Chrysostome Baskiotis

INTELLIGENCE ARTIFICIELLE

INTELLIGENCE ARTIFICIELLE COMPUTATIONNELLE

EISTI

Année 2013

INTRODUCTION

L'intelligence artificielle computationnelle (IAC), qui est un terme controversé mais, faute de mieux, on peut l'utiliser, regroupe principalement les algorithmes génétiques et évolutionnistes, les réseaux des neurones, les machines à support vectoriel ou, encore, l'ada-boost.

Le mot « computationnelle » est ici utilisé pour distinguer ces méthodes, qui procèdent à des calculs numériques des méthodes de l'intelligence artificielle symbolique qui utilisent des calculs symboliques, comme par exemple, l'algorithme " candidate elimination "ou les arbres de décision. On trouve aussi d'autres appellations pour ce cours, à savoir *intelligence artificielle sous-symbolique* ou *intelligence artificielle numérique*. Il faut aussi se garder de confondre ce cours avec l'*intelligence computationnelle* qui est une discipline beaucoup plus large, regroupant par exemple, en plus, les agents intelligents, les systèmes flous, les réseaux bayesiens, et autres. Sous un certain aspect on peut considérer que l'intelligence artificielle computationnelle est une partie de l'intelligence computationnelle.

L'essentiel de problèmes qu'on essaie de résoudre avec les méthodes de l'IAC sont des problèmes d'optimisation ou des problèmes qui peuvent se transformer en problèmes d'optimisation. Bien évidemment, il y a des méthodes analytiques de résolution de problèmes d'optimisation, comme par exemple les méthodes du gradient ou les multiplicateurs de Lagrange, etc. . Les méthodes de l'IAC s'appliquent dans des cas assez courants dans la pratique et dans lesquels nous ne connaissons pas la fonction à optimiser mais seulement ses valeurs à différents points, ou quand la fonction possède plusieurs optima locaux, ou lorsqu le nombre de dimensions de la fonction est grand, ou quand il y a du bruit dans les mesures, etc. Ainsi le minimum de connaissances que l'on doit avoir pour des telles méthodes est un ensemble des valeurs de paramètres d'entrée à différents points de fonctionnement du système.

Les méthodes de l'IAC que nous allons étudier dans ce cours sont des méthodes supervisées. Les caractéristiques communes de ces méthodes sont les suivantes :

- on dispose soit d'un ensemble des données relatifs à différents points du processus, c-à-d. des valeurs pour les variables d'entrée et la valeur de la sortie, que l'on appelle

population d'apprentissage, soit une méthode pour créer de tels points ;
- on applique une de méthodes de l'IAC et on obtient un résultat calculé ;
- on compare ce résultat calculé au résultat correspondant issu de la population d'apprentissage, dans le cas où celle-ci existe, ou au meilleur résultat déjà obtenu dans le cas où il n'y a pas de population d'apprentissage. Cette comparaison fournit une *valeur de performance* ou *d'adaptation* (fitness value) du résultat calculé ;
- en exploitant la valeur de performance, la méthode permet la modification des paramètres des variables, dans le cas où il y a une population d'apprentissage, ou la création des nouvelles données dans l'autre cas, avec comme objectif, dans les deux cas, d'améliorer la valeur de performance ;
- au terme de ce processus, il n'est pas garanti qu'on atteint l'optimum mais seulement un point au moins sous-optimal (ce qui n'exclut pas d'obtenir aussi le point optimal).

Une autre caractéristique de ces méthodes, est leur approche pragmatique de chaque problème. Comme il s'agit des méthodes qui mettent en application plus des principes que des théories mathématiques, on ne peut pas, pour chaque problème, avoir une démarche identique quant aux réglages de différents structures et paramètres initiaux. Le plus souvent il n'existe pas, pour ces réglages, une méthodologie établie une fois pour toutes et donc on est obligé,en règle générale, de les faire "à la main".

Il est tentant, sous ces conditions, de chercher à trouver la meilleure solution, le meilleur modèle pour tous les problèmes. Pour calmer ces ardeurs, on signale l'existence du « *no-free lunch theorem* » ou de « *conservation law for generalization performance* » (Wolpert) qui stipule que quelque soit la méthode utilisée, cette méthode ne se comportera pas mieux sur la totalité des problèmes qu'un système de choix aléatoire des résultats. En fait une méthode se comporte bien seulement sur une petite partie des problèmes pouvant lui être associés et ses bonnes performances dans cette partie seront compensées par les mauvaises performances sur les autres problèmes. La conclusion qui s'impose est que chaque méthode a ses propres particularités et son propre domaine de compétence.

Par conséquent pour chaque problème que nous aurons à traiter, il faut trouver la ou les méthodes les plus adaptées et régler leurs paramètres pour obtenir la solution. Ceci n'est pas une mince affaire et l'objectif de ce cours est de permettre à l'élève, en appliquant des méthodes de l'IAC, de démêler ce qui peut être extrait de la malice des données.

Références

C. SCHUMACHER, M. D. VOSE, L. D. WHITLEY :, The No Free Lunch and problem description length, in L. SPECTOR ET AL. (Eds.) : *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pp. 565-570, Morgan Kaufmann, 2001

D. H. WOLPERT : The lack of a prior distinction between learning algorithms and the existence of a priori distinctions of learning algorithms, *Neural Computation*, vol.9, pp. 1341-1421, 1996

D. WOLPERT, W. MACREADY : No Free Lunch Theorems for optimization, IEEE Transactions on Evolutionary Computation, vol. 1, p. 67, 1997

1

INTRODUCTION AUX ALGORITHMES GÉNÉTIQUES

Les algorithmes génétiques (AG) constituent une alternative aux méthodes analytiques d'optimisation et ils n'ont pas les faiblesses et les limitations de celles-ci. Ces algorithmes sont fondés sur le principe de l'évolution qui permet la survie de l'élément le plus adapté. Leur fonctionnement est le suivant : sur une population d'individus on effectue, à l'aide d'un schéma de selection qui favorise les individus les plus adaptés, un choix d'un sous-ensemble d'individus. Les éléments de ce sous-ensemble subissent une suite de transformations et donnent ainsi naissance à de nouveaux individus qui vont former la prochaine génération sur laquelle nous allons répéter la même procédure. On peut démontrer que la population converge en loi vers une population constituée d'un seul individu répeté autant de fois qu'il y a d'éléments dans la population. Il est à noter que cet individu n'est pas obligatoirement le meilleur, l'optimal. On espère seulement qu'en répétant cet algorithme plusieurs fois, on aboutira au moins une fois à cet individu optimal.

Ce chapitre présente les structures et le formalisme des algorithmes génétiques.

1.1 Structure des algorithmes génétiques

L'élément de base des AG est le *gène* qui permet de construire des *chromosomes*.

DÉFINITION 1.1.1 *Un* gène *x est une structure simple qui code une caractéristique sous une forme adaptée. Pour la même caractéristique, un gène peut avoir plusieurs valeurs différentes. Chacune de ces valeurs s'appelle* allèle.
Un chromosome $\mathbf{x}$ *est un vecteur de gènes* $\mathbf{x} = [x_1, \ldots, x_n]$.

Le *gène* est la quantité élémentaire d'information qui sera propagée (héritée) lors du passage d'une population à la suivante. En biologie, une cellule contient un ensemble des chromosomes, le *génome*, qui est en fait le vecteur essentiel pour la transmission de l'information (hérédité). La réalisation concrète d'un génome pour une cellule s'appelle *génotype*. On considère en fait le génotype d'un individu comme étant la disposition des symboles du codage des gènes dans le chromosome. Les algorithmes génétiques n'utilisent pas la notion du génome, ni celle du génotype. Ainsi pour les algorithmes génétiques une population sera formée des chromosomes et l'information se propagera de population en population en utilisant les chromosomes comme vecteur de transmission de cette information.

Un chromosome ne passe pas d'une population à la prochaine de façon automatique. Il est d'abord, soumis à une procédure de selection en vue de former l'ensemble des parents de la nouvelle population. Ensuite, cet ensemble subira des transformations afin de créer la prochaine population. Tous les éléments de la population seront évalués selon une *fonction de performance* ou d'*adaptation*. On arrête le déroulement de l'algorithme si un nombre prédéterminé d'étapes a été franchi ou si la valeur maximale (resp. minimale) de la fonction de performance n'évolue pas de population en population.

De façon succincte, un algorithme génétique se déroule comme suit :

1. *Initialisation* : On forme aléatoirement la population initiale des chromosomes.
2. *Évaluation* : Pour chaque élément de la population on calcule sa valeur pour la fonction de performance.
3. *Test d'arrê*t : Si un critère d'arrêt est satisfait, on sort de l'algorithme en fournissant le meilleur élément et sa performance.
4. *Ensemble des parents* : En utilisant la dernière population créée et la fonction de performance on forme un ensemble de parents qui seront utilisés pour la nouvelle population.
5. *Nouvelle population* : Par l'intermédiaire des opérations de transformation, on fabrique à partir de l'ensemble des parents, une nouvelle population de chromosomes.
6. *Répetition* : On recommence l'algorithme à partir de la 2e étape.

Nous présentons par la suite de manière détaillée, chacune de ces étapes.

1.2 Initialisation

Pour former une population de chromosomes, on doit d'abord décider du codage de l'information par les gènes et former aussi une distance entre les chromosomes.

Pour le codage des gènes, nous avons quatre possibilités :

- *codage binaire* : chaque gène prend deux valeurs 0 ou 1. Ainsi un chromosome est un vecteur de chiffres binaires qui traduisent en base binaire, une valeur réelle en base décimale.
- *codage binaire de Gray* : le code de Gray assure que deux nombres consécutifs diffèrent dans leur codage binaire que d'un seul bit.
- *codage décimal* : chaque gène prend un chiffre décimal. Ainsi un chromosome représente un nombre réel.

– *codage ordinal* : c'est un ensemble des nombres entiers de 1 à n qui représente une permutation parmi les $n!$ permutations possibles.

Le tableau suivant fournit un exemple de trois premiers codages.

Décimal	Binaire	Code de Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Dans les deux cas de codage binaire, on peut envisager que la distance entre deux chromosomes soit la *distance de Hamming* qui est égale au nombre de gènes qui, étant sur la même place (*locus*) dans le chromosome, ont des valeurs différentes. Ainsi si $\mathbf{x} = [x_1, \ldots, x_n]$ et $\mathbf{y} = [y_1, \ldots, y_n]$ sont deux chromosomes, alors leur distance de Hamming est donnée par

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} | x_i - y_i |$$

Il est à noter que cette distance est euclidienne.

Dans le cas du codage décimal, on peut considérer comme distance la valeur absolue de la différence de deux nombres.

Si nous faisons un codage binaire, le nombre de gènes d'un chromosome est fonction de la précision souhaitée pour les calculs. En effet, supposons qu'un chromosome doit représenter une valeur réelle dans l'intervalle $[a, b]$. Si on considère un chromosome binaire codé sur ℓ bits $\mathbf{x} = [x_{\ell-1}, \ldots, x_0]$, alors le nombre réel correspondant est

$$X = \frac{b-a}{2^\ell - 1} \sum_{k=0}^{\ell-1} 2^k x_k$$

On en conclut que si on veut une précision des calculs inférieure ou égale à Δ, alors il faut que

$$\ell = \log_2 \frac{b-a+\Delta}{\Delta}$$

Par exemple si $a = -100, b = 100$ et $\Delta = 10^{-4}$, il faut prendre $\ell = \log_2 \frac{200+0.0001}{0.0001} = 20.931569 \approx 21$ bits.

Dans le cas d'un problème qui a plus d'une variable, il faut faire attention à ne pas avoir une grande différence de précision entre les différentes variables. Si par exemple nous avons un problème avec deux variables $x_1 \in [-1000, 1000]$ et $x_2 \in [-1, 1]$ et si on veut une précision de 10^{-2} pour x_1, alors il faut prendre $\ell = 18$. Cette valeur de ℓ donne une précision de 10^{-6} à x_2. Dans ce cas il faut faire un changement d'échelle afin de rapprocher les deux intervalles de variation.

Le dernier élément qu'il faut fixer est la taille de la population. Il s'agit d'un paramètre important, directement lié à l'espace de recherche de la solution. Il est convenu à l'initialisation de choisir au hasard la première population. Les éléments de cette population sont donc issus

de l'espace de recherche par un choix aléatoire uniforme, choix qui garantit que l'espace de recherche soit aussi fidèlement que possible représenté dans la population initiale. Il va de soi que plus le nombre d'éléments de cette population est important, mieux l'espace de recherche est représenté. Mais en contrepartie plus la mémoire à utiliser sera importante et plus le temps de calcul sera long. Le choix du nombre d'éléments dans la population initiale résulte donc d'un compromis entre finesse de représentation de l'espace de recherche et vitesse des calculs. Notons aussi que la méthode de ce choix sera la même pour toutes les autres populations qui seront engendrées par la suite, tout au moins pour l'algorithme de base. Dans le cas des chromosomes binaires, si on note par p la probabilité que tous les allèles apparaissent au moins une fois dans chaque élément du chromosome, on a la formule

$$p = \left(1 - 0.5^{N-1}\right)^{\ell} \tag{1.1}$$

qui pourra être approchée par

$$p \simeq \exp\left(-\ell/2^{N-1}\right) \tag{1.2}$$

Si on fixe la probabilité p (habituellement on prend $p = 99.9\%$), on peut calculer le nombre d'éléments de la population par la formule :

$$N \simeq \left[1 + \log\left(-\ell/\ln p\right)/\log 2\right] \tag{1.3}$$

Par exemple pour une probabilité de $p = 99.9\%$ et longueur du chromosome $\ell = 32$, on a $N = 15.965063 \simeq 16$.

1.3 Évaluation

Pour améliorer le rendement de chaque nouvelle population vis-à-vis du problème considéré, on doit pouvoir évaluer la qualité de chaque chromosome. Pour cette raison on doit disposer d'une *fonction de performance* ou d'*adaptation* (fitness function) qui permettra de mesurer la qualité des chromosomes par rapport au problème considéré. Nous pouvons ainsi envisager que cette fonction représente le *phénotype* du chromosome.

S'il s'agit d'un problème d'optimisation, la fonction de performance peut être la fonction à optimiser. Dans les autres cas, il faut envisager d'établir une fonction dont les valeurs reflètent la spécificité du problème.

1.4 Test d'arrêt

On peut prévoir d'arrêter le déroulement de l'algorithme après avoir effectué un nombre d'itérations fixé d'avance. Néanmoins, il est plus prudent de laisser l'algorithme fonctionner jusqu'au moment où il converge. Convergence ici signifie la réalisation concomitante de deux choses :

1. il y a dans la population un chromosome présent plusieurs fois ;
2. lors du passage d'une population à la suivante, ce même chromosome reste présent.

Si on se trouve dans cette situation et que l'on continue l'algorithme, en fonction des techniques employées pour le renouvellement des populations, tôt ou tard la quasi-totalité d'une population sera composée de ce seul chromosome. Nous pouvons donc arrêter l'algorithme dès que ce phénomène apparaît, sans attendre que toute la population soit composée de ce seul chromosome.

1.5 Ensemble des parents

Une nouvelle population sera élaborée en utilisant un ensemble de parents. Cet ensemble sera issu de la population actuelle selon différentes techniques.

La technique la plus simple consiste à prendre la totalité de la population comme ensemble des parents ou, encore, à choisir les meilleurs chromosomes de la population et de procéder à leur traitement.

Les autres techniques tiennent compte pour décider si un chromosome peut être un parent, de la valeur de sa fonction de performance mais revue par la notion du rang. La population est triée selon la valeur de la fonction de performance. Ainsi la performance associée à chaque chromosome n'est pas la valeur de sa fonction de performance mais son rang dans la liste triée des chromosomes. La performance fondée sur le rang est une méthode plus robuste que la fonction de performance calculée en fonction du problème. Soient n le nombre total de chromosomes d'une population, r_i le rang d'un chromosome $\mathbf{x}_i$, avec $r_i = 1$ pour le chromosome qui est rangé dernier et s est la *pression de sélection* qui est un terme informel égal au rapport de la valeur maximale de la fonction de performance sur la valeur moyenne de cette même fonction pour la population en cours. Si on note par $f_r(\mathbf{x}_i)$ la performance du rang de $\mathbf{x}_i$, on a

- *Rang linéaire* :

$$f_r(\mathbf{x}_i) = 2 - s + 2(s-1)\frac{r_i - 1}{n-1}$$

- *Rang non-linéaire* :

$$f_r(\mathbf{x}_i) = \frac{n y^{r_i - 1}}{\sum_{i=1}^{n} y^{i-1}}$$

où y est calculé comme étant la racine du polynôme

$$(s-n)\,y^{n-1} + s y^{n-2} + \cdots + sy + s = 0$$

Dans la littérature il y a plusieurs approches supplémentaires pour la formation de l'ensemble des parents. En particulier lorsque le nombre de fonctions de performance est supérieur à 1, (optimisation multicritères) auquel cas on applique des méthodes d'optimisation de Pareto. Cet aspect des choses dépasse largement le cadre du présent cours.

1.6 Nouvelle population

L'ensemble des parents sera utilisé pour former la nouvelle population.

On peut envisager d'utiliser un mécanisme d'*élitisme* ou non. L'élitisme consiste à introduire dans la nouvelle population une partie (par exemple 20%) de l'ancienne population, sans modification, en prenant par exemple les chromosomes avec les meilleures performances. Ainsi on perpétue l'existence dans la nouvelle population, des meilleurs chromosomes de l'ancienne au risque de se concentrer sur un optimum local.

Indépendamment de l'élitisme, pour introduire des nouveaux chromosomes à la nouvelle population on fait subir aux chromosomes de l'ensemble des parents trois séries d'opérations. Pour la présentation de ces opérations nous avons besoin des notations et définition suivantes :

DÉFINITION 1.6.1 *Soit la population* Π *qui contient n chromosomes. Nous définissons les deux quantités suivantes :*

1. Fonction de performance *pour le chromosome* $\mathbf{x}$ *:* $f(\mathbf{x})$ *définie en fonction du problème considéré.*
2. Probabilité d'existence d'un chromosome $\mathbf{x}$ *:* $p(\mathbf{x}) = \frac{f(\mathbf{x})}{\overline{f}}$, *où* $\overline{f} = \sum_{\mathbf{x}\in\Pi} f(\mathbf{x})$.

Les opérations sur les chromosomes sont les suivantes :

1. *Sélection* : Pour créer une paire de nouveaux chromosomes on choisit deux chromosomes de l'ensemble des parents à l'aide d'une procédure aléatoire non uniforme. Il y a essentiellement trois procédures de sélection :
   (a) *Roue de la fortune* : La probabilité de choix d'un chromosome $\mathbf{x}$ est proportionnelle à la probabilité $p(\mathbf{x})$ du chromosome.
   (b) *Tournoi à k membres* : On choisit au hasard k chromosomes de l'ensemble des parents. On place dans la nouvelle population le meilleur ou les deux meilleurs chromosomes parmi les k, selon leur fonction de performance.
   (c) *Échantillonnage stochastique universel* : On place sur un segment de droite des intervalles adjacents dont la longueur correspond à la probabilité des chromosomes $p(\mathbf{x})$. Sur cet intervalle on va placer n pointeurs équidistants. Pour placer le premier pointeur sur le segment, on tire un nombre au hasard entre 0 et $1/n$. Soit α le nombre tiré. Le premier pointeur est placé à α, le deuxième à $\alpha + 1/n$, et ainsi de suite. Ainsi chaque pointeur se trouve à l'intérieur d'un intervalle construit précédement. On tire ensuite n nombres entiers au hasard entre 1 et n. Chaque nombre tiré détermine un chromosome de l'ensemble des parents qui devient candidat pour la nouvelle population.
2. *Croisement* : On prend au hasard un couple de chromosomes $\mathbf{x}$, $\mathbf{y}$ qui font partie de la sélection opérée précédement. On applique, selon une probabilité p_c, appelée *probabilité de croisement*, un croisement simple ou double qui fabrique deux chromosomes par croisement de deux chromosomes choisis selon la formule
   (a) *Croisement simple*

$$\begin{array}{lcl} \mathbf{x} = x_1 \dots x_k \dots x_n & \text{croisement} & \mathbf{x} = x_1 \dots x_k y_{k+1} \dots y_n \\ & \text{à la position} & \\ \mathbf{y} = y_1 \dots y_k \dots y_n & k & \mathbf{y} = y_1 \dots y_k x_{k+1} \dots x_n \end{array}$$

(b) *Croisement double*

$$\begin{array}{lcl} \mathbf{x} = x_1 \dots x_k \dots x_l \dots x_n & \text{croisement} & \mathbf{x} = x_1 \dots x_k y_{k+1} \dots y_l x_{l+1} \dots x_n \\ & \text{aux positions} & \\ \mathbf{y} = y_1 \dots y_k \dots y_l \dots y_n & k \text{ et } l & \mathbf{y} = y_1 \dots y_k x_{k+1} \dots x_l y_{l+1} \dots y_n \end{array}$$

Le croisement permet d'exploiter de manière plus approfondie des régions de l'espace de recherche où se trouvent déjà des chromosomes de la population.

3. *Mutation* : À chaque nouveau chromosome issu du croisement, on applique pour chaque gène une *mutation*, c'est-à-dire un changement de sa valeur – de 0 à 1 ou de 1 à 0 dans le cas binaire. Cette mutation s'applique avec une probabilité p_m appelée *probabilité de mutation*. Cette opération permet de partir d'une région de l'espace de recherche et d'aller explorer d'autres régions. Elle introduit ainsi de la diversité dans la population des chromosomes.

Le succès de l'application d'un algorithme génétique dépend des valeurs des différents paramètres utilisés : probabilités, nombre de chromosomes dans la population, nombre d'itérations, réglage du test d'arrêt, codage des informations par les chromosomes, etc. Il n'y a pas de techniques qui permettent la détermination des valeurs de ces paramètres de façon indiscutable. Pour chaque type de problème à résoudre il faut accepter de faire des essais avant d'arriver à une conclusion définitive.

1.7 Exercices

EXERCICE 1.1 *Soit un problème à une variable à résoudre à l'aide d'un algorithme génétique avec codage binaire. Si on suppose que la variable se trouve dans* $[0,5]$ *et on voudrait avoir une précision de 0.001, calculer la longueur* ℓ *du chromosome.*

EXERCICE 1.2 *On code les entiers de 0 à 15 en utilisant 16 chromosomes de longueur* $\ell = 4$.

Pour chaque chromosome parmi les 16, on associe 4 autres chromosomes issus de celui-là et ayant un bit muté. Calculer la moyenne empirique de la différence absolue entre la valeur exacte et les valeurs mutées, ainsi que la variance.

EXERCICE 1.3 *Le codage d'un problème se fait en utilisant un vecteur de longueur* ℓ *et ayant* ℓ *éléments différents. Donc les solutions de ce problème par algorithme génétique sont des permutations de* ℓ *éléments différents. Calculer la proportion des chromosomes de longueur* ℓ *qui sont des solutions valides pour ce problème.*

2

ANALYSE MATHÉMATIQUE DES ALGORITHMES GÉNÉTIQUES

Ce chapitre est dédié à la présentation du substrat mathématique des algorithmes génétiques. Nous ferons juste un développement succinct, sans aller aux détails des différentes propositions dont la présentation complète se trouve dans le livre de Holland. L'objectif est de permettre à l'élève de s'apercevoir que un algorithme génétique, malgré le caractère aléatoire de sa construction, converge en loi vers une solution optimale, qui n'est pas obligatoirement l'optimum global.

2.1 Schémas

Pour étudier la convergence d'un algorithme génétique, on introduit la notion de schéma qui est une notion qui relaxe la notion du chromosome.

DÉFINITION 2.1.1 *Un* schéma $\mathbf{h} = [h_1, \ldots, h_\ell]$ *est un vecteur de gènes, avec le gène* h_i *prenant une de trois valeurs*

$$h_i = \begin{cases} 1 \\ 0 \\ * \end{cases}$$

où $*$ *signifie valeur indéterminée (qui peut être indifféremment* 0 *ou* 1*).*
Pour un schéma $\mathbf{h}$ *donné, on définit*

- *son* ordre $o(\mathbf{h})$ *qui est le nombre de gènes déterminés ;*
- *sa* distance définie $\delta(\mathbf{h})$ *qui est la distance qui sépare le premier gène déterminé du dernier gène déterminé.*

Exemples : Pour le schéma $\mathbf{h} = 100*1**$ nous avons $o(\mathbf{h}) = 4$ et $\delta(\mathbf{h}) = 5-1 = 4$. Pour le schéma $\mathbf{h} = **\mathbf{1}****$ on a $o(\mathbf{h}) = 1$ et $\delta(\mathbf{h}) = 0$.

Un schéma peut donc représenter plusieurs chromosomes et un chromosome peut se retrouver dans plusieurs schémas.

2.2 Populations et schémas

Les schémas permettent l'étude des conséquences des opérateurs de reproduction des algorithmes génétiques. Considérons la population de chromosomes $\Pi(t)$ d'effectif n, à l'itération t de l'algorithme. Considérons aussi un schéma $\mathbf{h}$ particulier et supposons que dans la population il y a $m = m(\mathbf{h},t)$ chromosomes qui sont des exemplaires de ce schéma. Supposons que la formation de l'ensemble des parents à partir de cette population se fait selon la règle de la roue de la fortune.

Un chromosome $\mathbf{x}$ a une probabilité

$$p(\mathbf{x},t) = \frac{f(\mathbf{x},t)}{\sum_{\mathbf{x}'\in\Pi(t)} f(\mathbf{x}',t)}$$

d'être dans l'ensemble des parents, où f est la fonction de performance. Nous pouvons définir la performance moyenne du schéma $\mathbf{h}$ par :

$$f(\mathbf{h},t) = \frac{\sum_{\mathbf{x}\in\mathbf{h}} f(\mathbf{x},t)}{m(\mathbf{h},t)}$$

On suppose que dans la nouvelle population créée à l'itération $t+1$, nous avons $m(\mathbf{h},t+1)$ exemplaires du schéma $\mathbf{h}$ dont la valeur est donnée par la relation

$$m(\mathbf{h},t+1) = m(\mathbf{h},t)\, n \frac{f(\mathbf{h},t)}{\sum_{\mathbf{x}\in\Pi(t)} f(\mathbf{x},t)}$$

Si on note par

$$\overline{f}(t) = \frac{\sum_{\mathbf{x}\in\Pi(t)} f(\mathbf{x},t)}{n}$$

la performance moyenne de la population $\Pi(t)$, nous obtenons

$$m(\mathbf{h},t+1) = m(\mathbf{h},t)\, \frac{f(\mathbf{h},t)}{\overline{f}(t)} \tag{2.1}$$

qui est l'équation aux différences pour l'évolution du nombre d'exemplaires d'un schéma.

Cette relation montre qu'un schéma évolue comme le rapport entre la moyenne de la performance du schéma et la moyenne de la performance de la population. Donc si ce rapport est supérieur à 1, on aura, en probabilité, une augmentation du nombre d'exemplaires du schéma $\mathbf{h}$

aux populations suivantes, et dans le cas contraire il y aura une diminution du nombre d'exemplaires qui conduira à une extinction du schéma de la population.

Pour avoir une idée plus qualitative de la relation (2.1), supposons que la performance moyenne d'un schéma est une portion constante de la performance moyenne, c'est-à-dire que $f(\mathbf{h},t) = \overline{f}(t) + c\overline{f}(t)$, où c constante pour tout t. Donc

$$m(\mathbf{h},t+1) = m(\mathbf{h},t)\,\frac{\overline{f}(t)+c\overline{f}(t)}{\overline{f}(t)} = (1+c)\,m(\mathbf{h},t)$$

En utilisant cette relation pour $t = 0,1,\ldots$, on obtient

$$m(\mathbf{h},t) = m(\mathbf{h},0)\,(1+c)^t$$

Les financiers verront la formule des intérêts composés et les gens normaux une progression géométrique qui indique une croissance ou une décroissance exponentielle. Ce qui signifie que s'il y a augmentation du nombre d'exemplaires d'un schéma **h**, cette augmentation sera rapide et avec un nombre d'itérations relativement petit, c'est-à-dire avec un nombre de nouvelles populations relativement petit, on aura une population presque exclusivement composée d'exemplaires du schéma **h**. Il s'agit d'une éventualité à prendre en considération pour éviter de converger trop rapidement vers une valeur qui n'est pas intéressante.

2.3 Opérations de reproduction et schémas

Nous allons maintenant examiner l'influence des opérations de reproduction – croisement et mutation – à l'évolution d'un schéma.

Considérons une population de n chromosomes où chaque chromosome a comme longueur ℓ. Soit un schéma **h** d'ordre $o(\mathbf{h})$ et de distance définie $\delta(\mathbf{h})$.

Lors d'une opération de croisement le schéma reste intact si la coupure se fait en dehors des gènes qui déterminent la distance définie. Donc la probabilité de survie d'un schéma après un croisement est [1]

$$p_s(\mathbf{h},t) = 1 - \frac{\delta(\mathbf{h})}{\ell - 1}$$

Si, de plus, on suppose que le croisement intervient avec une probabilité p_c de croisement, on aura finalement

$$p_s(\mathbf{h},t) \geq 1 - p_c\frac{\delta(\mathbf{h})}{\ell - 1}$$

En utilisant la relation (2.1), on obtient pour le nombre d'exemplaires d'un schéma après les opérations de selection et de croisement, la relation

$$m(\mathbf{h},t+1) \geq m(\mathbf{h},t)\,\frac{f(\mathbf{h},t)}{\overline{f}(t)}\left(1 - p_c\frac{\delta(\mathbf{h})}{\ell - 1}\right)$$

La dernière opération à examiner est la mutation qui altère la valeur des gènes. Un schéma peut être détruit par une mutation si un gène déterminée est modifié. Donc si aucun de $o(\mathbf{h})$

1. Notons que le généticien Thomas Morgan (1866-1945) fut le premier à définir ces notions en étudiant la transmission des mutations chez les drosophiles. La probabilité de survie d'un schéma est, dans ce cas, mesurée en centimorgans.

gènes déterminés n'est altéré, le schéma survivra. La probabilité de mutation étant p_m, un gène a une probabilité de $1-p_m$ de survie et le schéma $\mathbf{h}$ a une probabilité de $(1-p_m)^{o(\mathbf{h})}$ de survie. D'habitude on a pour la probabilité de mutation p_m une très faible valeur et par conséquent $(1-p_m)^{o(\mathbf{h})} \simeq 1-o(\mathbf{h})\,p_m$. On peut donc compléter la relation précédente et on a le théorème fondamental des algorithmes génétiques :

THÉORÈME 2.3.1 (THÉORÈME DES SCHÉMAS) *Étant donné une population $\Pi(t)$ des chromosomes et un schéma $\mathbf{h}$ d'ordre $o(\mathbf{h})$, de distance définie $\delta(\mathbf{h})$ et pour lequel il y a $m(\mathbf{h},t)$ exemplaires dans la population, le nombre d'exemplaires de ce schéma à la population suivante satisfait à l'inégalité*

$$m(\mathbf{h},t+1) \geq m(\mathbf{h},t)\,\frac{f(\mathbf{h},t)}{\overline{f}(t)}\left(1-p_c\frac{\delta(\mathbf{h})}{\ell-1}+1-o(\mathbf{h})\,p_m\right)$$

Ce théorème montre que pour avoir une augmentation du nombre d'exemplaires d'un schéma d'une population à la prochaine, il ne suffit pas d'avoir une performance moyenne élevée. Il faut de plus avoir une distance définie petite et aussi un ordre petit.

2.4 Exercices

EXERCICE 2.1 *Montrer que tout chromosome de longueur ℓ est un exemplaire de 2^ℓ schémas différents.*

EXERCICE 2.2 *Soient dans un chromosome de longueur ℓ, deux gènes qui se trouvent à des places spécifiques, séparés par $m \geq 0$ autres gènes. Calculer la probabilité qu'ils restent ensemble après*

1. *un croisement en un point ;*
2. *un croisement en deux points.*

EXERCICE 2.3 (DISTRIBUTION DE HARDY-WEINBERG) *Considérons une population composée de*

- *homozygotes AA en proportion p ;*
- *hétérozygotes Aa et aA en proportion $2q$, et*
- *homozygotes aa en proportion r.*

À partir de cette population on engendre une nouvelle population avec croisement et sans selection pour l'ensemble des parents et sans mutation. Donner la distribution des éléments ci-dessus dans la nouvelle population.

EXERCICE 2.4 *Soient les quatre chromosomes suivants*

$\mathbf{x}_1$	*11101111*
$\mathbf{x}_2$	*10101011*
$\mathbf{x}_3$	*00010100*
$\mathbf{x}_4$	*01000011*

et les schémas

$\mathbf{h}_1$	$1*******$
$\mathbf{h}_2$	$0*******$
$\mathbf{h}_3$	$******11$
$\mathbf{h}_4$	$***0*01*$
$\mathbf{h}_5$	$1*****1*$
$\mathbf{h}_6$	$1110****$

1. *Pour chaque schéma donner l'ordre et la distance définie.*
2. *Pour chaque schéma donner le nombre d'exemplaires qui sont dans la population.*

EXERCICE 2.5 *Sous quelles conditions*

1. *l'union ensembliste de deux schémas est un schéma ?*
2. *l'intersection ensembliste de deux schémas est un schéma ?*

EXERCICE 2.6 *Une population est composée des chromosomes suivants avec leurs valeurs de la fonction de performance*

	$\mathbf{x}$	$f(\mathbf{x},t)$
$\mathbf{x}_1$	*10001*	*20*
$\mathbf{x}_2$	*11100*	*10*
$\mathbf{x}_3$	*00011*	*5*
$\mathbf{x}_4$	*01110*	*15*

1. *Soit le schéma* $\mathbf{h}_1 = 1****$ *. Quel est le nombre estimé d'exemplaires de ce schéma à la prochaine population ?*
2. *Même question pour le schéma* $\mathbf{h}_2 = 0**1*$ *.*

EXERCICE 2.7 *Soit la population suivante*

	$\mathbf{x}$	$f(\mathbf{x},t)$
$\mathbf{x}_1$	*100100*	*20*
$\mathbf{x}_2$	*001000*	*20*
$\mathbf{x}_3$	*110111*	*30*
$\mathbf{x}_4$	*100101*	*20*
$\mathbf{x}_5$	*100010*	*10*

Supposons que la probabilité de croisement est $p_c = 0.75$ *et la probabilité de mutation* $p_m = 0.1$*. Calculer le nombre estimé d'exemplaires du schéma* $\mathbf{h} = *\mathbf{0}***\mathbf{0}$ *à la population suivante.*

2.5 Références

Les livres utilisés pour la rédaction de deux premiers chapitres sont les suivants :

[1] L. CHAMBERS (ed.) : *Practical handbook of genetic algorithms, Applications*, vol I, CRC Pres, 1995

[2] L. DAVIS (ed.) : *Handbook of genetic algorithms*, van Nostrand, 1991

[3] D. E. GOLDBERG : *Genetic algorithms*, Addison-Wesley, 1989

[4] J. H. HOLLAND : *Adaptation in natural and artificial systems*, Second edition , MIT Press 1992

[5] Z. MICHALEWICZ : *Genetic algorithms + data structures = evolution programs*, Springer-Verlag,

1992
[6] C. R.REEVES, J. E.ROWE : *Genetic Algorithms : Principles and Perspectives A Guide to GA Theory*, Kluwer, 2002
[7] L. D. WHITLEY, M. D. VOSE (eds) : *Foundations of genetic algorithms*, Morgan Kaufmann, 1995

Pour les élèves intéressés par les méthodes de computation inspirées de la biologie nous proposons deux références :
R. DAWKINS : *Le gène égoïste*, Odile Jacob
P. JOUXTEL : *Comment les systèmes pondent une introduction à la mimétique*, Le Pommier

3

INTRODUCTION AUX RÉSEAUX DES NEURONES ARTIFICIELS

Les réseaux neuronaux artificiels (RNA) traitent de manière computationnelle des informations qui reçoivent en entrée. Leur fonctionnement est fondé sur le modèle de la « boîte noire ». On dispose d'une population d'apprentissage et pour les valeurs des variables en entrée de cette population un RNA produit des sorties qui sont comparées avec les valeurs des variables de sortie réelles, dites *sorties désirées*. À l'aide d'un algorithme, cette comparaison permet d'améliorer les paramètres du réseau jusqu'à obtenir une approximation des sorties désirées qui soit acceptable. Dans ce cas le réseau se comporte comme une boîte noire qui pour un ensemble d'entrées en dehors de la population d'apprentissage, fournit des sorties presque correctes. Il s'agit en fait d'une connaissance phénoménologique qui ne nous informe à rien sur la structure du système que nous sommes en train d'étudier.

3.1 Définitions de base

Le premier exemple d'un neurone formel est donné par McCulloch et Pitts en 1943. Une extension de ce neurone est le neurone *appelé unité logique avec seuil* (threshold logic unit - TLU) dont la définition est la suivante :

DÉFINITION 3.1.1 *Un* neurone formel *est une fonction non-linéaire* $f : \mathbb{R}^{n+1} \times \mathbb{R}^{n+1} \times \mathbb{R} \to \mathbb{R}$ *, appelée* fonction d'activation *et définie par*
$\mathbb{R}^{n+1} \times \mathbb{R}^{n+1} \times \mathbb{R} \ni (x_1, \ldots, x_n; w_1, \ldots, w_n; \vartheta) \longmapsto f(x_1, \ldots, x_n; w_1, \ldots, w_n; \vartheta) \in \mathbb{R}$, *où*

- $x_1, \ldots, x_n$ *les entrées au neurone ;*
- $w_1, \ldots, w_n$ *pondération des entrées ;*
- ϑ *seuil de comparaison, appelé biais (et qui est l'équivalent du terme constant dans une régression linéaire).*

La sortie x du neurone est donnée par la relation

$$x = f\left(\sum_{i=1}^{n} w_i x_i - \vartheta\right) \tag{3.1}$$

On notera par la suite $s = \sum_{i=1}^{n} w_i x_i - \vartheta$ *la quantité sur laquelle s'applique* f *et l'on appellera* état du neurone.

La figure suivante présente le neurone de McCulloch–Pitts.

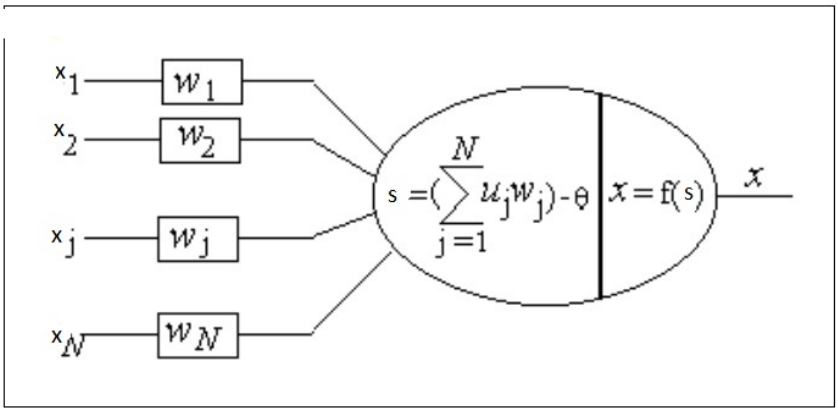

FIGURE 3.1 – Neurone de McCullosh-Pitts

Nous pouvons envisager plusieurs formes pour la fonction d'activation. Nous donnons ci-après les trois formes le plus utilisées dans les cas binaire et bipolaire.

1. Cas binaire, c'est-à-dire $f(s) \in [0,1]$

   (a) Fonction de seuil. Il s'agit du neurone de McCulloch–Pitts donnée par la relation

   $$f(x) = \begin{cases} 1, \text{ si } s > \delta \\ 0, \text{ si } s \leq -\delta \end{cases} \tag{3.2}$$

   où $\delta \geq 0$. S $= 0$, f est la fonction signe.

   (b) Fonction linéaire par morceaux :

   $$f(s) = \begin{cases} 0, \text{ si } s \leq -\delta \\ s, \text{ si } -\delta < s < \delta \\ 1, \text{ si } s \geq \delta \end{cases} \tag{3.3}$$

   (c) Fonction sigmoïde. Elle est très utilisée. Elle pourra être définie de plusieurs façons. Nous donnons la définition à l'aide de la fonction logistique :

   $$f(s) = \frac{1}{1 + e^{-\lambda s}} \tag{3.4}$$

où λ est le paramètre de pente de la fonction. Pour différentes valeurs de λ on obtient des fonctions sigmoïdes avec des pentes différentes. Notons que la pente de la fonction pour $s = 0$ est $\lambda/4$.

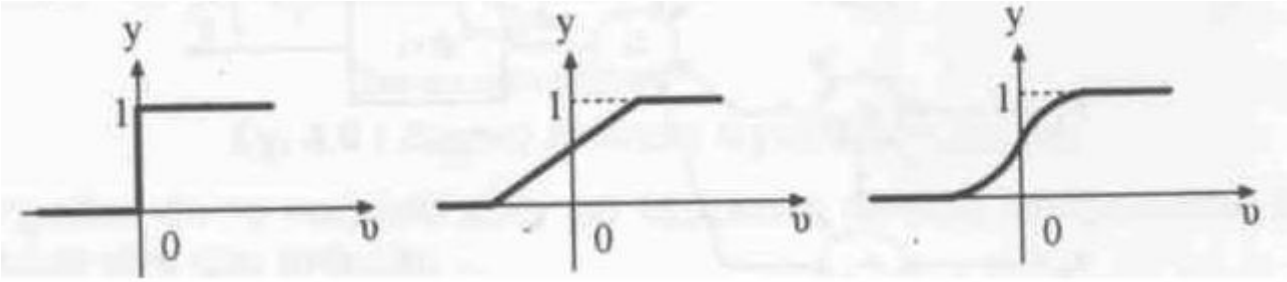

FIGURE 3.2 – Les fonctions d'activation binaires

2. Cas bipolaire, c'est-à-dire $f(s) \in [-1,1]$

   (a) Fonction de seuil ou de signe. Elle est donnée par la relation suivante :

$$\text{sign}(s) = \begin{cases} 1, & \text{si } s \geq 0 \\ -1, & \text{si } s < 0 \end{cases} \quad \text{où } f(s) = \begin{cases} 1, & \text{si } s \geq \delta \\ -1, & \text{si } s < \delta \end{cases} \tag{3.5}$$

   (b) Fonction linéaire par morceaux :

$$f(s) = \begin{cases} -1, & \text{si } s \leq -15 \\ s, & \text{si } -1 < s < 1 \\ 1, & \text{si } s \geq 1 \end{cases} \tag{3.6}$$

   (c) Fonction sigmoïde.On utilise pour sa définition la tangente hyperbolique :

$$f(s) = \text{tgh}\left(\frac{s}{2}\right) = \frac{1-e^{-s}}{1+e^{-s}} \tag{3.7}$$

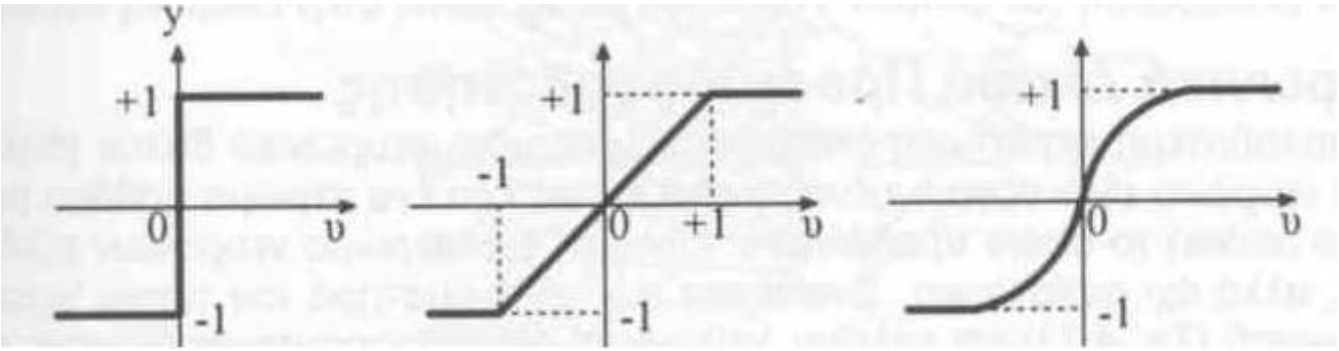

FIGURE 3.3 – Les fonctions d'activation bipolaires

Si on met plusieurs neurones ensemble, alors la sortie de l'un devient une des entrées de l'autre. Si de plus l'ensemble ainsi formé a une structure particulière, alors on parle de *réseau des neurones artificiels*. Nous donnons ci-après la défintion d'un réseau particulier, le *réseau canonique*.

DÉFINITION 3.1.2 *Un* réseau des neurones artificiels $\mathcal{R}$, non recurrent, sous forme canonique, *est composé d'un ensemble de neurones x_j^ℓ disposés en $L+1$ couches, où par x_j^ℓ on note le j-ième neurone de la ℓ-ième couche.*

La couche $\ell = 0$ est la couche d'entrée *et la couche $\ell = L$ est la* couche de sortie.

Chaque couche ℓ est repésentée par le vecteur $\mathbf{x}^\ell \in \mathbb{R}^{n_\ell}$, où par n_ℓ on note le nombre de neurones de la ℓ-ième couche. En particulier on pose $n_0 = n$ et $n_L = m$.

La figure ci-après présente un RNA à une couche cachée

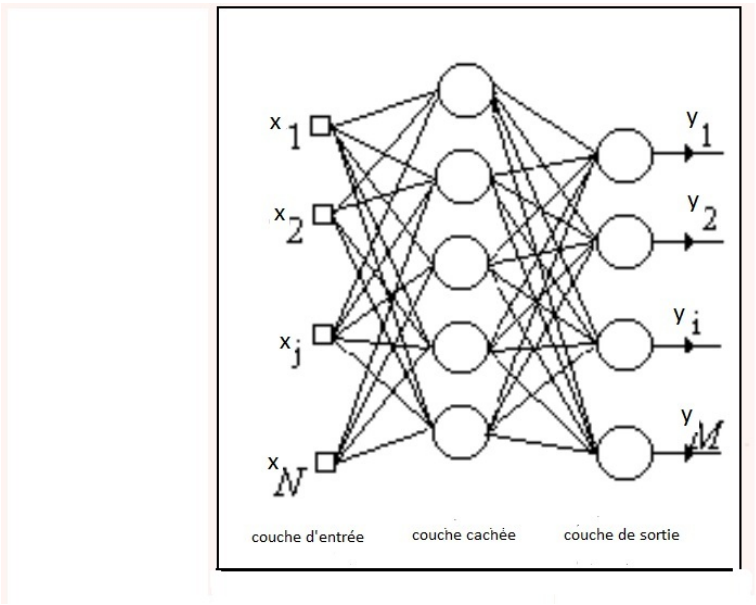

FIGURE 3.4 – RNA avec une couche cachée

À la définition suivante, on reprend la notion du neurone telle qu'elle a été présentée ci-dessus mais en l'appliquant au formalisme qui vient d'être introduit.

DÉFINITION 3.1.3 *Un neurone x_j^ℓ réalise une fonction non-linéaire $f^\ell : \mathbb{R}^{n_{\ell-1}} \times \mathbb{R}^{n_{\ell-1}+1}$ notée par la suite $f^\ell\left(x_1^{\ell-1}, \ldots, x_{n_{\ell-1}}^{\ell-1}, w_{j0}^\ell, w_{j1}^\ell, \ldots, w_{jn_{\ell-1}}^\ell\right)$ et qui s'appelle* fonction d'activation. *Le plus souvent f s'applique à la quantité*

$$s_j^\ell = w_{j0}^\ell + \sum_{i=1}^{n_{\ell-1}} w_{ji}^\ell x_i^{\ell-1};\ j = 1, \ldots, n_\ell,\ \ell = 1, \ldots, L \tag{3.8}$$

*qui est l'*état *du neurone et qui peut encore s'écrire*

$$s_j^\ell = \sum_{i=0}^{n_{\ell-1}} w_{ji}^\ell x_i^{\ell-1};\ j = 1, \ldots, n_\ell,\ \ell = 1, \ldots, L \tag{3.9}$$

si on admet l'existence d'un neurone $x_0^{\ell-1}$ dont la valeur est toujours égale à -1.

$w_{ji}^\ell \in \mathbb{R}$ *est le* poids de la connexion *entre les neurones x_j^ℓ et $x_i^{\ell-1}$. x_j^ℓ est la* valeur d'état *du neurone x_j^ℓ.*

On peut exprimer cette relation sous forme matricielle : On pose

$$\mathbf{W}^\ell = \left[w_{ji}^\ell\right] \in \mathbb{R}^{n_l \times n_{l-1}}$$

et on a

$$\mathbf{s}^{\ell} = \mathbf{W}^{\ell}\mathbf{x}^{\ell-1};\ \ell = 1,\ldots,L \tag{3.10}$$

Dans ce cas on peut avoir la fonction vectorielle d'activation

$$\mathbf{f}^{\ell}\left(\mathbf{s}^{\ell}\right) = \mathbf{f}^{\ell}\left(\mathbf{W}^{\ell}\mathbf{x}^{\ell-1}\right) \in \mathbb{R}^{n_{\ell}};\ \ell = 1,\ldots,L \tag{3.11}$$

et on a

$$\mathbf{x}^{\ell} = \mathbf{f}^{\ell}\left(\mathbf{s}^{\ell}\right)$$

Remarquons que la couche d'entrée n'est pas une couche comme les autres. En fait les entrées au réseau ne sont pas des neurones, elles n'ont pas d'état, ni de fonction d'activation.

Remarquons aussi que cette défintion des RNA, dits canoniques, ne permet pas d'avoir des connexions entre des neurones qui se trouvent sur des couches qui ne sont pas consécutives, ni entre les neurones de la même couche. De plus la définition donnée n'inclut pas les RNA récurrents, c'est-à-dire des RNA où certains neurones bouclent sur eux-mêmes ou sur des neurones des couches précédentes. En réalité ce qui vient d'être dit ne constitue pas une limitation, car tout RNA peut se transformer en RNA canonique. En particulier un réseau réccurent peut se ramener à un réseau canonique dans lequel certaines sorties sont rétroactivement ramenées aux entrées. Cette transformation dépasse le cadre de ce poly.

3.2 Typologie des RNA

On peut classer les RNA de plusieurs façons, à savoir

1. Selon leur structure. La structure topologique est la caractéristique essentielle d'un RNA. Les deux éléments qui déterminent l'architecture d'un RNA sont le nombre de couches cachées et les connexions entre les neurones. Pour le premier élément nous distinguons

   (a) *réseaux sans couche cachée*, et

   (b) *réseaux avec une ou plusieurs couches cachées*.

   Pour le deuxièmeélément nous distinguons

   (a) *réseaux non récurrents* ;

   (b) *réseaux récurrents*.

2. Selon la méthode d'apprentissage. L'apprentissage est la propriété fondamentale des RNA qui les permet d'« apprendre » de leur environnement et donc, d'améliorer leurs performances. Cet apprentissage se fait par la modification des valeurs des poids et des seuils selon un algorithme. En fonction de l'algorithme nous distinguons

   (a) *Apprentissage par correction de l'erreur*. Pour chaque entrée, on calcule l'erreur entre la sortie obtenue par le RNA et la sortie désirée. Cette erreur sert par la

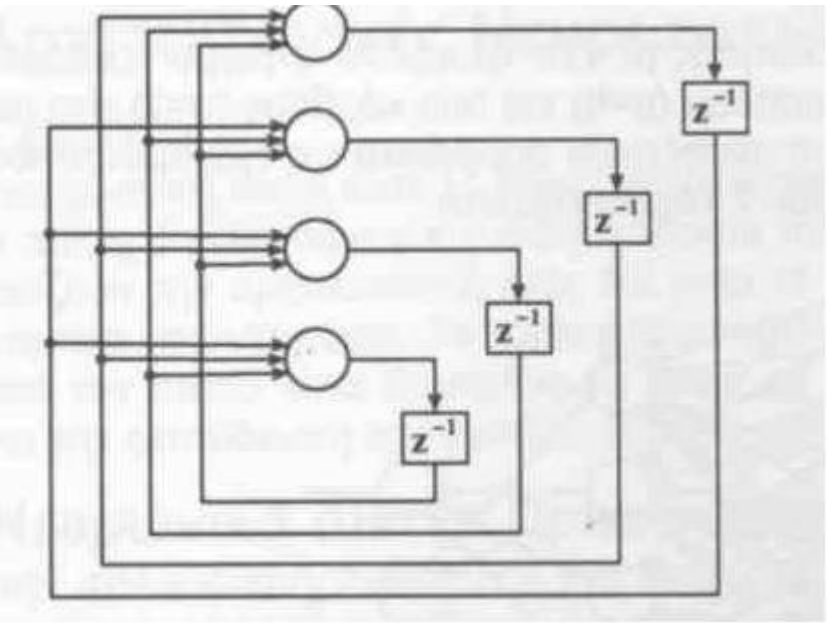

FIGURE 3.5 – RNA récurrent sans couche cachée

suite à adapter les valeurs des poids et des seuils afin que l'erreur dans des calculs futurs soit plus petite.

Il s'agit d'une méthode qui est surtout utilisée pour des problèmes d'optimisation.

(b) *Apprentissage de Hebb*. C'est la plus ancienne des méthodes d'apprentissage en RNA. Selon cette méthode la valeur du poids de la connexion entre deux neurones doit être fonction du produit des valeurs d'activation de deux neurones. Le résultat de cette méthode est que si les deux neurones ont un comportement synchrone([1]), le poids de leur connexion augmente et dans le cas contraire, il diminue.

Cet apprentissage est utilisé dans des applications des mémoires autoassociatives.

(c) *Apprentissage compétitif*. Selon cette méthode, parmi les neurones de sortie, il y a un seul, le *vainqueur*, qui est activé suivant un mécanisme approprié.

On utilise cette méthode surtout pour des problèmes de classification.

(d) *Apprentissage de Boltzmann*. Il s'agit d'une méthode stochastique d'apprentissage. Les neurones sont des structures récurrentes et ils ont deux états $+1$ et -1. Cet apprentissage est caractérisé par une fonction d'énergie et une température qu'il s'agit de ramener à l'équilibre.

3. Selon l'influence de l'environnement. L'environnement peut ou non intervenir lors de l'apprentissage par un RNA, ce qui donne trois possibilités :

1. c'est-à-dire les deux neurones ont des valeurs de même signe

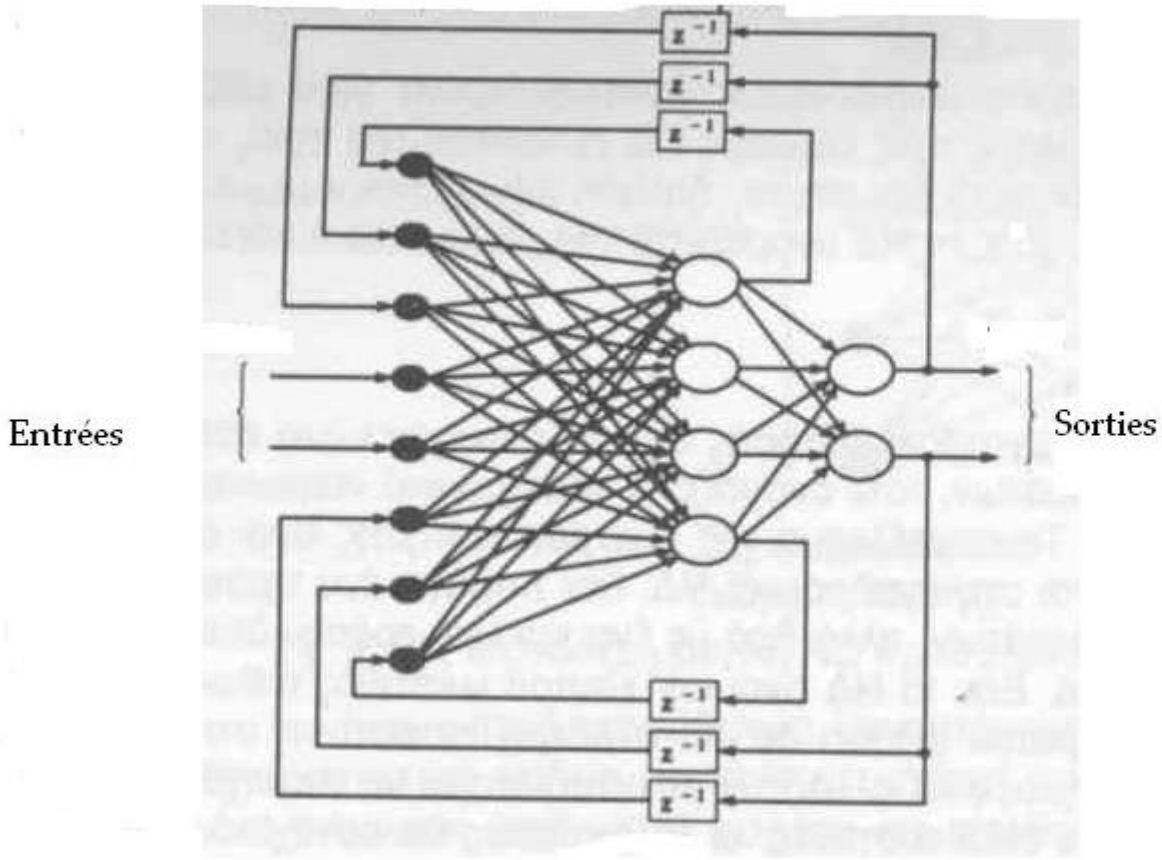

FIGURE 3.6 – RNA récurrent avec couche cachée

(a) *Apprentissage supervisé*. Pour chaque entrée présentée au RNA on connaît la sortie réelle (désirée). La mise à jour des valeurs des poids et des seuils tient compte de la comparaison entre la sortie réelle et la sortie calculée. Cette connaissance de la sortie réelle peut revêtir deux formes :

   i. La connaissance découle d'une fonction connue analytiquement. L'environnement est donc connu et le RNA servira pour effectuer une *approximation optimale* de cette fonction.

   ii. Nous ne connaissons pas la forme analytique de la fonction et la connaissance découle du fait qu'on a plusieurs mesures du couple entrée-sortie. L'environnement est inconnu et le RNA réalisera une *regression non-linéaire*, c'est-à-dire il fournira une estimation des paramètres de cet environnement. Habituellement ces mesures sont entachées de bruit, ce qui rend la convergence du RNA plus difficile.

   Il y a essentiellement deux algorithmes pour l'apprentissage supervisé :

   i. *algorithme des moindres carrée*s, et

   ii. *algorithme de retro-propagation*, RP, (back propagation BP), qui est une généralisation du précédent.

Une limitation importante de l'apprentissage supervisé est sa capacité de généralisation (ou d'inférence), c'est-à-dire sa capacité à calculer correctement la sortie à une entrée donnée. Si l'entrée est dans le domaine de variation des entrées apprises, alors la réponse du RNA est correcte. Mais la qualité de cette réponse se degrade substantiellement si on s'écarte du domaine de variation appris. Cette limitation peut, au moins en partie, être eliminée à l'aide de la méthode suivante d'apprentissage.

(b) *Apprentissage par renforcement*. On introduit une fonction de performance qui s'appelle *signal de renforcement* et qui nous informe sur la qualité de la sortie calculée par rapport à la sortie désirée. Ce signal sera utilisé pour modifier les poids.

(c) *Apprentissage non supervisé*. Nous n'avons aucune connaissance de l'environnement et on tente à apprendre une mesure de qualité de la représentation. Les paremètres du RNA sont optimisés par rapport à cette mesure.
Pour effectuer l'apprentissage non supervisé, on utilise la méthode de l'apprentissage compétitif.

3.3 Tâches d'apprentissage des RNA

Les RNA sont utilisés pour résoudre des problèmes pratiques qui peuvent se transformer en des problèmes équivalents d'apprentissage. Il va de soi que toute topologie, toute méthode et tout algorithme ne conviennent pour la résolution de n'importe quel problème. Nous donnons dans la suite un certain nombre de problèmes représentatifs qui peuvent être résolus par les RNA.

1. *Approximation des fonctions*. Nous avons une fonction non linéaire d'entrée-sortie

$$y = g(\mathbf{x}; \mathbf{w}) \text{ ; où } \mathbf{x} \text{ est l'entrée, } \mathbf{w} \text{ le poids et } y \text{ la sortie} \tag{3.12}$$

On cherche à construire un RNA qui approche la fonction g à l'aide d'un ensemble d'exemples d'entrée-sortie $\left(\boldsymbol{\xi}_\mu, \psi_\mu\right)_\mu$.

Il s'agit d'un apprentissage supervisé. Comme topologie nous pouvons envisager un RNA non récurrent à une ou plusieurs couches cachées et on utilisera un apprentissage par correction d'erreur.

2. *Problèmes d'association*. La tâche d'apprentissage peut prendre deux formes :

   (a) Auto-association. On cherche à construire un RNA qui stocke un ensemble des formes et qui sera capable par la suite, sur présentation d'une de ces formes avec description partielle (c'est-à-dire avec des éléments manquants) ou déformée à cause du bruit, de la retrouver.
   Il s'agit d'un apprentissage non supervisé. Comme topologie nous pouvons envisager un RNA récurrent et on pourrait utiliser un apprentissage de Hebb.

   (b) Hétéro-association. Le problème est le même que en auto-association à ceci près que nous disposons au lieu d'une forme en entrée, d'une forme en entrée associée à une sortie. Bien sûr ici nous avons un apprentissage supervisé.

3. *Classement*. Nous disposons d'un ensemble des classes préexistantes. La tâche du RNA est d'attribuer à un exemple la classe à laquelle il appartient.

   On peut appliquer un apprentissage supervisé, comme aussi un apprentissage non supervisé en fonction du type des données disponibles.

4. *Classification.* La tâche du RNA est de placer en des classes différentes, en nombre inconnu, un ensemble d'exemples. On utilise pour la classification des mesures de similarité entre les éléments de la même classe et, éventuellement, des mesures de dissimilarité entre les éléments des classes différentes.
   Il s'agit d'un apprentissage non supervisé.

5. *Prédiction.* La tâche du RNA est de prédire la prochaine valeur d'une série temporelle dont on connaît ses valeurs jusqu'à l'instant actuel.

   On peut utiliser un apprentissage non supervisé avec comme méthode l'apprentissage par correction d'erreur.

6. *Commande.* Nous disposons d'un système de commande dont les paramètres sont inconnus et pour lequel nous avons de mesures d'entrée $\mathbf{u}(t)$ et de sortie $y(t)$. La tâche d'un RNA est de calculer des entrées telles que le système nous délivre des sorties désirées.

   Il s'agit d'un apprentissage supervisé.

Dans les chapitres suivants nous présenterons pour les deux types de structure des RNA quelques algorithmes caractéristiques ainsi que leurs conditions d'application.

3.4 Références

Pour la redaction de ce chapitre ont été utilisés les livres et articles ci-dessous :

[1] N. K. BOSE, P. LIANG : *Neural network fundamentals*, McGraw-Hill, 1996
[2] G. DREYFUS ET AL. : *Réseaux de neurones*, Eyrolles,
[3] J. A. FRREMAN, D. M. SKAPURA : *Neural networks*, Addison-Wesley, 1991
[4] S. HAYKIN : *Neural networks*, IEEE Press, 1994
[5] B. MÜLLER, J. REINHARDT : *Neural networks*, Springer-Verlag, 1990
[6] R. ROJAS : *Neural networks*, Springer, 1996

4

RNA NON RÉCURRENTS SANS COUCHE CACHÉE

Bien qu'il soit très rare d'avoir un RNA sans couche cachée, nous en donnons dans de chapitre un bref aperçu de ce type de réseau. Il y a pour cela des raisons tant historiques – ce sont les premiers RNA présentés – que théoriques – l'étude des RNA multicouches en découle.

En règle générale les réseaux sans couche cachée sont utilisés pour faire du classement des entrées entre deux classes. Ainsi la sortie y prend deux valeurs : soit 0 et 1, soit -1 et 1.

4.1 Perceptron

Il s'agit d'une amélioration du TLU introduite par Rosenblatt en 1958. Nous avons toujours n entrées, un neurone de sortie dont l'état est une combinaison linéaire des entrées avec les poids. Comme fonction d'activation on utilise la fonction de seuil qui est, selon le type des entrées, soit binaire, soit bipolaire.

Ainsi nous avons pour la valeur de l'état la relation

$$s = \sum_{i=1}^{n} w_i \xi_i - \vartheta = \mathbf{w}^\top \boldsymbol{\xi} - \vartheta \tag{4.1}$$

où $\mathbf{w}^\top$ est le vecteur des poids, $\boldsymbol{\xi}$ le vecteur des entrées et ϑ le biais. Pour la fonction d'activation

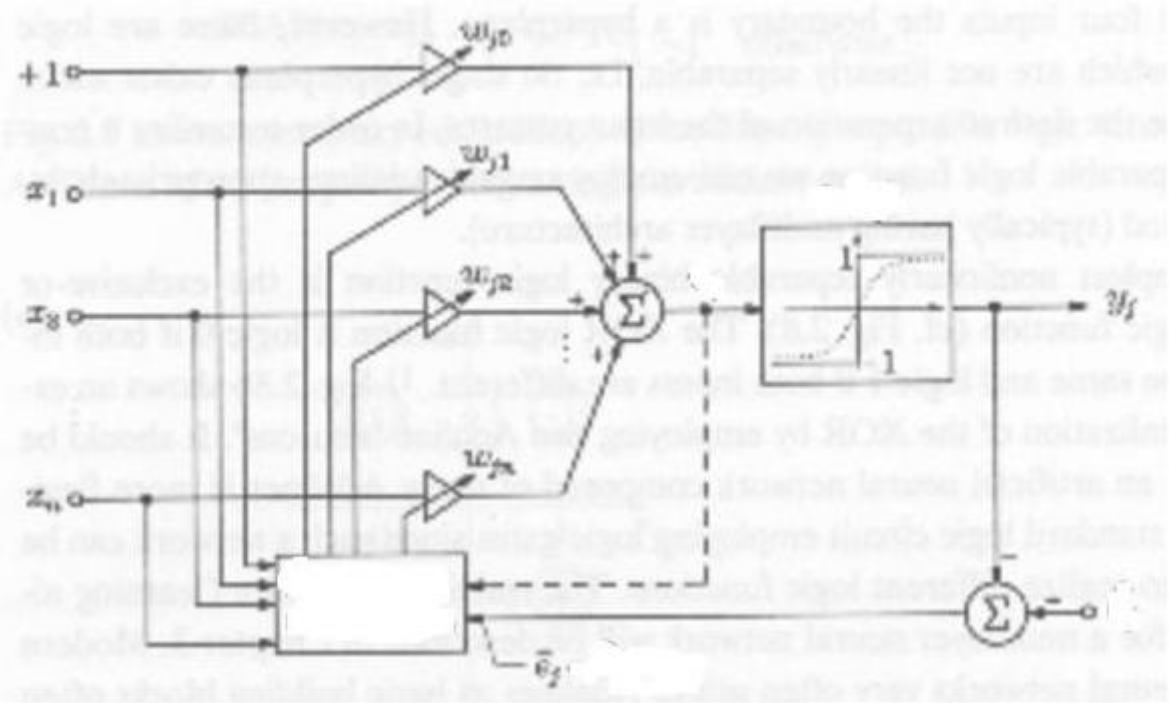

FIGURE 4.1 – Perceptron

on a

$$\text{soit } f(s) = \begin{cases} 1, \text{ si } s \geq \delta \\ 0, \text{ si } s < \delta \end{cases}, \text{ soit } f(s) = \begin{cases} 1, & \text{si } s > \delta \\ 0, & \text{si } -\delta \leq s \leq \delta \\ -1, & \text{si } s < \delta \end{cases} \tag{4.2}$$

Dans la suite on étudiera le cas bipolaire, ce qui n'est pas une limitation car on peut passer de l'un à l'autre cas par la transformation

$$bipolaire = 2 \times binaire - 1 \;\Rightarrow\; binaire = \frac{bipolaire + 1}{2} \tag{4.3}$$

Les vecteurs d'entrée appartiennent à deux classes distinctes et la tâche du perceptron est de fournir à la sortie la classe à laquelle appartient le vecteur d'entrée. La séparation entre les deux classes doit être linéaire, ce qui constitue la principale limitation du perceptron. La frontière entre les deux classes est donnée par l'hyperplan

$$\mathbf{w}^\top \boldsymbol{\xi} - \vartheta = 0 \tag{4.4}$$

Dans la suite on incorporera le biais dans le poids et on aura $\mathbf{w} = [\vartheta, w_1, \ldots, w_n]^\top$ et nous allons introduire une entrée supplémentaire constante et égale à -1 et on aura $\boldsymbol{\xi} = [-1, \xi_1, \ldots, \xi_n]$.

Pour s'adapter aux notations utilisées au chapitre précédent on supposera qu'il y ait une couche d'entrée $\mathbf{x}^0 = \mathbf{x}$ telle que $\mathbf{x} = \boldsymbol{\xi}$.

L'algorithme du perceptron est le suivant :

Variables et paramètres

- p entrées-sorties différentes $\left(\boldsymbol{\xi}_\mu, \psi_\mu\right)$, $\mu = 1, \ldots, p$, avec $\boldsymbol{\xi}_\mu = \left[-1, \xi_{1\mu}, \ldots, \xi_{n\mu}\right] \in \mathbb{R}^{n+1}$. On prendra $\mathbf{x}_\mu = \boldsymbol{\xi}_\mu$.
- Un vecteur des poids $\mathbf{w} = [\vartheta, w_1, \ldots, w_n]^\top = [w_0, w_1, \ldots, w_n]^\top \in \mathbb{R}^{n+1}$.
- La sortie calculée y_μ pour chaque entrée, avec $y_\mu \in \{0,1\}$ ou $y_\mu \in \{-1,1\}$.
- Un paramètre $\eta \in]0,1]$, appelé taux d'apprentissage dont le rôle est d'accélérer la convergence.

Étape 1 : Initialisation

On remplit au hasard le vecteur $\mathbf{w}$ avec des valeurs petites (habituellement entre -1 et $+1$, voire entre -0.5 et $+0.5$). On peut aussi prendre $\mathbf{w} = \mathbf{0}$.

Étape 2 : Apprentissage

On traite toutes les entrées selon un ordre aléatoire.

1. On choisit au hasard une entrée $\boldsymbol{\xi}_\mu$ parmi celles qui n'ont pas encore été choisies.
2. On calcule l'état du neurone de la sortie $s = \mathbf{w}^\top \boldsymbol{\xi}_\mu$
3. On calcule la valeur du neurone de la sortie à l'aide de la fonction d'activation $f : y_\mu = f(s)$ où f est donnée par les relations (4.2).
4. On adapte le vecteur des poids

$$\mathbf{w} = \mathbf{w} + \eta \cdot \left(\psi_\mu - y_\mu\right) \cdot \boldsymbol{\xi}_\mu \tag{4.5}$$

Notons que le facteur $\frac{1}{2}$ est utilisé seulement dans le cas bipolaire.

Étape 3 : Test de convergence

S'il y a eu convergence, c'est-à-dire si le vecteur des poids $\mathbf{w}$ est resté inchangé ce qui signifie que $y_\mu = \psi_\mu$; $\forall \mu$, alors l'algorithme se termine et le modèle du problème est resumé par ce vecteur des poids.

Sinon, on revient à l'étape 2.

Dans la formule de l'adaptation des poids la quantité $\left(\psi_\mu - y_\mu\right)$ est un signal d'erreur. Cette quantité est ponderée par le taux d'apprentissage η. Si on veut privilégier les résultats des itérations précedentes, c'est-à-dire faire en sorte que le poids soit une sorte de moyenne des poids déjà calculés, alors on prendra un η faible. Ceci aura comme conséquence d'avoir des estimations des poids stables. Si au contraire on voudrait avoir une adaptation rapide aux changements des vecteurs d'entrée, on prendra un η grand.

4.1.1 Convergence du perceptron

Rosenblatt a démontré en 1962 que cet algorithme converge, à condition que les classes soient linéairement séparables. Plus précisement, nous avons le

THÉORÈME 4.1.1 *Si les sous-ensembles X_1, X_2 des vecteurs d'apprentissage sont linéairement séparables, alors l'algorithme du peceptron appliqué à ces deux sous-ensembles converge après un nombre fini d'itérations.*

Il faut bien réaliser que si les deux classes ne sont pas linéairement séparables, l'algorithme de perceptron ne convergera pas. Il est donc nécessaire avant d'appliquer le perceptron à un ensemble de données, d'être sûr de leur séparabilité linéaire. On pourrait, à la place du perceptron, appliquer l'algorithme de Ho-Kashyap (cf. [6]) qui s'applique à des données séparables linéairement mais il indique qu'il échoue si les données ne sont pas séparables linéairement. Nous présentons brièvement cet algorithme, qui n'est pas une méthode du type RNA mais qui pourrait être utilisé pour tester le type de séparabilité des données.

Variables et paramètres

- p entrées-sorties différentes $\left(\boldsymbol{\xi}_\mu, \psi_\mu\right)$, $\mu = 1, \ldots, p$, avec $\boldsymbol{\xi}_\mu = \left[\psi_\mu, \xi_{1\mu}, \ldots, \xi_{n\mu}\right] \in \mathbb{R}^{n+1}$ et $\psi_\mu = +1$ ou -1. On prendra $\mathbf{x}_\mu = \boldsymbol{\xi}_\mu$ et on formera la matrice $\mathbf{X} = \left[\mathbf{x}_1^\top, \ldots, \mathbf{x}_p^\top\right] \in \mathbb{R}^{p\times(n+1)}$
- Un vecteur $\mathbf{b} \in \mathbb{R}^p$
- Un paramètre $\alpha \in]0,2[$

Étape 1 : Initialisation

1. Initialisation du vecteur $\mathbf{b}$
2. Initialisation du vecteur des poids

$$\mathbf{w}(0) = \left(\mathbf{X}^\top \mathbf{X}\right) \mathbf{X}^\top \mathbf{b}(0) \in \mathbb{R}^{n+1} \tag{4.6}$$

3. Initialisation du vecteur de l'erreur

$$\mathbf{e}(0) = \mathbf{X}\mathbf{w}(\mathbf{0}) - \mathbf{b}(0) \in \mathbb{R}^p \tag{4.7}$$

Étape 2 : Apprentissage

1. Mise à jour du vecteur des poids

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha \left(\mathbf{X}^\top \mathbf{X}\right) \mathbf{X}^\top \mid \mathbf{e}(k) \mid \tag{4.8}$$

2. Mise à jour du vecteur $\mathbf{b}$

$$\mathbf{b}(k+1) = \mathbf{b}(k) + (\mathbf{e}(k) + \mid \mathbf{e}(k) \mid) \tag{4.9}$$

3. Mise à jour du vecteur de l'erreur

$$\mathbf{e}(k+1) = \mathbf{X}\mathbf{w}(\mathbf{k}+\mathbf{1}) - \mathbf{b}(k+1) \tag{4.10}$$

Étape 3 : Test de convergence

La convergence est assurée dans un nombre fini d'itérations si $\mathbf{e}(k) = \mathbf{0}$.

Si $\mathbf{X}^\top \mid \mathbf{e}(k) \mid = \mathbf{0}$ avec $\mathbf{e}(k) \neq \mathbf{0}$, alors les données ne sont pas linéairement séparables.

Nous pouvons aussi démontrer un résultat inverse, à savoir si les données ne sont pas linéairement séparables, alors

– soit nous avons $\mathbf{X}^\top \mid \mathbf{e}(k) \mid = \mathbf{0}$ avec $\mathbf{e}(k) \neq \mathbf{0}$;
– soit $\mathbf{X}^\top \mid \mathbf{e}(k) \mid \longrightarrow 0$ avec $\mathbf{e}(k) \longrightarrow \mathbf{e}(\infty) \neq \mathbf{0}$ si $k \to \infty$.

Remarquons que l'algorithme de Ho-Kashyap converge plus vite que le perceptron([1]).

Une autre manière d'aborder le problème de la séparation linéaire est donnée par l'algorithme « pocket » présenté en 1990 par S. I. Gallant dans le cadre de l'utilisation des RNA par les systèmes experts. Suivant cet algorithme, à chaque itération on évalue le nombre d'exemples mal classés. Si ce nombre est inférieur au nombre d'exemples mal classés par les itérations précédentes, on garde le vecteur des poids (on le met « in the pocket »). Si après un nombre d'itérations fixé d'avance, on n'a pas convergé, on prend comme solution le vecteur des poids du « pocket » qui est le vecteur qui donne le nombre minimal d'exemples mal classés.

4.2 Qualité de la solution du perceptron

Considérons deux sous-ensembles des données linéairement séparables. L'hyperplaan séparateur est donné par l'équation (4.4). Si on considère le cas où le biais fait partie du vecteur des poids, cette relation s'écrit

$$\mathbf{w}^\top \mathbf{x} = \mathbf{0} \tag{4.11}$$

Étant donné un vecteur $\mathbf{x}$, sa distance de l'hyperplan est donnée par

$$d_{\mathbf{w}}(\mathbf{x}) = \frac{\mathbf{w}^\top \mathbf{x}}{\|\mathbf{w}\|} \tag{4.12}$$

avec $\|\mathbf{w}\| = \sqrt{\mathbf{w}^\top \mathbf{w}}$ la norme euclidienne.

La distance nous permet de définir la marge.

DÉFINITION 4.2.1 *La marge $\kappa(\mathbf{w})$ d'une solution $\mathbf{w}$ d'un perceptron appliqué sur l'ensemble de p exemples ξ_μ est la distance minimale des ces exemples, augmentés du terme constant, de l'hyperplan :*

$$\kappa(\mathbf{w}) = \min_{x_1 \cdots x_p} \frac{\mathbf{w}^\top \mathbf{x}_i}{\|\mathbf{w}\|} \tag{4.13}$$

La région d'epaisseur $2\kappa(\mathbf{w})$, centrée sur l'hyperplan ne contient aucun exemple.

Pour un ensemble d'exemples linéairement séparés en deux classes, il y a une infinité des plans séparateurs. La définition suivante introduit le perceptron de marge maximale.

DÉFINITION 4.2.2 *Le perceptron de marge maximale ou perceptron de stabilité optimale est le perceptron qui fournit la solution $\mathbf{w}$ de marge maximale, c'est-à-dire*

$$\kappa^\circ = \max_{\mathbf{w}} \kappa(\mathbf{w}) \tag{4.14}$$

Nous n'avons pas de méthode qui permet de s'assurer qu'un hyperplan séparateur fourni par l'algorithme du perceptron est de marge maximale. Néanmoins un plan séparateur avec une

1. Pour une comparaison détaillée de cet algorithme avec d'autres méthodes, se reporter à [7]

marge importante assure la stabilité de la solution dans le cas où il y des exemples perturbés avec du bruit. Notons aussi que si nous utilisons une fonction d'activation avec seuil δ, et s'il y a convergence, la marge de la solution est δ. Mais nous sommes pas en mesure de calculer la valeur de δ a priori.

4.3 Paramètres stabilisateurs et algorithme minover

Nous donnons dans la suite la définition des paramètres stabilisateurs pour un exemple.

DÉFINITION 4.3.1 *Soit un perceptron qui sur un ensemble d'exemples séparables a détérminé un vecteur des poids* $\mathbf{w}$. *Considérons un exemple d'entrée-sortie* $\left(\boldsymbol{\xi}_\mu, \psi_\mu\right)$. *On définit le paramètre stabilisateur de cet exemple par la relation*

$$\rho_{\mathbf{w}}\left(\boldsymbol{\xi}\right) = \psi_\mu \mathbf{w}^\top \boldsymbol{\xi}_\mu \tag{4.15}$$

Si à la place de la sortie désirée ψ_μ , *on utilise la sortie calculée* y_μ, *on a le paramètre stabilisateur calculé*

$$r_{\mathbf{w}}\left(\boldsymbol{\xi}\right) = y_\mu \mathbf{w}^\top \boldsymbol{\xi}_\mu \tag{4.16}$$

On peut facilement voir que si la sortie du perceptron y_μ est correcte, alors $r_\mu\left(\mathbf{w}\right) > 0$. Nous avons aussi

DÉFINITION 4.3.2 *Pour un exemple* $\left(\boldsymbol{\xi}_\mu, \psi_\mu\right)$ *donné, nous définissons sa stabilité à l'aide de la formule*

$$c_{\mathbf{w}}\left(\boldsymbol{\xi}\right) = \frac{r_{\mathbf{w}}\left(\boldsymbol{\xi}\right)}{\|\mathbf{w}\|} \tag{4.17}$$

En comparant avec (4.12), on a que la stabilité d'un exemple est sa distance de l'hyperplan séparateur :

$$c_{\mathbf{w}}\left(\boldsymbol{\xi}\right) = d_{\mathbf{w}}\left(\boldsymbol{\xi}\right) \tag{4.18}$$

L'exemple qui a la plus petite stabilité – ou, de façon équivalente, le plus petit paramètre stabilisateur – est l'exemple le moins stable, en ce sens qu'une petite modification de ses composantes peut provoquer par calcul du perceptron son changement de classe.

Cette observation est à la base d'une variante du perceptron, l'algorithme minover proposé par Krauth et Mézard en 1987. Cet algorithme modifie l'étape 2 de l'algorithme du perceptron. Au lieu de choisir un exemple au hasard, on choisira l'entrée $\boldsymbol{\xi}_{\mu^\circ}$ qui a la plus petite stabilité (ou le plus petit paramètre stabilisateur). Le reste de l'algorithme est identique. Cet algorithme converge plus rapidement que le perceptron. Ceci est dû au fait que chaque fois pour appliquer l'apprentissage on choisit la plus mauvaise des situations, c'est-à-dire l'exemple le moins stable.

4.4 Moindres carrés

Il s'agit d'une variante du perceptron dans laquelle la valeur du neurone de sortie est son état, c'est-à-dire

$$y = s = \sum_{i=1}^{n} w_i \xi_i \tag{4.19}$$

Nous n'avons donc pas de fonction d'activation et non plus de biais. La sortie y ici peut être considéré comme un filtre spatial des entrées.
L'adaptation des poids se fera sur la base de l'erreur

$$\delta_\mu = \psi_\mu - y_\mu ; \mu = 1, \ldots, p \tag{4.20}$$

La tâche du RNA est de minimiser l'erreur moyenne quadratique

$$J(\mathbf{w}) = \frac{1}{2} E\left(\delta_\mu^2\right) \tag{4.21}$$

La solution à ce problème de minimisation est donnée par la solution du système d'équations

$$\mathbf{R}\left(\boldsymbol{\xi}_\mu, \boldsymbol{\xi}_\mu^\top\right) \mathbf{w} = \mathbf{r}\left(\boldsymbol{\xi}_\mu, \psi_\mu\right) \tag{4.22}$$

où nous avons noté

- $\mathbf{R}\left(\boldsymbol{\xi}_\mu, \boldsymbol{\xi}_\mu^\top\right)$ la matrice de corrélation du vecteur d'entrée $\boldsymbol{\xi}_\mu$

$$\mathbf{R}\left(\boldsymbol{\xi}_\mu, \boldsymbol{\xi}_\mu^\top\right) = E\left(\boldsymbol{\xi}_\mu \cdot \boldsymbol{\xi}_\mu^\top\right) \tag{4.23}$$

- $\mathbf{r}\left(\boldsymbol{\xi}_\mu, \psi_\mu\right)$ le vecteur de la corrélation croisée entre l'entrée $\boldsymbol{\xi}_\mu$ et la sortie correspondante ψ_μ.

Ces équations sont les équations du filtre de Wiener-Hopf([2]). Leur solution fournit le vecteur optimal des poids $\mathbf{w}$. Pour obtenir cette solution il faut inverser la matrice de corrélation. Comme le vecteur des poids doit être adapté pour chaque entrée, cette approche devient lourde. Pour éviter d'inverser la matrice, on utilisera l'algorithme de descente du gradient. On obtient ainsi pour l'adaptation des poids la formule

$$\mathbf{w} = \mathbf{w} + \eta \cdot \left(\mathbf{r}\left(\boldsymbol{\xi}_\mu, \psi_\mu\right) - \mathbf{R}\left(\boldsymbol{\xi}_\mu, \boldsymbol{\xi}_\mu^\top\right) \cdot \mathbf{w}\right) \tag{4.24}$$

qui s'écrit aussi

$$\mathbf{w} = \eta \mathbf{r}\left(\boldsymbol{\xi}_\mu, \psi_\mu\right) + \left(\mathbf{I} - \eta \mathbf{R}\left(\boldsymbol{\xi}_\mu, \boldsymbol{\xi}_\mu^\top\right)\right) \mathbf{w} \tag{4.25}$$

Néanmoins même cette formule allegée contient encore des calculs qui sont coûteux en temps. Il s'agit surtout des calculs des corrélations. On les remplacera par des estimations des corrélations, qui ont l'avantage de pouvoir être traitées de manière réccurente. Ainsi à la place de la matrice de corrélation $\mathbf{R}\left(\boldsymbol{\xi}_\mu, \boldsymbol{\xi}_\mu^\top\right) = \left[E\left(\xi_{i\mu} \xi_{j\mu}\right)\right]$, on utilise son estimation

$$\widehat{\mathbf{R}}\left(\boldsymbol{\xi}_\mu, \boldsymbol{\xi}_\mu^\top\right) = \left[\xi_{i\mu} \xi_{j\mu}\right] \tag{4.26}$$

2. Pour une présentation détaillée du filtre de Wiener-Hopf voir, par exemple, S. Haykin : *Adaptive filter theory*, Prentice-Hall, 1986, pp.100-118.

et à la place du vecteur de la corrélation croisée $\mathbf{r}\left(\boldsymbol{\xi}_\mu, \psi_\mu\right) = \left[E\left(\psi_\mu \xi_{i\mu}\right)\right]$, on utilise son estimation

$$\widehat{\mathbf{r}}\left(\boldsymbol{\xi}_\mu, \psi_\mu\right) = \left[\psi_\mu \xi_{i\mu}\right] \tag{4.27}$$

Dans ce cas l'estimation des poids s'écrit

$$\widehat{\mathbf{w}} = \widehat{\mathbf{w}} + \eta \cdot \left(\psi_\mu - y_\mu\right) \cdot \boldsymbol{\xi} \tag{4.28}$$

qui est la méthode des moindres carrés.

L'algorithme est le suivant.

Variables et paramètres

- p entrées-sorties différentes $\left(\boldsymbol{\xi}_\mu, \psi_\mu\right)$, $\mu = 1, \ldots, p$, avec $\boldsymbol{\xi}_\mu = \left[\xi_{1\mu}, \ldots, \xi_{n\mu}\right] \in \mathbb{R}^n$ On prendra $\mathbf{x}_\mu = \boldsymbol{\xi}_\mu$.
- Un vecteur des poids $\mathbf{w} = [w_1, \ldots, w_n]^\top = [w_1, \ldots, w_n]^\top \in \mathbb{R}^n$.
- La sortie calculée y_μ pour chaque entrée, avec $y_\mu \in \{0, 1\}$ ou $y_\mu \in \{-1, 1\}$.
- Un paramètre $\eta \in]0, 1]$, appelé taux d'apprentissage dont le rôle est d'accélérer la convergence.

Étape 1 : Initialisation

On remplit au hasard le vecteur $\mathbf{w}$ avec des valeurs petites (habituellement entre -1 et $+1$, voire entre -0.5 et $+0.5$). On peut aussi prendre $\mathbf{w} = \mathbf{0}$.

Étape 2 : Apprentissage

On traite toutes les entrées selon un ordre aléatoire.

1. On choisit au hasard une entrée $\boldsymbol{\xi}_\mu$ parmi celles qui n'ont pas encore été choisies.
2. On calcule la valeur du neurone de la sortie : $y_\mu = \mathbf{w}^\top \boldsymbol{\xi}_\mu$.
3. On calcule l'erreur entre sortie calculée et sortie désirée
$$\delta_\mu = \psi_\mu - y_\mu \tag{4.29}$$
4. Adaptation du vecteur des poids
$$\widehat{\mathbf{w}} = \widehat{\mathbf{w}} + \eta \delta_\mu \boldsymbol{\xi}_\mu \tag{4.30}$$

Étape 3 : Test de convergence

S'il y a eu convergence, c'est-à-dire si le vecteur des poids $\mathbf{w}$ est resté inchangé, alors l'algorithme se termine et le modèle du problème est resumé par ce vecteur des poids.

Sinon, on revient à l'étape 2.

4.5 Adaline

Widrow et Hoff en 1960 on reprit l'algorithme des moindres carrés dans lequel ils ont ajouté comme fonction d'activation la fonction signe et aussi un seuil ϑ. Ils ont appelé leur algorithme Adaline (ADAptive LINear Element).

De cette façon ils ont modifié la quantité sur laquelle porte l'adaptation des poids. En effet comme la sortie calculée et la sortie désirée peuvent prendre comme valeurs $-1, 0, +1$, l'erreur δ aura comme valeurs les entiers entre -2 et 2. Donc tout se passe comme si Adaline minimisait le nombre moyen d'erreurs.

L'algorithme Adaline diffère du précédent à l'étape 2 qui devient

Étape 2 : Apprentissage

On traite toutes les entrées selon un ordre aléatoire.

1. On choisit au hasard une entrée $\boldsymbol{\xi}_\mu$ parmi celles qui n'ont pas encore été choisies.
2. On calcule la valeur du neurone de la sortie : $y_\mu = f\left(\mathbf{w}^\top \boldsymbol{\xi}_\mu\right)$.
3. On calcule l'erreur entre sortie calculée et sortie désirée
$$\delta_\mu = \psi_\mu - y_\mu \tag{4.31}$$
4. Adaptation du vecteur des poids
$$\widehat{\mathbf{w}} = \widehat{\mathbf{w}} + \eta \delta_\mu \boldsymbol{\xi}_\mu \tag{4.32}$$

Notons aussi que parfois la convergence pourrait s'accélérer si on divise l'adaptation de poids par la norme du vecteur $\boldsymbol{\xi}_\mu$, c-à-d.

$$\widehat{\mathbf{w}} = \widehat{\mathbf{w}} + \eta \delta_\mu \frac{\boldsymbol{\xi}_\mu}{\|\boldsymbol{\xi}_\mu\|}$$

4.6 Exercices

EXERCICE 4.1 *On construit un perceptron pour la porte logique* `AND` *avec les caractéristiques suivantes :*

- *les entrées sont binaires et les sorties bipolaires ;*
- *le biais* $\theta = 1$ *;*
- *le taux d'apprentissage est* $\eta = 1$ *;*
- *le seuil* $\delta = 0.2$
- *le vecteur es poids initaux* $\mathbf{w} = [-3, 2, 3]^\top$ *où la première composante correspond au biais.*

1. *Appliquer l'algorithme du perceptron et tracer pour les différents vecteurs des poids, les droites séparatrices.*
2. *Donner le plan séparateur de la solution.*
3. *Calculer pour chaque exemple sa distance au plan séparateur, son paramètre stabilisateur et sa stabilité.*

EXERCICE 4.2 *Montrer, par des méthodes algébriques, que sur l'exemple précédent l'utilisation d'un biais non nul est nécessaire pour obtenir une solution.*

EXERCICE 4.3 *Étudier, sur l'exemple précédent, l'influence du taux d'apprentissage* η *sur la convergence. En particulier examiner ce qui se passe si on prend* $\eta = 0.5$ *avec vecteur des poids* $\mathbf{w} = [-3, 2, 3]^\top$.

EXERCICE 4.4 *Supposons que la fonction d'activation est*

$$f(s) = \begin{cases} 1, & si \ \ s \geq 0 \\ -1, & si \ \ s < 0 \end{cases}$$

Appliquer l'algorithme du perceptron à l'exemple précédent avec cette fonction d'activation.

EXERCICE 4.5 *Même question si*

$$f(s) = \begin{cases} 1, & si \ \ s \geq 0 \\ 0, & si \ \ s < 0 \end{cases}$$

EXERCICE 4.6 *Appliquer l'algorithme Adaline à l'exemple de l'exercice 1, avec fonction d'activation*

$$f(s) = \begin{cases} 1, & si \ \ s \geq 0 \\ -1, & si \ \ s < 0 \end{cases}$$

et vecteurs des poids initiaux $\mathbf{w} = [-5, 0, 1]^\top$.

EXERCICE 4.7 *Utiliser l'algorithme du perceptron pour des réseaux à une ou deux entrées qui simulent le fonctionnement d'une porte*

1. AND logique
2. OR logique
3. Complément logique
4. NAND logique
5. NOR logique

EXERCICE 4.8 *Montrer par une méthode algébrique que le perceptron simple ne peut pas simuler une porte OU exclusif (XOR).*

EXERCICE 4.9 *Soient les exemples*

$$\Xi = \{([1,1],1),([1,-1],1),([0,1],1),([-1,-1],-1),([-1,1],-1),([0,1],1)\} \tag{4.33}$$

1. *Vérifier par une représentation graphique que les deux sous-ensembles sont linéairement séparables.*
2. *En prenant comme vecteur des poids initial* $\mathbf{w} = [1,1,1]^\top$ *et comme biais* $\vartheta = 1$*, appliquer l'algorithe du perceptron jusqu'à la convergence. Les exemples seront traités dans l'ordre lors de l'étape 2 de l'algorithme.*
3. *Calculer pour chaque exemple sa distance du plan séparateur, son paramètre stabilisateur et sa stabilité.*
4. *Donner pour la solution la marge* $\kappa(\mathbf{w})$.
5. *Trouver l'exemple le moins stable. Commentaires sur sa robustesse.*
6. *Appliquer l'algorithme minover et comparer avec l'algorithme du perceptron.*

4.7 Références

Pour la redaction de ce chapitre ont été utilisés les livres et articles ci-dessous :

[1] N. K. BOSE, P. LIANG : *Neural network fundamentals*, McGraw-Hill, 1996

[2] G. DREYFUS ET AL. : *Réseaux de neurones*, Eyrolles,

[3] R. M. GOLDEN : *Mathematical methods for neural network analysis and design*, MIT Press, 1996

[4] S. HAYKIN : *Neural networks*, IEEE Press, 1994

[5] J. HERTZ, A. KROGH, R. G. PALMER : *Introduction to the theory of neural computation*, Addiso-Wesley, 1991

[6] E. HO, R. L. KASHYAP : An algorithm for linear inequalites and its application, IEEE Trans. Electronic Computers, 14 (1965), 161-167

[7] F. LAUER, G. BLOCH : Ho-Kashyap Classifier with Early Stopping for Regularization, Pattern Recognition Letters, 27, 9, (2006), 1037-1044

[8] P. PERETTO : *An introduction to the modeling of neural networks*, Cambridge Un. Press, 1992

5

RNA NON RÉCURRENTS AVEC COUCHE(S) CACHÉE(S)

Les couches cachées dans un RNA ajoutent de la non-linéarité supplémentaire. En effet sur un RNA sans couches cachées, il y a une non-linéarité uniquement sur la sortie. Avec l'addition des couches cachées, chaque neurone d'une couche agit comme un dispositif non-linéaire et cette non-linéarité est propagée sur les couches suivantes.

Si on utilise un tel réseau pour approcher une fonction, il y a un fondement théorique et une conjecture. Le fondement théorique est le théorème suivant démontré par Kolmogorov en 1957 :

THÉORÈME 5.0.1 *Si h est une fonction continue définie sur le cube à n dimensions $I_n = [0,1]^n$, alors elle peut s'écrire sous la forme*

$$h(x_1,\ldots,x_n) = \sum_{i=1}^{2n+1} f_i^{(2)} \left(\sum_{j=1}^{n} f_{ij}^{(1)}(x_j) \right) \tag{5.1}$$

où $f_{ij}^{(1)}$ et $f_i^{(2)}$ sont des fonctions continues à une variable et $f_{ij}^{(1)}$ est monotone croissante, ne dépendante de h, c'est-à-dire telle que

$$\left[f_{ij}^{(1)}(x) - f_{ij}^{(1)}(x') \right] \cdot (x - x') > 0, \text{ si } x \neq x' \tag{5.2}$$

Cette formule peut être réalisée par un RNA avec une couche cachée et une sortie avec un seul neurone, dans lequel les poids vers la couche cachée $w_{ij}^{(1)}$ et vers la couche de sortie $w_i^{(2)}$seraient les fonctions $f_{ij}^{(1)}$ et $f_i^{(2)}$ respectivement.

La conjecture de Kolmogorov est le même théorème mais à la place des fonctions $f_{ij}^{(1)}(x_j)$ nous avons des combinaisons linéaires des x_j et sans préciser les limites de la somme qui peuvent ainsi être infinies. Cette conjecture nous renvoie donc à un RNA avec une seule couche cachée.

5.1 Rappel des notions et des définitions

Un RNA multi-couches peut avoir un nombre quelconque des couches cachées. En général on se limite à une, deux ou trois couches cachées. La figure suivante représente un RNA avec deux couches cachées.

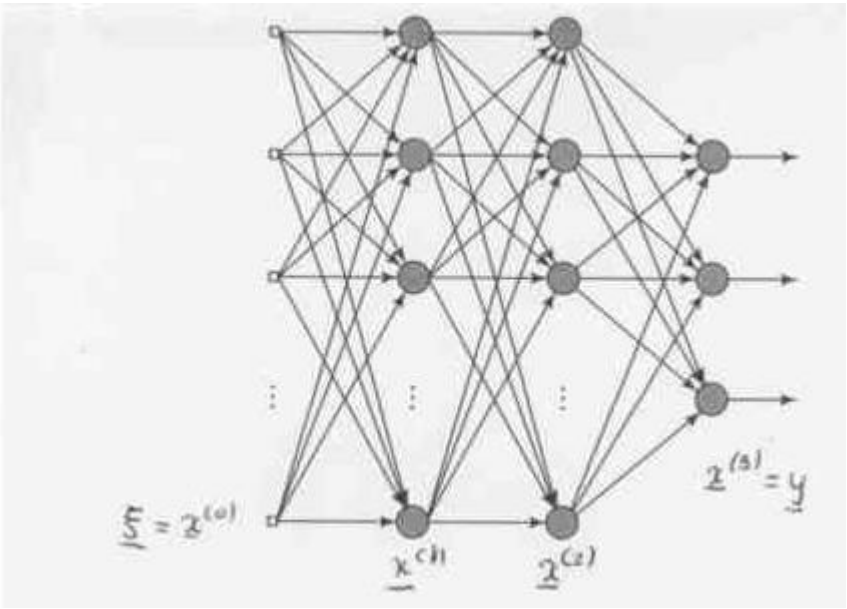

FIGURE 5.1 – RNA avec deux couches cachées

La notation utilisée pour les RNA multi-couches est celle du chapitre 3, que nous rappelons brièvement.

- Entrées-sorties $(\boldsymbol{\xi}_\mu, \boldsymbol{\psi}_\mu)$, avec $\boldsymbol{\xi}_\mu \in \mathbb{R}^n$, $\boldsymbol{\psi}_\mu \in \mathbb{R}^m$, $\mu = 1, \ldots, p$. $\boldsymbol{\xi}_\mu$ représente une entrée dans un espace à n dimensions, $\boldsymbol{\psi}_\mu$ la sortie correspondante dans un espace à m dimensions et on a p couples d'entrées-sorties différents pour la population d'apprentissage.
- La première couche qui introduit dans le réseau les entrées est composée du vecteur des neurones $\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ \boldsymbol{\xi}_\mu \end{bmatrix} = \left[1, \xi_{1\mu}, \ldots, \xi_{n\mu}\right]^\top = \left[x_0^{(0)}, x_1^{(0)}, \ldots, x_n^{(0)}\right] \in \mathbb{R}^{n+1}$. La composante constante $x_0^{(0)} = 1$ correspond au biais qui sera introduit dans le vecteur des poids.
- La première couche cachée est donnée par le vecteur des neurones $\mathbf{x}^{(1)} = \left[x_0^{(1)}, x_1^{(1)}, \ldots, x_{n_1}^{(1)}\right] \in \mathbb{R}^{n_1+1}$, où la première composante $x_0^{(1)}$ est le biais $\vartheta^{(1)}$ de la première couche cachée.
- Pour calculer les valeurs des neurones de la première couche cachée on utilise les entrées $\mathbf{x}^{(0)}$ et la matrice des poids des liaisons des neurones de la couche d'entrée avec

les neurones de la première couche cachée. Ainsi le poids entre les neurones $x_i^{(0)}$ et $x_j^{(1)}$ est égal à $w_{ji}^{(1)} \in \mathbb{R}$. On a donc la matrice des poids

$$\mathbf{W}^{(1)} = \begin{bmatrix} w_{00}^{(1)} & w_{01}^{(1)} & \cdots & w_{0n}^{(1)} \\ w_{10}^{(1)} & w_{11}^{(1)} & \cdots & w_{1n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_1 0}^{(1)} & w_{n_1 1}^{(1)} & \cdots & w_{n_1 n}^{(1)} \end{bmatrix} \in \mathbb{R}^{(n_1+1)\times(n+1)} \tag{5.3}$$

Dans la mesure où il n'y a pas de liaison entre les biais de deux couches consécutives, ni entre les biais d'une couche et les neurones de la couche précédente, on en coclut que la première ligne de la matrice $\mathbf{W}^{(1)}$ est nulle. En fait cette première ligne existe pour des raisons de compatibilité de la multiplication avec le vecteur $\mathbf{x}^{(0)}$.

Le calcul de $\mathbf{x}^{(1)}$ se fait en deux étapes :

– Calcul de l'état des neurones de la première couche cachée :

$$\mathbf{s}^{(1)} = \mathbf{W}^{(1)}\mathbf{x}^{(0)} \tag{5.4}$$

– Calcul de la sortie des neurones

$$\mathbf{x}^{(1)} = \mathbf{f}^{(1)}\left(\mathbf{s}^{(1)}\right) \tag{5.5}$$

où $\mathbf{f}^{(1)}$ est une fonction vectorielle.

Initialement Rosemblat avait envisagé que toutes les composantes sont soit les fonctions seuil, soit les fonctions signe. Ensuite on a utilisé des fonctions sigmoïdes, à savoir

– soit la fonction logistique

$$f(t) = \frac{1}{1+e^{-t}} \tag{5.6}$$

– soit la tangente hyperbolique

$$f(t) = tgh(t) = \frac{1-e^{-t}}{1+e^{-t}} \quad (^1) \tag{5.7}$$

– Pour le calcul des valeurs des neurones $\mathbf{x}^{(\ell)}$ d'une couche $\ell = 1, \ldots, L$, on utilise les valeurs des neurones $\mathbf{x}^{(\ell-1)}$ de la couche précédente, ainsi que la matrice des poids des liaisons des neurones de deux couches $\mathbf{W}^{(\ell-1)}$, selon la formule

$$\mathbf{x}^{(\ell)} = \mathbf{f}^{(\ell)}\left(\mathbf{W}^{(\ell)}\mathbf{x}^{(\ell-1)}\right) = \mathbf{f}^{(\ell)}\left(\mathbf{s}^{(\ell)}\right) \tag{5.8}$$

avec

$$\mathbf{s}^{(\ell)} = \mathbf{W}^{(l)}\mathbf{x}^{(l-1)} \tag{5.9}$$

– Si $\ell = L$, nous avons la couche de sortie et dans ce cas le calcul nous fournit la sortie $\mathbf{y}$, à savoir

$$\mathbf{y}_\mu = \mathbf{f}^{(L)}\left(\mathbf{W}^{(L)}\mathbf{x}^{(L-1)}\right) = \mathbf{f}^{(L)}\left(\mathbf{s}^{(L)}\right) \tag{5.10}$$

– L'objectif du RNA est de minimiser l'erreur de sortie. Cette erreur peut s'exprimer comme suit :

$$\mathbf{e}_\mu = \boldsymbol{\psi}_\mu - \mathbf{y}_\mu \tag{5.11}$$

d'où on calcule la norme de l'erreur

$$E_\mu = \frac{1}{2} \left\| \mathbf{e}_\mu \right\|^2 \tag{5.12}$$

La minimisation porte sur la somme des erreurs pour tous les exemples :

$$E = \frac{1}{p} \sum_{\mu=1}^{p} E_\mu \tag{5.13}$$

Dans la suite du chapitre nous présentons le perceptron à une couche cachée. Nous continuerons au chapitre suivant avec l'étude de la rétropropagation comme technique d'optimisation des résultats d'un RNA multi-couche. Ce qui différencie ces deux types des réseaux est la méthode de la mise à jour des poids des neurones qui est différente selon le type.

5.2 Perceptron à une couche cachée

Le calcul des valeurs des neurones de la couche cachée et celle de la sortie du perceptron à une couche cachée est donné ci-dessus. Il reste à décrire la mise à jour des poids. On distingue deux cas :

1. Poids des liaisons couche cachée – couche de sortie. On calcule l'erreur de la sortie par rapport à la sortie désirée :

$$\mathbf{e} = \boldsymbol{\psi}_\mu - \mathbf{y} \tag{5.14}$$

La correction de la matrice des poids est donnée par la relation

$$\Delta \mathbf{W}^{(2)} = \eta \cdot \mathbf{e} \cdot \left(\mathbf{x}^{(1)} \right)^\top \tag{5.15}$$

d'où on obtient finalement pour l'adaptation des poids

$$\mathbf{W}^{(2)} = \mathbf{W}^{(2)} + \Delta \mathbf{W}^{(2)} \tag{5.16}$$

2. Poids des liaisons entrée – couche cachée. La correction de la matrice des poids est plus complexe. En effet on a

$$\Delta \mathbf{W}^{(1)} = \eta \left(\mathbf{W}^{(\mathbf{1})} \right)^\top \cdot \mathbf{e} \cdot \left(\mathbf{x}^{(\mathbf{0})} \right)^\top \tag{5.17}$$

L'adaptation des poids se fait selon al formule

$$\mathbf{W}^{(1)} = \mathbf{W}^{(1)} + \Delta \mathbf{W}^{(1)} \tag{5.18}$$

Il est évident que si on a plus d'une couche cachée, les calculs seront encore plus complexes. Il a plusieurs méthodes qui permettent de contourner cette difficulté. Nous présentons par la suite deux de ces méthodes : madaline et rétropropagation.

5.3 Madaline

Madaline vient de Many ADAptive LInear NEurons qui sont disposés sur un réseau multi-couches. En réalité il s'agit des plusieurs réseaux Adaline mis ensemble avec une structure en couches. Nous donnons l'algorithme pour un réseau avec une couche cachée et un neurone de sortie, la généralisation à plusieurs couches cachées étant évidente.

Variables et paramètres

- p entrées-sorties différentes $\left(\boldsymbol{\xi}_\mu, \psi_\mu\right)$, $\mu = 1, \ldots, p$, avec $\boldsymbol{\xi}_\mu = \left[\xi_{1\mu}, \ldots, \xi_{n\mu}\right] \in \mathbb{R}^n$ On prendra $\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ \boldsymbol{\xi}_\mu \end{bmatrix} \in \mathbb{R}^{n+1}$.
- La matrice des poids entre entrées et couche caché $\mathbf{W}^{(1)}$ et le vecteur des poids entre couche caché et sortie $\mathbf{w}^{(2)}$.
- La sortie calculée y_μ pour chaque entrée, avec $y_\mu \in \{0, 1\}$ ou $y_\mu \in \{-1, 1\}$.
- Un paramètre $\eta \in]0, 1]$, appelé taux d'apprentissage dont le rôle est d'accélérer la convergence.

Étape 1 : Initialisation

On remplit au hasard la matrice $\mathbf{W}^{(1)}$ et le vecteur $\mathbf{w}^{(2)}$ avec des valeurs petites (habituellement entre -1 et $+1$, voire entre -0.5 et $+0.5$).

Étape 2 : Apprentissage

On traite toutes les entrées selon un ordre aléatoire.

1. On choisit au hasard une entrée $\boldsymbol{\xi}_\mu$ parmi celles qui n'ont pas encore été choisies et on pose $\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ \boldsymbol{\xi}_\mu \end{bmatrix}$.
2. On calcule l'état des neurones de la couche cachée
$$\mathbf{s}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} \tag{5.19}$$
3. On calcule la valeur des neurones de la couche cachée
$$\mathbf{x}^{(1)} = \mathbf{f}^{(1)}\left(\mathbf{s}^{(1)}\right) \tag{5.20}$$
4. On calcule l'état du neurone de la sortie
$$\mathbf{s}^{(2)} = \left(\mathbf{w}^{(2)}\right)^\top \mathbf{x}^{(1)} \tag{5.21}$$
5. On calcule la valeur du neurone de la couche de la sortie :
$$y = \mathbf{x}^{(2)} = f^{(2)}\left(\mathbf{s}^{(2)}\right) \tag{5.22}$$
où $f^{(2)}$ est la fonction signe.
6. On calcule l'erreur entre sortie calculée et sortie désirée
$$e_\mu = \psi_\mu - y \tag{5.23}$$
7. Adaptation du vecteur des poids :

– Si $e = 0$, pas d'adaptation des poids.
Sinon
– Si $e = 1$, alors pour tout neurone j de la couche cachée dont la valeur $x_j^{(1)}$ est suffisamment proche de 0 (par exemple $x_j^{(1)} \in [-0.25, 0.25]$), adaptation des ses poids :

$$w_{ji} = w_{ji} + \eta \left(1 - x_j^{(1)}\right) x_i^{(0)} \; ; i = 0, \ldots, n \tag{5.24}$$

– Si $e = -1$, alors pour tout $x_j^{(1)} > 0$, on adapte les poids correspondants :

$$w_{ji} = w_{ji} + \eta \left(-1 - x_j^{(1)}\right) x_i^{(0)} \; ; i = 0, \ldots, n \tag{5.25}$$

Étape 3 : Test de convergence

S'il y a eu convergence, c'est-à-dire si les matrices des poids $\mathbf{W}^{(1)}$ et $\mathbf{W}^{(2)}$ sont restées inchangées, alors l'algorithme se termine et le modèle du problème est resumé par ces matrices des poids.

Sinon, on revient à l'étape 2.

5.4 Exercices

EXERCICE 5.1 *Construire un perceptron à une couche cachée qui simule le fonctionnement de* `XOR` *(ou exclusif) et calculer les poids.*

EXERCICE 5.2 *En utilisant JavaNNS construire un réseau qui simule le fonctionnement d'Adaline. Utiliser ce réseau pour une porte logique* `AND`*.*

EXERCICE 5.3 *En utilisant JavaNNS construire un réseau qui simule le fonctionnement de Madaline. Utiliser ce réseau pour une porte logique* `XOR`*. endexc*

5.5 Références

Pour la redaction de ce chapitre ont été utilisés les livres et articles ci-dessous :

[1] G. DREYFUS ET AL. *: Réseaux de neurones, Eyrolles,*

[2] L. FAUSETT *: Fundamentals of neural networks, Prentice Hall, 1994*

[3] S. HAYKIN *: Neural networks, IEEE Press, 1994*

Table des matières