Trans. Inform. Theory*, 34(1089), 1988.

[3] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the theory of neural computation*. Addison-Wesley, 1991.

[4] B. Müller, J. Reinhardt, and M. T. Strickland. *Neural Networks an Introduction*. Springer Verlag, 1995.

[5] Jean-Pierre Nadal. *Réseaux de neurones : de la physique à la psychologie*. Armand Colin, 1993.

[6] R. Rojas. *Neural Networks : a systematic introduction*. Springer Verlag, 1996.

Mon Index