Titre: Title:	Modification d'un algorithme d'apprentissage pour réseaux de neurones appliqués à la couverture de surface
Auteur: Author:	Réza Assadi
Date:	2003
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Assadi, R. (2003). Modification d'un algorithme d'apprentissage pour réseaux de neurones appliqués à la couverture de surface [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7214/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/7214/
Directeurs de recherche: Advisors:	Richard Labib, & Luc Adjengue
Programme: Program:	Non spécifié

UNIVERSITÉ DE MONTRÉAL

MODIFICATION D'UN ALGORITHME D'APPRENTISSAGE POUR RÉSEAUX DE NEURONES APPLIQUÉS À LA COUVERTURE DE SURFACE

RÉZA ASSADI

DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(MATHÉMATIQUES APPLIQUÉES)

NOVEMBRE 2003

Your file Votre référence
ISBN: 0-612-89174-7
Our file Notre référence
ISBN: 0-612-89174-7

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

MODIFICATION D'UN ALGORITHME D'APPRENTISSAGE POUR RÉSEAUX DE NEURONES APPLIQUÉS À LA COUVERTURE DE SURFACE

présenté par : ASSADI RÉZA

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

à été dûment accepté par le jury d'examen constitué de :

M. BRAULT Jean-Jules, Ph.D., président

M. LABIB Richard, Ph.D., membre et directeur de recherche

M. ADJENGUE Luc-Désiré, Ph.D., membre et codirecteur de recherche

M. ROUSSELLE Jean, Ph.D., membre

À mes parents, sans qui je ne serai pas arrivé jusqu'ici.

REMERCIEMENTS

Je tiens tout d'abord à remercier tous ceux qui ont travaillé de près ou de loin à la réalisation de ce document, plus précisément mon directeur de recherche, M. Richard Labib qui m'a apporté son aide et son soutien dans la compréhension de ce sujet. Je lui suis également très reconnaissant de m'avoir inculqué les qualités nécessaires pour éventuellement devenir un bon chercheur. Mais, également M. Marc Moore et M. Luc-Désiré Adjengue pour leur soutien financier au cours de mes études.

J'aimerais aussi exprimer ma reconnaissance à l'endroit de ma famille, ma mère, Marie-Françoise, mon père, Ébrahim et ma soeur Leila pour n'avoir jamais ménagé leurs efforts pour m'encourager et me soutenir dans mes périodes de détresse et pour avoir endurer mes moments de déraison. Je leur dois ma réussite. Trouvez dans ces quelques lignes ma très profonde gratitude.

Je ne saurais oublier mes amis Nada, Tania, Nabil, Ulysse, Parham et Jonathan. Je tiens également à remercier ma tante Anne-Hélène pour m'avoir aidé avec la rédaction de ce document, Karim mon frère d'arme dans cette aventure et plus particulièrement Romel et Susana pour leur encouragement, mais surtout de m'avoir ramené sur le bon chemin

lors de mes moments d'égarement. Je vous présente mes sincères remerciements. Enfin, j'adresse aussi des remerciements à tous ceux qui ont contribué à ma formation.

RÉSUMÉ

La couverture de surface avec un Perceptron Multicouches (Multi-Layer Perceptron MLP) est difficile à réaliser lorsque les objets couvrant la surface sont fixes. C'est-à-dire, si le problème était de couvrir une surface avec des cercles, il faudrait non seulement apprendre au réseau que la tâche est de couvrir une surface, mais également que celle-ci ne sera couverte qu'avec des cercles. Or pour simplifier l'apprentissage du MLP il suffit de minimiser l'information à apprendre en utilisant au maximum l'information connue. Il faut donc, pour le problème de couverture de surface avec des objets fixes, introduire au préalable l'information sur l'objet utilisé à l'intérieur du MLP.

Il faudra ainsi dans un premier temps étudier en détail l'effet de ce supplément d'information sur le MLP et l'algorithme d'apprentissage de rétro-propagation. Une modification majeure sera de restreindre certains poids à une valeur fixe durant le processus d'apprentissage, ce qui pourrait nuire à la capacité du MLP de trouver une solution optimale aux problèmes étudiés. Le chapitre 2 montrera qu'il est possible pour le MLP de trouver la solution optimale si seulement un seul poids est fixé. Dans le cas où il y en aurait plus d'un, il faudra rajouter des entrées et des poids et/ou des couches de façon à s'assurer que le MLP puisse trouver une solution optimale. Ces modifications au MLP amèneront également certaines modifications à l'algorithme d'apprentissage

Dans un deuxième temps, ces poids fixes et ces couches supplémentaires permettront à l'usager de mieux contrôler l'apprentissage du MLP de façon à trouver une solution particulière parmi toutes les solutions optimales possibles. Le chapitre 3 traitera du problème de couverture de surface, en particulier des problèmes de « covering » et de « packing », à l'aide du MLP modifié. Il y aura également quelques améliorations supplémentaires afin d'améliorer et/ou d'accélérer l'apprentissage.

ABSTRACT

Surface covering with a Multi-Layer Perceptron (MLP) is difficult to accomplish when the objects used to cover the surface are fixed. This implies that if the problem was to cover a surface using nothing but circles, we would have to not only teach the network that it has to cover a certain surface, but also that this surface has to be covered using only circles. Yet in order to simplify the learning phase of the MLP, one would only have to minimize the information to be learned by maximizing the use of the prior information. Consequently, for surface covering problems with fixed objects, we need only to add the prior information on the objects inside the MLP.

We will initially make a detail study of the effects this added information will have on the MLP and the Back-Propagation algorithm, its learning algorithm. One major modification will be to restrict certain weights to a fixed value during the learning process, which could hamper the Multi-Layer Perceptron's (MLP) capacity to find an optimal solution to the problem. Chapter 2 will show that for one fixed weight it is still possible for the MLP to find an optimal solution, but for more than one fixed weight, additional inputs and weights and/or additional layers are needed in order to insure the MLPs capacity to find an optimal solution. These modifications to the MLP will also entail modifications to the learning algorithm.

Furthermore, these fixed weights and added inputs will enable the user to better control the MLP in order to find the desired solution amongst all the possible optimal solutions. Chapter 3 will examine surface covering problems, specifically covering and packing problems, using the modified MLP. There will also be other improvements in order to improve and/or accelerate the learning phase.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

CHAPITRE 1 : INTRODUCTION

> L'Homme naît avec des sens et des facultés; mais il n'apporte avec lui en naissant aucune idée: son cerveau est une table rase qui n'a reçu aucune impression, mais qui est préparée pour en recevoir.
>
> Antoine-Laurent de Lavoisier

Le Grand Larousse Universel définit l'intelligence comme étant l'aptitude d'un être humain à s'adapter à une situation, à choisir des moyens d'action en fonction des circonstances. Or l'intelligence dépend en partie de deux éléments: le potentiel initial du cerveau et la qualité de l'apprentissage. La communauté scientifique a voulu alors recréer cette capacité qu'a le cerveau d'absorber, de comprendre, d'analyser, d'inférer et d'utiliser l'information autour de lui. Comme Dieu créa l'homme à son image, L'Homme, à son tour, veut créer la machine à son image. Il en est déjà à une forme primitive d'intelligence artificielle, le réseau de neurones.

L'outil principal de l'intelligence est le cerveau humain. Celui-ci peut ressembler à un ordinateur très complexe, qui fonctionne de façon non linéaire et parallèle. Non linéaire car il est nécessaire à l'apprentissage, et parallèle car il peut analyser plusieurs choses à

la fois. Or, le cerveau lui-même est constitué de ce qu'on appelle un réseau de neurones. Ce système semble être celui que la nature a choisi pour créer l'intelligence. Il semble donc naturel, et sage, d'imiter celui-ci lors de la création d'une intelligence artificielle. Suivant ce raisonnement, la tâche à accomplir revient à créer un réseau de neurones artificiel, celui-ci étant défini de la façon suivante ([5] Haykin, 1999) :

> « Un réseau de neurones est un processeur composé de plusieurs unités simples de traitement parallèlement distribuées, ayant une tendance naturelle pour le stockage et le rappel de connaissances expérimentales. Il ressemble au cerveau selon ces deux aspects :
>
> 1. La connaissance de l'environnement est acquise par le réseau à travers un processus d'apprentissage.
> 2. L'intensité des liens interneuronaux, appelé poids synaptiques, est utilisée pour stocker la connaissance acquise. ».

Le processus d'apprentissage, défini par l'entremise d'un algorithme itératif, celui-ci étant la répétition d'un ensemble de règles opératoires intervenant dans un calcul, est réalisé grâce à la modification des poids synaptiques, c'est-à-dire l'influence que chaque élément de l'information a sur le résultat.

Dans les années 40, Alan Turing [19] propose le concept de machine intelligente et McCulloch et Pitts [11] proposent une architecture de neurones, mais il faudra attendre le Perceptron de Rosenblatt [15] en 1958 pour avoir le premier algorithme

d'apprentissage avec preuve de convergence. Grâce à ces avancées une forme simple d'intelligence artificielle est née.

La figure 1.1 montre un Perceptron à n entrées ($x_1 \ldots x_n$), où les ω représentent les poids, c'est-à-dire l'influence que les entrées ont sur la sortie v. Ces poids seront très importants au cours de ce mémoire étant donné qu'ils seront les premiers à être modifiés et que les autres modifications reposeront sur celles-ci. θ représente le biais et φ représente la fonction d'activation, celle-ci différencie les entrées entre deux classes.

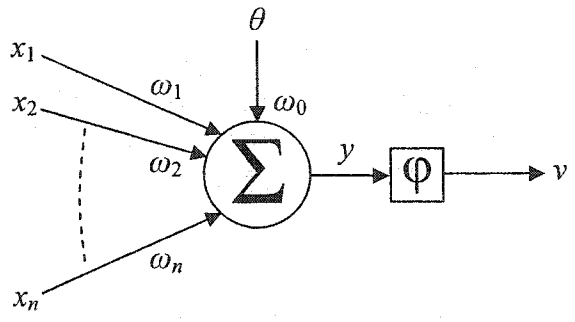


Figure 1.1: Perceptron à n entrées.

L'équation de sortie est :

$$v = \varphi\left(\sum_{i=0}^{n} \omega_i x_i\right) \tag{1.1}$$

Depuis, beaucoup de travail a été fait pour améliorer les capacités de cette nouvelle intelligence. Un Perceptron étant formé d'un seul neurone, il peut seulement différencier

2 classes linéairement séparables, c'est pourquoi le potentiel initial a dû être amélioré dans un premier temps. Cette évolution fut le réseau de neurones, c'est-à-dire plusieurs Perceptrons interconnectés. Par la suite, le nouveau réseau étant plus complexe, la qualité de l'apprentissage a dû être également améliorée. De ces améliorations est né le Perceptron Multicouches (Multi-Layer Perceptron MLP) avec pour algorithme d'apprentissage l'algorithme de rétro-propagation [16]. Le terme multicouches indique qu'un ou plusieurs neurones sont connectés aux entrées et que ceux-ci sont par la suite connectés à d'autres neurones et ainsi de suite jusqu'à la sortie; le nombre de couches correspond au nombre de niveaux entre les entrées et la sortie inclusivement. C'est-à-dire qu'un réseau à 4 couches comporte une couche d'entrée, deux couches cachées et une couche de sortie. Le MLP sera nécessaire plus tard lors de l'application du nouveau réseau de neurones qui sera développé au prochain chapitre. D'après Haykin [5], l'algorithme peut se diviser en cinq étapes :

1. Initialisation :

Cette étape consiste principalement à initialiser les poids. C'est-à-dire donner une valeur de départ, généralement aléatoire et limitée, à ceux-ci.

2. Présentation des vecteurs d'entraînement :

Cette étape consiste à présenter, dans un certain ordre, au réseau une série de vecteurs d'entraînements qu'on appellera « époques ».

Les prochaines étapes peuvent s'effectuer de deux façons différentes :

a. La première serait d'effectuer l'étape 3 pour tous les vecteurs d'entraînement avant de passer à l'étape 4, de cette façon on calculerait une erreur moyenne pour chaque époque.
b. La deuxième, celle qui sera employée lors des simulations au chapitre 3, serait d'effectuer les étapes 3 et 4 pour chaque vecteur.

Cette dernière à l'avantage d'être moins gourmande au niveau de la mémoire nécessaire lors des simulations, puisqu'on n'a pas à retenir toutes les sorties pour tous les vecteurs d'entraînements. Il est également moins probable de tomber dans un minimum local, puisque qu'elle est de nature stochastique.

3. Calcul en aval :

La sortie de la sommation du neurone j de la couche l sera :

$$y_j^{(l)}(t) = \sum_{i=0}^{n} \omega_{ji}^{(l)}(t) v_i^{(l-1)}(t) \tag{1.2}$$

Où $v_i^{(l-1)}(t)$ est la sortie du neurone i de la couche (l-1) à l'itération t. Pour i = 0, où $v_0^{(l-1)}(t) = +1$ et $v_j^{(0)}(t) = x_j(t)$, où x_j est le j-ième élément de $\boldsymbol{x(t)}$, le vecteur d'entrée.

Pour une fonction d'activation φ_j la sortie du neurone j de la couche l sera :

$$v_j^{(l)}(t) = \varphi_j\left(y_j(t)\right) \tag{1.3}$$

Et finalement puisque la sortie du réseau est $v_j^{(L)}(t)$ alors l'erreur sera donc :

$$e_j(t) = d_j(t) - v_j^{(L)}(t) \tag{1.4}$$

Où $d_j(t)$ est le j-ième élément de $d(t)$, le vecteur de sortie désiré.

4. Calcul en amont :

L'algorithme prend son nom de cette étape, puisqu'il consiste à rétro-propager l'erreur à travers le réseau. Le calcul consiste à évaluer le gradient local δ de chaque neurone de façon à pouvoir effectuer la descente du gradient :

$$\delta_j^{(l)}(t) = \begin{cases} e_j^{(L)}(t)\varphi_j'\left(y_j^{(L)}(t)\right) & \text{pour un neurone } j \text{ sur la couche } L \\ \varphi_j'\left(y_j^{(l)}(t)\right)\sum_k^n \delta_k^{(l+1)}(t)\omega_{kj}^{(l+1)}(t) & \text{pour un neurone } j \text{ sur la couche } l \end{cases} \quad (1.5)$$

où φ_j' représente la dérivée de φ par rapport à $\omega_{ji}^{(l)}$. Finalement on ajuste les poids par l'équation suivante :

$$\omega_{ji}^{(l)}(t+1) = \omega_{ji}^{(l)}(t) + \alpha\left(\omega_{ji}^{(l)}(t-1)\right) + \eta\delta_j^{(l)}(t)v_i^{(l-1)}(t) \quad (1.6)$$

où α est la constante du momentum, pour α =0 on élimine celui-ci car il n'est pas obligatoire, et η la constante d'apprentissage.

5. Itérations :

La dernière étape consiste à répéter les étapes 3 et 4 pour le même époque ou pour une nouvelle époque.

Encore une fois, ces étapes serviront plus tard lors de l'application des modifications qui seront apportées au MLP, c'est-à-dire les poids constants et le rajout d'entrées. Ceci n'est toutefois pas la première fois que le réseau MLP est modifié de façon à améliorer

un aspect ou un autre de l'apprentissage. Voici un aperçu des modifications qui ont déjà été effectuées au niveau des entrées.

Dans les années 80, Rumelhart et McClelland utilisent une somme de produits comme entrée à l'intérieur d'un réseau qu'ils appellent Sigma-Pi Networks [17] :

$$y = \sum_i \omega_{ji} \prod_k x_{ik} \tag{1.7}$$

Un autre cas, proposé par Lee et al. [9], utilise la matrice complète du produit des entrées.

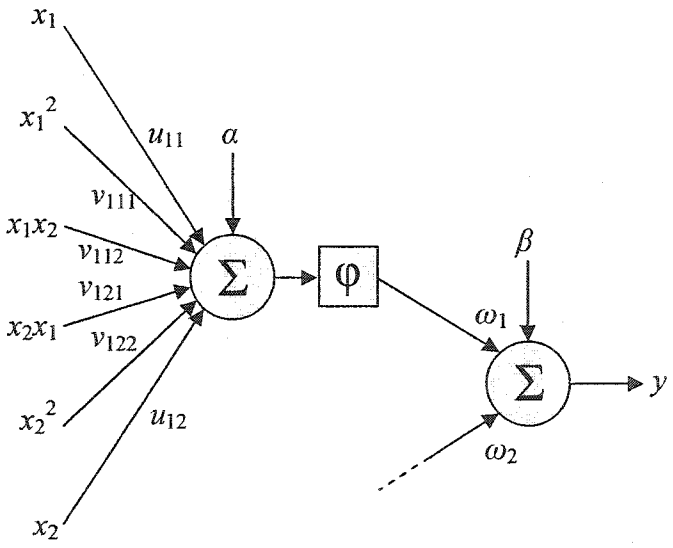


Figure 1.2: Représentation du réseau de Lee et al. La figure montre seulement le premier neurone de la première couche, la deuxième étant identique.

On a donc à la sortie l'équation suivante :

$$y = \sum_i \omega_i \varphi \left(\sum_j u_{ij} x_j + \sum_k \sum_l v_{ikl} x_k x_l + \alpha_i \right) + \beta \tag{1.8}$$

où u et v sont des poids et α et β sont des biais. La figure 1.2 est une représentation de ce neurone avec i, j, k et l prenant les valeurs 1 et 2.

Pao [12] élargit le problème à des entrées qui ont comme prétraitement des fonctions non linéaires, ce prétraitement pouvant varier et n'étant pas appliqué à toutes les entrées.

Par contre le travail qui se rapproche le plus de notre étude est probablement celui de Flake [3] en 1998. Il crée ce qu'il appelle SQUARE-MLP ou simplement SMLP, ce réseau, représentant un MLP où l'on ajoute des entrées élevées au carré, montre qu'il est possible d'avoir des frontières non linéaires. Il montre également qu'un cas particulier du SMLP est une approximation du RBF, l'équation de sortie de ce dernier étant la suivante :

$$F(x) = \sum_{i=1}^{n} \omega_i \exp\left(-\frac{1}{2\sigma_i^2} \|x - x_i\|^2 \right) \tag{1.9}$$

Où $\| x - x_i \|$ représente la distance euclidienne. Après quelques manipulations, on peut réécrire le membre à l'intérieur de la parenthèse sous la forme suivante :

$$\sum_j \left(\frac{-1}{\sigma_i^2} x_{ij} \right) x_j + \sum_k \left(\frac{1}{2\sigma_i^2} \right) x_k^2 + \left(\frac{1}{2\sigma_i^2} x_i^T \cdot x_i \right) = \sum_j \omega_j x_j + \sum_k \omega_k x_k^2 + \omega_i \quad (1.10)$$

Mais, pour avoir une forme gaussienne, Flake utilise une sigmoïde un peu modifiée de façon à ce que la sortie soit :

$$v = 2 - \frac{2}{1 + \exp\left(\sum_j \omega_j x_j + \sum_k \omega_k x_k^2 + \omega_i \right)} \quad (1.11)$$

On remarquera, au chapitre 2, que la sigmoïde qui sera utilisée lors des simulations viendra rejoindre un peu celle de Flake. La figure 1.3 illustre les similitudes entre la gaussienne (1.9) et la sigmoïde (1.11)

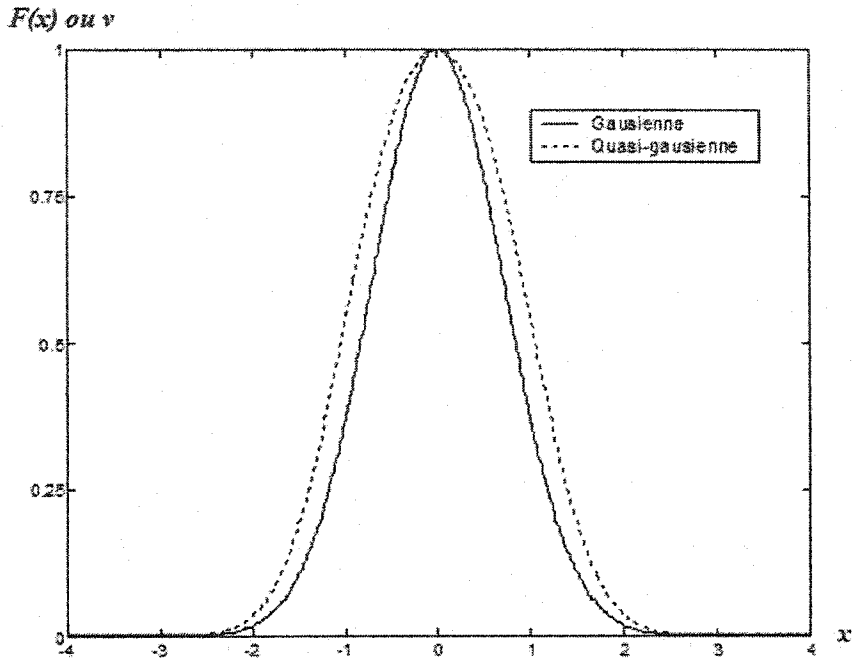


Figure 1.3: Comparaison entre la gaussienne et la sigmoïde avec des entrées quadratiques.

Il est donc possible d'utiliser un MLP comme s'il s'agissait d'un RBF. La manière dont les points tests seront gérés déterminera le type de réponse. Il faut noter que les points tests représentent l'information à priori, ils serviront à entraîner le réseau de neurone.

Dernièrement, Kolean et Hewett [8] montrent que si on élève les variables d'entrée au carré l'entraînement est grandement amélioré. Tandis que Perez et al. [13] utilisent des entrées non linéaires, dont l'exposant est différent de 1, pour améliorer l'apprentissage d'un réseau de neurones pour la reconnaissance de l'écriture. Cet aspect sera un de ceux qui seront étudiés dans ce document, en particulier l'effet que ces entrées non linéaires ont sur la frontière séparatrice.

Au niveau des poids, Kamiura et al. [7] montrent qu'il est possible à un réseau d'apprendre même si un de ses poids est contraint à une valeur spécifique. Cet aspect fut étudié de façon à montrer que si un poids reste bloqué à une valeur dans un circuit électrique, par erreur, il est encore possible au réseau d'apprendre. On montrera dans ce document les contraintes mathématiques de forcer un (ou des) poids à une constante et l'effet que celui-ci (ou ceux-ci) ont sur l'apprentissage. On montrera également comment cet aspect pourrait être utilisé de façon à accélérer l'apprentissage grâce à l'information à priori.

Mais quel est l'effet de ces changements sur la frontière séparatrice ? On montrera qu'il est possible, dans certaines conditions, de forcer un sous-ensemble des poids à des constantes, sans affecter le domaine des solutions. Il sera également démontré qu'il est possible, en conjuguant des poids constants avec des entrées non linéaires, de contrôler la frontière séparatrice, d'un Perceptron, de manière à créer des formes particulières représentant des objets en 2D, 3D, etc. On verra qu'avec quelques petites modifications à l'équation (1.11), il serait possible de décrire un cercle avec la frontière séparatrice. On étudiera aussi d'autres artifices qui permettront d'améliorer l'apprentissage. Ces modifications amèneront des changements à l'algorithme d'apprentissage, une section sera donc réservée à cet aspect.

Par la suite, on vérifiera le comportement du nouveau neurone à travers deux applications, le « packing » et le « covering ». Ces termes seront utilisés à travers ce mémoire à défaut d'avoir trouvé des termes français.

Le « packing » : Ceci consiste à maximiser la couverture d'une surface avec des objets, sans que ceux-ci ne se superposent.

Le « covering » : Ceci consiste à maximiser la couverture d'une surface avec des objets, tout en minimisant la surface de superposition de ceux-ci.

Plusieurs travaux, [1, 10, 18, 20] utilisent les réseaux de neurones pour résoudre des problèmes de « packing », mais ils considèrent uniquement les problèmes ayant pour objets des « polyominos », c'est-à-dire des objets ayant la forme de plusieurs dominos placés côte à côte, ceux-ci devant peuvent et doivent couvrir toute la surface.

On se penchera sur un problème plus général, l'efficacité d'un réseau de neurones à résoudre des problèmes de « packing » et de « covering » si les objets utilisés sont des cercles, ou autres, dans une région délimitée par une frontière aléatoire. Pour ce faire, un nouveau neurone sera développé et placé à l'intérieur d'un réseau de neurones, il sera par la suite entraîné avec l'algorithme de rétro-propagation, celui-ci ayant également été modifié pour inclure les modifications qui auront été apportées au MLP.

CHAPITRE 2 : MODIFICATIONS DU MLP

Comme il a été mentionné, peu d'études ont été faites sur les poids du Perceptron, c'est-à-dire lorsque ceux-ci sont fixés à des constantes. Or, ils font partie des éléments qui déterminent la frontière séparatrice du Perceptron, c'est-à-dire l'élément qui décide de la valeur de la sortie. C'est cette frontière qu'on tentera de manipuler afin de faire par la suite du « packing » et du « covering ». On fera dans un premier temps une étude sur l'effet qu'un poids fixé à une valeur constante a sur la frontière séparatrice. Dans un deuxième temps, on étudiera le prétraitement des entrées et les répercussions que celui-ci a sur la forme de la frontière séparatrice. Dans un troisième temps on examinera une forme généralisée du Perceptron et finalement son utilisation à l'intérieur d'un réseau de neurones.

Comme on l'a vu le Perceptron est composé de cinq parties : les entrées, le biais θ (en général $\theta = 1$), les poids, la fonction de sommation et la fonction d'activation φ.

À la sortie de la première fonction, on a l'équation suivante :

$$v(t) = \sum_{i=0}^{n} \omega_i(t) x_i(t) \tag{2.0}$$

La figure 2.1 représente un Perceptron à n entrées,

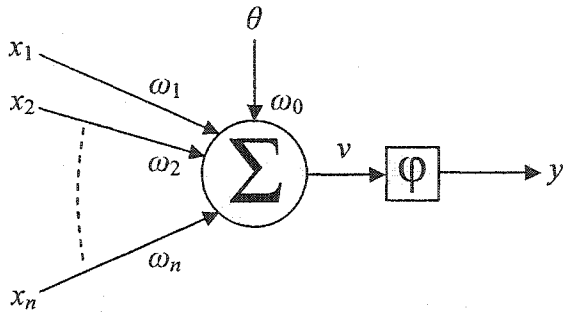


Figure 2.1: Un Perceptron à n entrées.

où x_0 représente θ. La deuxième fonction peut prendre plusieurs formes, idéalement on cherche à approximer une distribution de Heaviside de façon à bien séparer les deux classes, tout en étant dérivable. Cette distribution est donnée par :

$$\varphi(v(t)) = \begin{cases} 0 & \text{si } v(t) \leq 0 \\ 1 & \text{si } v(t) > 0 \end{cases} \tag{2.1}$$

Mais elle a le plus souvent la forme d'une sigmoïde ou d'une tangente hyperbolique. La sigmoïde est donnée par la fonction suivante :

$$\varphi(v(t)) = \frac{1}{1+e^{(-av(t))}} \tag{2.2}$$

La tangente hyperbolique est donnée par la fonction suivante :

$$\varphi(v(t)) = \tanh(av(t)) \tag{2.3}$$

Où a représente un paramètre qui définit le degré de courbure de la fonction. À moins que le contraire soit indiqué, on utilisera dorénavant, pour des raisons de simplicité, la première fonction d'activation, dont la forme graphique est présentée à la figure 2.2 :

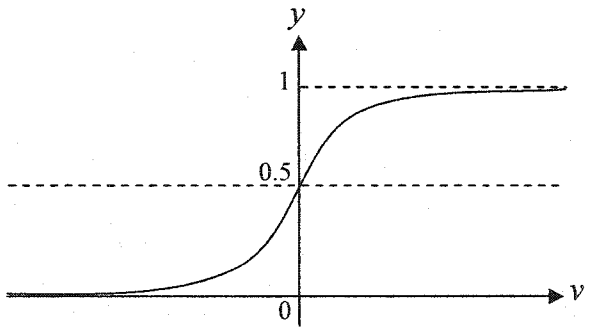


Figure 2.2: Forme de la fonction sigmoïde.

On remarque que la sortie sera, dans le cas de la sigmoïde, non seulement comprise entre [0,1], mais qu'elle aura tendance à être plus près des extrêmes, pour a très grand. On peut alors interpréter la sortie de la façon suivante :

$$x \in \begin{cases} C_1 & \text{si} \quad v < 0 \\ C_2 & \text{si} \quad v > 0 \end{cases} \tag{2.4}$$

où C_1 ($y \simeq 0$) et C_2 ($y \simeq 1$) représentent les deux classes qu'on désire séparer. Donc $v = 0$ représente la frontière séparatrice entre la classe C_1 et la classe C_2. On a maintenant les outils nécessaires pour notre étude.

2.1 LES POIDS

Les poids servent à gérer l'influence que les entrées auront sur la sortie, ils déterminent donc, en partie, la forme de la frontière séparatrice. Prenons $v = 0$, la frontière séparatrice du Perceptron, on peut réécrire l'équation (2.0) sous la forme suivante :

$$x_i(t) = -\frac{1}{\omega_i(t)} \sum_{\substack{j=0 \\ j \neq i}}^{n} \omega_j(t) x_j(t) \tag{2.5}$$

On a alors, comme frontière séparatrice, un hyperplan à n dimensions. Or, il arrive parfois que certaines contraintes, comme la forme de la frontière que l'on cherche ou encore si on veut qu'il y ait plusieurs frontières disjointes, obligent un des poids à être constant, dans quel cas le réseau perd un degré de liberté. Il faut donc vérifier s'il pourra quand même effectuer une séparation des deux classes avec un poids constant.

2.1.1 LEMME 1

Soit $\mathscr{D}$ l'ensemble des hyperplans possibles d'un Perceptron à n dimensions. Soit également C_1 et $C_2 \in R^n$, deux classes dont les éléments sont linéairement séparables, $\mathscr{F}_\omega$, la frontière séparatrice, appartenant à $\mathscr{D}$, qui les sépare parfaitement et ω' le vecteur des coefficients de l'hyperplan $\mathscr{F}_\omega$.

Prenons un poids, ω_i, $i \in [1, n]$, et fixons le à k, $k \in R$, durant la période d'apprentissage. Alors, il existe un $\omega \neq \omega'$, tel que $\mathscr{F}_\omega$ sépare parfaitement C_1 et C_2, $\forall\, \omega_i \neq 0$.

Preuve :

Prenons l'équation (2.5) de façon à ce que l'entrée x_i soit celle où $\omega_i = k$:

$$x_i(t) = -\frac{1}{k}\sum_{\substack{j=0 \\ j\neq i}}^{n} \omega_j(t)x_j(t) \tag{2.6}$$

Si on rentre la constante à l'intérieur de l'équation et qu'on réintroduit y, l'équation (2.6) devient :

$$v(t) = x_i(t) + \sum_{\substack{j=0 \\ j \neq i}}^{n} \left(\frac{\omega_j(t)}{k} \right) x_j(t) \tag{2.7}$$

Finalement, rentrons x_i à l'intérieur de la sommation :

$$v(t) = \sum_{j=0}^{n} \left(\frac{\omega_j(t)}{k} \right) x_j(t) \tag{2.8}$$

où avec $\omega_j(t) = k$ et $k \neq 0$

On note que l'équation (2.6) à la même forme que l'équation de départ (2.0). On doit également noter que ω_j s'adaptera à la nouvelle valeur ω_i de façon à conserver le même rapport (ω_j / ω_i). Donc le réseau pourra trouver une frontière séparatrice $\mathscr{F}_\omega$ qui sépare parfaitement les deux classes. La seule exception étant le cas où $\omega_i = 0$, ceci implique qu'il est en pratique impossible que ω_j conserve le même rapport.

Ceci est vrai en théorie lorsque les classes contiennent une infinité de points. Mais, en pratique l'ensemble des points représentant chacune des classes C_1 et C_2 est fini, il est donc possible de trouver un hyperplan qui les sépare.

2.1.2 LEMME 2

Étant donné les hypothèses du lemme 1, et en rajoutant l'hypothèse que les deux classes, C_1 et C_2, sont composées d'un nombre fini d'éléments, il sera alors possible d'enlever la restriction $\omega_i \neq 0$.

Donc si un poids ω_i, $i \in [1, n]$, un élément de ω est forcé à la valeur 0 durant le processus d'apprentissage, il sera alors possible de trouver un $\omega \neq \omega'$ tel que $\mathscr{F}_\omega$ sépare parfaitement C_1 et C_2.

Preuve :

Posons les définitions suivantes :

p_i :le point le plus près de la frontière séparatrice $\mathscr{F}_\omega$, pour la classes i.

l_i :les points dont leur projection sur la frontière séparatrice $\mathscr{F}_\omega$, sera la plus grande distance euclidienne.

d_i :la distance entre le point p_i et la frontière séparatrice $\mathscr{F}_\omega$.

Puisque $d_i > 0$, il existe une infinité d'hyperplans parallèles à $\mathscr{F}_\omega$, entre les p_i. L'ensemble de ces hyperplans forme un couloir dans lequelle tous les hyperplans

séparent parfaitement les deux classes. Donc, même si $\omega_i^{'} = 0$, il existe en pratique un autre hyperplan, où $\frac{\omega_j}{\omega_i^{'}} \neq \infty$, qui satisfait l'équation (2.4).

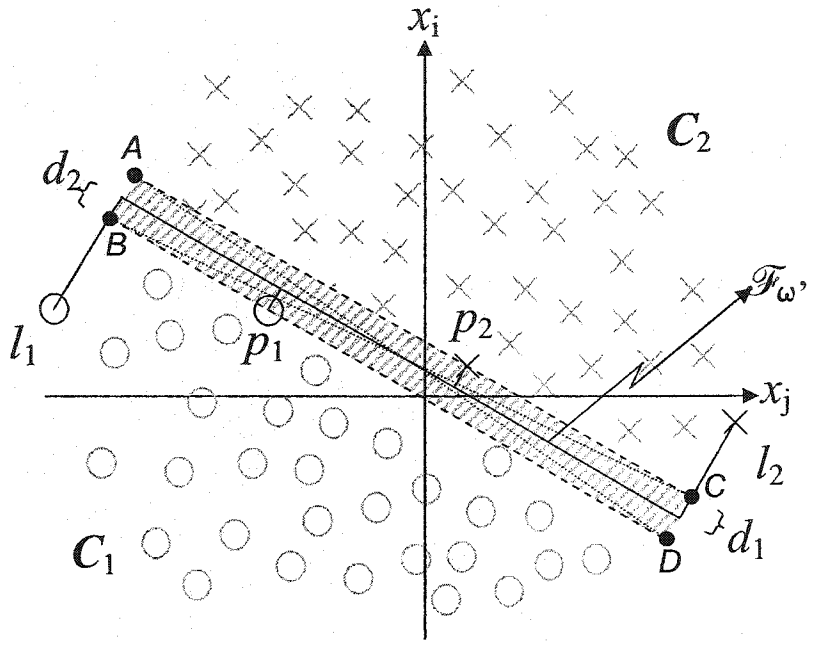


Figure 2.3: Bande comprenant tous les hyperplans pouvant séparer correctement les deux classes.

Prenons le cas où on cherche la valeur d'un ω_j particulier, on prendra alors le plan x_j-x_i, les autres dimensions n'étant pas composées du ω_j particulier. On voit bien, dans la figure 2.3, la bande qui représente toutes les droites possibles qui séparent les deux classes dans ce plan. On remarque également qu'il est possible de déduire l'intervalle de ω_j pour un $\omega_i = k$ donné. On a :

où les fractions représentent la plus petite et la plus grande pente possible à l'intérieur de la bande. Il faut noter que dans le cas où la pente serait positive les limites de ω_j seraient de signes opposés. On peut également déduire les valeurs des points A (A_{x1}, A_{x2}, …, A_{xn}), B (B_{x1}, B_{x2}, …, B_{xn}), C (C_{x1}, C_{x2}, …, C_{xn}), et D (D_{x1}, D_{x2}, …, D_{xn}) à partir des points l_i et p_i.

Donc, dans le cas où on aurait un poids fixé à une constante, il sera toujours possible au Perceptron de trouver une frontière séparatrice.

Il est toutefois possible qu'on veuille forcer plus d'un poids à une constante, dans quel cas le problème pourrait ne plus être séparable, puisque un des coefficients de l'équation (2.8) serait constant. Le Perceptron ne pourra donc pas optimiser la pente de la droite dans un des plans, ce qui pourrait rendre le problème insoluble.

2.1.3 LEMME 3

Étant données les hypothèses du lemme 1 et du lemme 2 et x le vecteur, de longueur n, des entrées d'un Perceptron, ω le vecteur des poids associés au vecteur x. Soit également, x_s, un sous-ensemble du vecteur x, le vecteur, de longueur n_s, des entrées dont les poids ω_s, un sous-ensemble du vecteur ω, qui leur est associé et gardé constant durant l'apprentissage.

Pour un nombre n_s de poids constants, il faut alors ajouter au minimum (n_s-1) entrées, ceux-ci étant des fonctions $f(x_{sm})$, et leur poids au Perceptron de façon à assurer une séparabilité des deux classes C_1 et C_2.

Preuve :

Deux cas peuvent se produire, la première étant le cas où toutes les fonctions $f(x_{sm})$ sont linéaires et la deuxième étant le cas où certaines, ou toutes, les fonctions $f(x_{sm})$ sont non linéaires. L'équation (2.10) montre l'équation (2.5) avec n_s poids fixes :

$$v(t) = x_{s_1}(t) + \sum_{m=2}^{n_s} \frac{k_m}{k_1} x_{sm}(t) + \sum_{\substack{j=0 \\ j \neq km}}^{n} \left(\frac{\omega_j(t)}{k_1} \right) x_j(t) \tag{2.10}$$

où k_1 représente la première constante et k_m les suivantes.

Cas 1: Les fonctions $f(x_{sm})$ sont linéaires

Ce cas est facile, puisque les nouvelles entrées sont linéaires, il suffit de les factoriser avec les entrées déjà existantes de façon à avoir des nouveaux poids. Ceci implique que toutes les nouvelles entrées seront de la forme suivante :

$$f(x_{sm}) = ax_{sm} + b \tag{2.11}$$

En rajoutant ces entrées à l'équation (2.10), on obtient l'équation suivante :

$$v(t) = x_{s_1}(t) + \sum_{m=2}^{n_s} \frac{k_m}{k_1} x_{sm}(t) + \sum_{\substack{j=0 \\ j \neq km}}^{n} \left(\frac{\omega_j(t)}{k_1} \right) x_j(t) + \sum_{p=2}^{n_s} \omega_p(t) f(x_{sp}) \tag{2.12}$$

En substituant $f(x_{sp})$ avec l'équation (2.11), puis en factorisant les entrées ensemble, on obtient l'équation suivante :

$$v(t) = x_{s_1}(t) + \sum_{m=2}^{n_s} \left(\frac{k_m}{k_1} + \omega_m(t) \right) x_{sm}(t) + \sum_{\substack{j=0 \\ j \neq km}}^{n} \left(\frac{\omega_j(t)}{k_1} \right) x_j(t) + \sum_{p=2}^{n_s} \omega_p(t) b_p \tag{2.13}$$

On remarque à l'équation (2.13) que tous les poids sont variables sauf un, ce qui est soluble d'après le lemme 1, où le réseau était capable de trouver une solution avec un seul poids fixe.

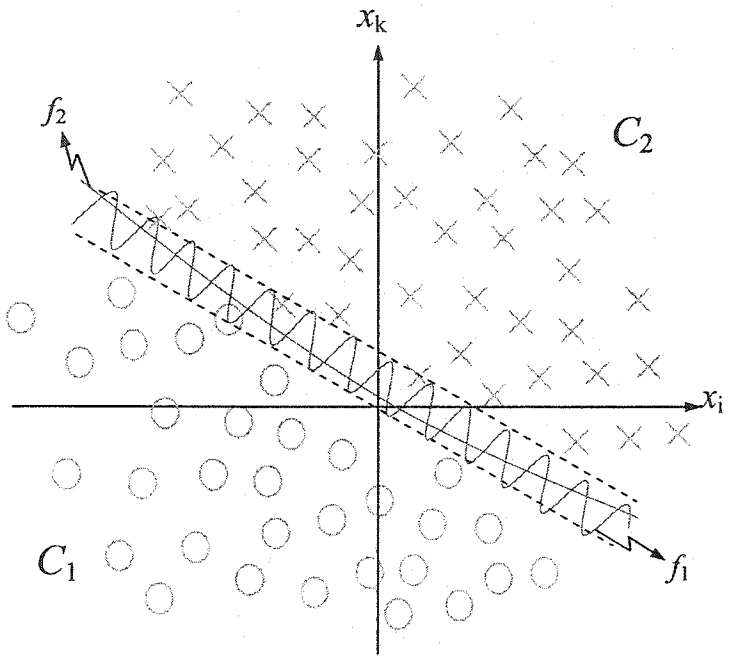


Figure 2.4: Fonctions non linéaires f_1 et f_2 à l'intérieur de la bande séparatrice qui sépare parfaitement les deux classes $\boldsymbol{C}_1$ et $\boldsymbol{C}_2$.

Cas 2: Les fonctions $\boldsymbol{f}(\boldsymbol{x}_{sm})$ sont non linéaires

Le deuxième cas n'est pas si simple, puisque l'ajout d'entrées non linéaires rendrait la frontière séparatrice $\mathscr{F}_\omega$ non linéaire, et puisque le problème est par hypothèse linéairement séparable, il est impossible que la frontière puisse séparer correctement les deux classes. Mais, il y a également l'hypothèse que les classes $\boldsymbol{C}_1$ et $\boldsymbol{C}_2$ sont composées d'un nombre fini d'éléments, ceci implique qu'il est possible de résoudre le problème en se servant de la bande utilisée au lemme 2, puisque celle-ci peut contenir une fonction non linéaire (figure 2.4).

Il est également possible en utilisant une forme tronquée de la série de Taylor de réécrire les fonctions $f(x_{sm})$ sous la forme suivante :

$$f(x_{sm}) = \sum_{q=0}^{r} \frac{(x_{sm} - x_{sm0})^q f^{(q)}(x_{sm0})}{q!} + R_r \tag{2.14}$$

où R_r est le reste. En insérant l'équation (2.14) dans l'équation (2.11) on obtient l'équation suivante :

$$\begin{aligned} v = x_{s_1}(t) + \sum_{m=2}^{n_s} \left(\frac{k_m}{k_1} + \omega_{m_1}(t) f(x_{sm_0}) \right) x_{sm}(t) + \sum_{\substack{j=0 \\ j \neq km}}^{n} \left(\frac{\omega_j(t)}{k_1} \right) x_j(t) + \\ \sum_{p=2}^{n_s} \left[\omega_p(t) R_{rp} - \omega_p(t) x_{sp0} f(x_{sp0}) + \sum_{\substack{q=0 \\ q \neq 1}}^{r} \omega_p(t) \left(\frac{(x_{sp} - x_{sp0})^q f^{(q)}(x_{sp0})}{q!} \right) \right] \end{aligned} \tag{2.15}$$

Si tous les poids ω_p étaient nuls la deuxième ligne de l'équation disparaîtrait, on aurait alors une équation qui ressemblerait à l'équation (2.11) avec un seul poids fixe, et basé sur le lemme 1, ce poids ne restreint pas le réseau lors de l'apprentissage. Mais les ω_p peuvent ne pas être nuls, dans quel cas la frontière séparatrice serait non linéaire. Naturellement, dans les deux cas il se peut que k_1 soit nulle, mais dans ce cas le problème peut être résolu d'après le lemme 2.

On remarque aussi que le lemme 3 ressemble au théorème de Cover [5] :

> Une dichotomie de dimension n qui est linéairement non séparable peut l'être si le problème est projeté dans un espace de dimension plus grande.

$$\begin{array}{llll} \omega x > 0 & x \in C_1 & \omega f(x) > 0 & f(x) \in C_1 \\ \omega x < 0 & x \in C_2 & \omega f(x) < 0 & f(x) \in C_2 \end{array} \longrightarrow \qquad (2.16)$$

Le théorème de Cover montre qu'un problème qui n'est pas linéairement séparable peut l'être si on augmente sa dimension. Cependant le lemme 3 diffère du théorème de Cover au niveau des hypothèses initiales, il part d'un problème qui est déjà linéairement séparable à *n* dimensions, donc les entrées supplémentaires servent à mieux contrôler la forme de la frontière séparatrice.

Mais, le rajout d'entrées peut modifier la forme de la frontière séparatrice, ce qui peut être non voulu.

2.1.4 PROPOSITION 1

Soit C_1 et C_2 deux classes linéairement séparables et ω_s, de longueur n_s, le sous-ensemble des poids constants d'ω. Le réseau sera capable de trouver une frontière séparatrice qui sépare parfaitement les deux classes C_1 et C_2 si on lui ajoute une couche. Il doit cependant y avoir n neurones sur la première couche et un seul sur la deuxième.

Si tous les poids d'une des deux couches sont fixés il sera toujours possible au réseau de trouver des valeurs au poids restant de façon à bien séparer les deux classes. Deux contraintes s'imposent :

- ***les poids (excluant le biais) d'un neurone ne peuvent pas être nuls.***
- ***le cas où tous les poids de la première couche seraient fixés, toutes les frontières décrites par les neurones de cette couche devront être différentes.***

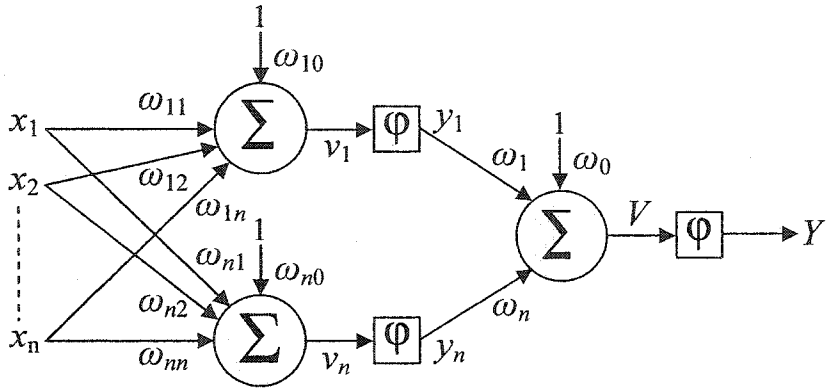


Figure 2.5: Réseau de neurones à deux couches et à *n* entrées.

Preuve :

Insérons notre Perceptron à l'intérieur d'un réseau à deux couches et à *n* entrées comme à la figure 2.5, avec des fonctions d'activation sigmoïdales.

On aura alors comme sortie :

$$Y = \varphi\left(\sum_{i=0}^{n} \omega_i y_i\right) \tag{2.17}$$

où y_1 et y_2 représentent les sorties de la première couche. On a donc :

$$y_n = \varphi\left(\sum_{i=0}^{n} \omega_{ni} x_i\right) \tag{2.18}$$

Insérons (2.12) dans (2.11), puis (2.11) dans (2.10) pour obtenir l'équation suivante :

$$Y = \frac{1}{1+e^{\left(\sum_{j=0}^{n} \omega_j \frac{1}{1+e^{\left(\sum_{i=0}^{n} \omega_{ji} x_i\right)}}\right)}} \tag{2.19}$$

D'après l'équation (2.19) deux cas sont possibles, la première est de fixer les poids de la première couche, la deuxième est de fixer les poids de la deuxième couche. Il est important de se rappeler que les deux classes sont linéairement séparables par hypothèse.

Cas 1: Les poids de la première couche sont fixés

Ce cas force certaines transformations spatiales sur les entrées et laisse les poids, ω_j, de la deuxième couche varier de façon à trouver un hyperplan qui satisfait le problème. Étant donné que ces transformations sont linéaires de nature, il ne change pas la nature linéaire du problème, donc il reste linéairement séparable. La seule contrainte est sur le choix des poids fixes, ils doivent composer au moins n frontières différentes afin d'assurer que le problème soit linéairement séparable à l'entrée de la deuxième couche. Il faut également mentionner que tous les poids d'un neurone ne peuvent pas être nul, car ceci aurait pour effet d'annuler ce neurone, ce qui à son tour pourrait diminuer la dimension du problème en une dimension plus petite que la dimension initiale. Dans ce premier cas, les ω_j servent à compenser les poids de la première couche.

Cas 2: Les poids de la deuxième couche sont fixés

Ce cas laisse le réseau décider des transformations spatiales à effectuer de façon à satisfaire l'hyperplan décrit par le neurone de sortie, c'est-à-dire la deuxième couche. Encore une fois, tous les poids du neurone de sortie ne peuvent pas être nul puisque cela aurait pour effet de rendre le problème non soluble, la sortie serait une constante. Dans ce deuxième cas, les ω_{ji} servent à compenser les poids de la deuxième couche.

Regardons quelques exemples en deux dimensions pour faciliter l'illustration de cette proposition.

Supposons $n'= n$, il est donc possible que les classes C_1 et C_2 ne soit pas séparables après la première couche, la figure 2.6 illustre un cas particulier de ceci. On remarque, qu'aucun des deux neurones de la première couche n'est capable de séparer les deux classes.

La figure 2.7 montre le plan x_1 et x_2 vu du neurone de la deuxième couche. On note qu'il sera en mesure de séparer les deux classes, et d'après le lemme 1, il pourrait aussi avoir un de ses poids fixé à une constante.

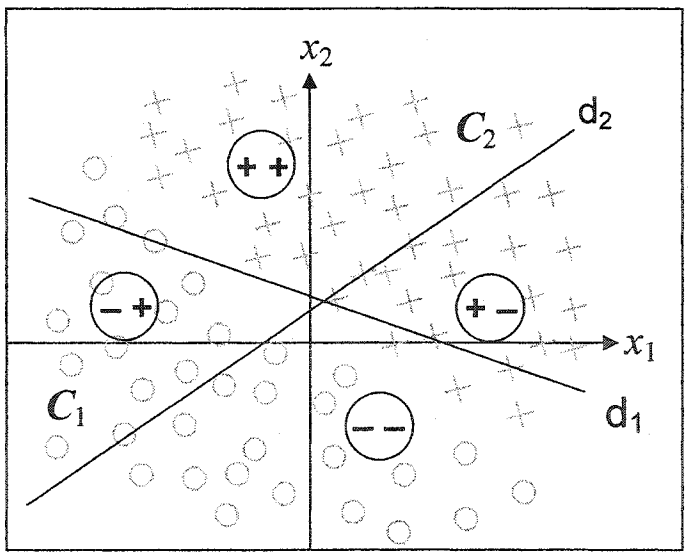


Figure 2.6: Séparation des deux classes C_1 et C_2 par les droites d_1 et d_2 vues de la première couche.

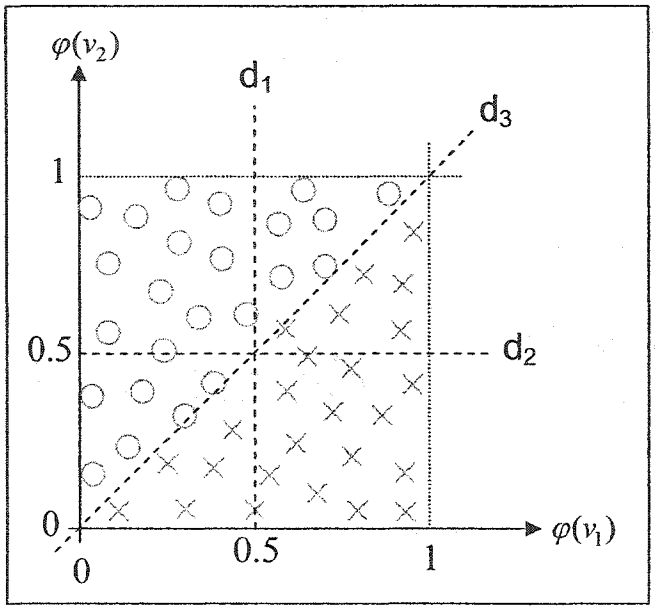


Figure 2.7: Séparation des deux classes C_1 et C_2 par la droite d_3, vues de la deuxième couche.

Voyons un autre exemple, où la constante d'apprentissage a est assez élevée et les droites d_1 et d_2 se coupent, non pas entre les deux classes, mais à l'intérieur de l'une d'elle.

La figure 2.8 a) montre les classes C_1 et C_2 et la frontière séparatrice $\mathscr{F}_\omega$. On remarque que la classe C_1 est comprise seulement dans la région où d_1 et d_2 sont négatifs, tandis que la classe C_2 est comprise dans les quatre régions. Étant donné que la constante d'apprentissage a est grande, on remarque dans la figure 2.8 b) que les points ont tendance à se pousser vers les coins, ce qui rend en général la tâche de la deuxième couche un peu plus simple. Mais, ce n'est pas notre cas puisque dans le cadran inférieur gauche il y a des points appartenant aux deux classes.

couche un peu plus simple. Mais, ce n'est pas notre cas puisque dans le cadran inférieur gauche il y a des points appartenant aux deux classes.

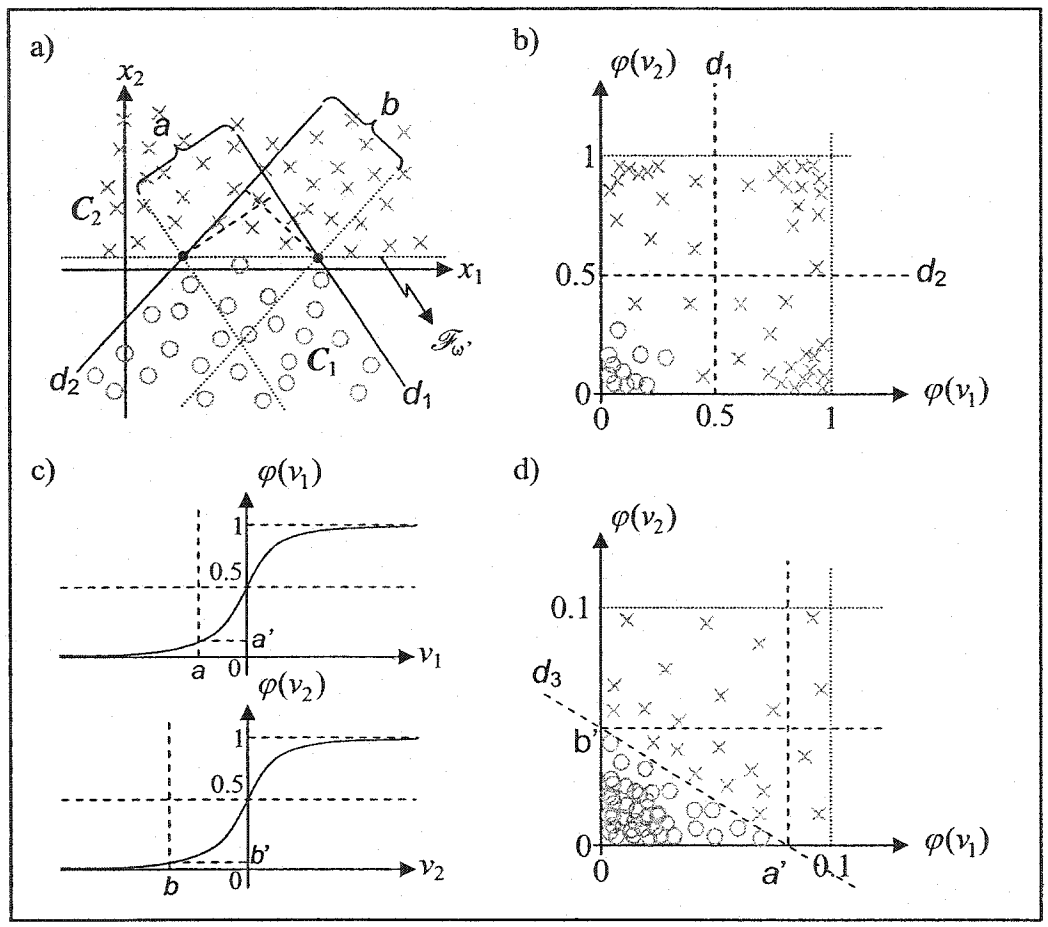


Figure 2.8: a) Les classes C_1 et C_2 vues de la première couche, b) Les classes C_1 et C_2 vues de la deuxième couche, c) Les droites séparatrices vues par les sigmoïdes, d) Vue rapproché du troisième quadrant.

Regardons de plus près ce qui ce passe dans ce cadran. La figure 2.8 d) montre que les points de la classe C_1 se trouvent en dessous d'une droite reliant les points a' et b'. Ils sont respectivement $\varphi(a)$ et $\varphi(b)$ (figure 2.8 c)), où a et b représente la distance maximale que les points d'une classe peuvent avoir par rapport à une des deux droites d_1

et d_2. On constate encore une fois que les deux classes, C_1 et C_2, ont pu être séparées par le neurone de la deuxième couche, d_3 pouvant être déduit de a' et b'.

Il peut maintenant y avoir autant de poids fixés qu'il est nécessaire et le réseau pourra quand même séparer les deux classes. Il suffit de rajouter une deuxième couche ou des entrées à la première pour pouvoir trouver la frontière séparatrice. On a vu que lorsqu'on rajoute des couches, la forme de la frontière séparatrice reste linéaire, mais ce n'est pas nécessairement le cas lorsqu'on rajoute des entrées.

Jusqu'ici on ne s'est concentré que sur le réseau de neurones, on lui a rajouté des entrées non linéaires, des poids constants et des couches supplémentaires si nécessaire. Ces nouvelles options qui ont été démontrées à travers les différents lemmes ont permis un meilleur contrôle de la frontière séparatrice utilisée par le réseau de neurones, et par conséquence, ils permettent un meilleur contrôle du réseau de neurones lui-même. Ceci sera illustré au chapitre 3, mais on doit d'abord pouvoir entraîner ce réseau. On se servira pour cette tâche de l'algorithme de rétro-propagation.

2.2 MODIFICATION DE L'ALGORITHME DE RÉTRO-PROPAGATION ET GÉNÉRALISATION DU PERCEPTRON

L'algorithme a été conçu pour un réseau dont les neurones ressemblent à des Perceptrons. Il faudra donc le modifier pour prendre en considération les entrées non

linéaires et les poids fixes. Les étapes 1, 2, 3 et 5 ne changeront pas, seule l'étape 4 devra changer et ce, lors de la descente du gradient dont l'équation est la suivante :

$$\varphi_j'\left(v_j^{(l)}(t)\right)=\frac{\partial\left(y_j^{(L)}\right)}{\partial\left(v_j^{(L)}\right)}\frac{\partial\left(v_j^{(L)}\right)}{\partial\left(y_i^{(L-1)}\right)}\frac{\partial\left(y_i^{(L-1)}\right)}{\partial\left(v_i^{(L-1)}\right)}\cdots\cdots\frac{\partial\left(y_h^{(l)}\right)}{\partial\left(v_h^{(l)}\right)}\frac{\partial\left(v_h^{(l)}\right)}{\partial\left(\omega_{hg}^{(l)}\right)} \qquad (2.20)$$

Ce sont plus précisément les dérivées $\partial\left(v_j^{(l)}\right)/\partial\left(y_i^{(l-1)}\right)$ et $\partial\left(v_j^{(l)}\right)/\partial\left(\omega_{ji}^{(l)}\right)$ qui devront changer.

Initialement, on avait :

$$\frac{\partial\left(\mathrm{v}_j^{(l)}\right)}{\partial\left(\mathrm{y}_i^{(l-1)}\right)}=\omega_{ji}^{(l)} \quad \text{et} \quad \frac{\partial\left(\mathrm{v}_j^{(l)}\right)}{\partial\left(\omega_{ji}^{(l)}\right)}=y_i^{(l-1)} \qquad (2.21)$$

Une fois les changements apportés, les dérivées deviennent :

$$\frac{\partial\left(\mathrm{v}_j^{(l)}\right)}{\partial\left(\mathrm{y}_i^{(l-1)}\right)}=\frac{\partial\left(\mathrm{f}\left(y^{(l-1)},\omega^{(l)}\right)\right)}{\partial\left(\mathrm{y}_i^{(l-1)}\right)} \quad \text{et} \quad \frac{\partial\left(\mathrm{v}_j^{(l)}\right)}{\partial\left(\omega_{ji}^{(l)}\right)}=\frac{\partial\left(\mathrm{f}\left(y^{(l-1)},\omega^{(l)}\right)\right)}{\partial\left(\omega_{ji}^{(l)}\right)} \qquad (2.22)$$

Naturellement, les dérivées (2.21) sont des cas particuliers de (2.22). Cette modification à l'algorithme a pour effet de changer l'équation (1.6), l'importance des modifications dépend de la fonction *f*.

On peut maintenant généraliser la forme du Perceptron (figure 2.9). Au lieu d'avoir une sommation comme fonction à l'intérieur du neurone, on peut avoir une fonction F qui contient un nombre Ω de paramètres et le vecteur d'entrée x sans prétraitement. La seule condition est que F soit dérivable.

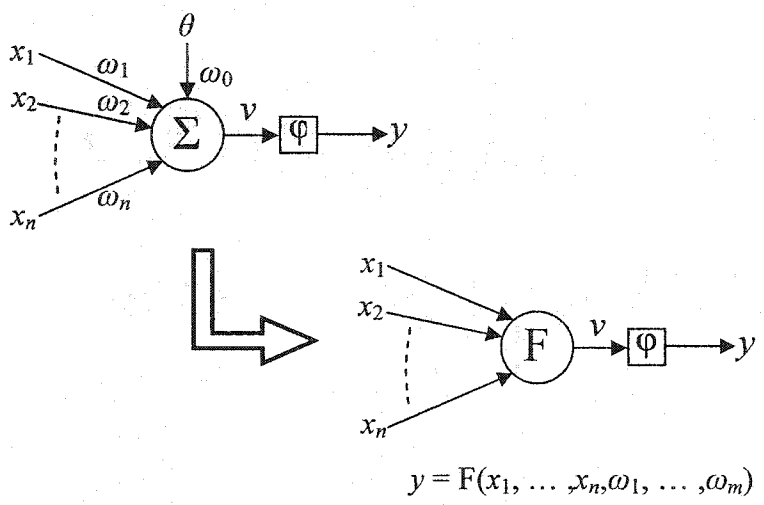


Figure 2.9: Forme du nouveau neurone.

On a donc pu développer un neurone qui permet de caractériser une fonction F qui n'est pas une simple sommation, c'est-à-dire qui peut avoir des poids constants et des entrées non linéaires. Inversement, ces améliorations permettent au neurone de décrire une frontière séparatrice non linéaire et à l'usager un meilleur contrôle sur la forme de cette dernière. Avec ce nouveau neurone on peut résoudre plus facilement des problèmes qui étaient difficile, voir impossible, à résoudre.

CHAPITRE 3 : GÉOMÉTRIE COMBINATOIRE, RÉSULTATS ET CONCLUSION

Comme tous les problèmes reliés à la géométrie combinatoire, la solution doit être vérifiée expérimentalement. On effectuera, pour des raison de vérification, deux types de simulations : le « packing » et le « covering » d'un carré avec des cercles. Mais on verra tout d'abord un aperçu de la géométrie combinatoire.

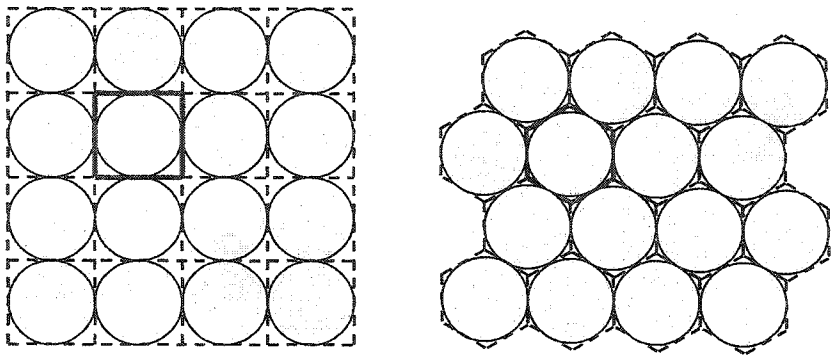
3.1 GÉOMÉTRIE COMBINATOIRE

La géométrie combinatoire est un domaine des mathématiques qui a reçu beaucoup d'attention depuis son apparition. Kepler fut probablement le premier a publié sur le sujet, mais d'autres comme Newton, Gauss, Lagrange et plusieurs autres mathématiciens s'y sont consacrés. On se limitera ici à deux sous domaines de la géométrie combinatoire, c'est-à-dire le « packing » et le « covering ».

3.1.1 LE « PACKING »

Le « packing » est l'agencement de n objets qui remplissent au maximum un espace donné sans se chevaucher. Comme par exemple le nombre maximal de verres que l'on peut mettre à l'intérieur d'une boîte, ou encore comment placer x verres à l'intérieur

d'une boîte. On appelle le treillage : un carrelage de polygone qui permet d'avoir un « packing » particulier.



« Packing » carré **« Packing » hexagonal**

Figure 3.1: Treillage carré et hexagonal pour le « packing ».

C'est probablement au 16^{e} siècle [4] que la question du « packing » de sphères fut étudiée pour la première fois. Un certain Sir Walter Raleigh demanda à son mathématicien, T. Hariot, de développer des formules pour évaluer le nombre de boulets de canon dans une pile régulière. En 1591, il donna, à Raleigh, sa solution sous la forme d'une table basée sur le triangle de Pascal avant même que celui-ci ne l'eut inventé (figure 3.1).

Kepler, à cause d'une correspondance avec Hariot, énonça en 1611, ce qui devint communément appelé « la conjecture de Kepler », que le plus dense treillage pour le « packing » de sphères (k_3), k_2 pour le cercle, est le treillage cubique à face centrée ou hexagonale, ceci représente les premiers résultats sur le problème de « packing ».

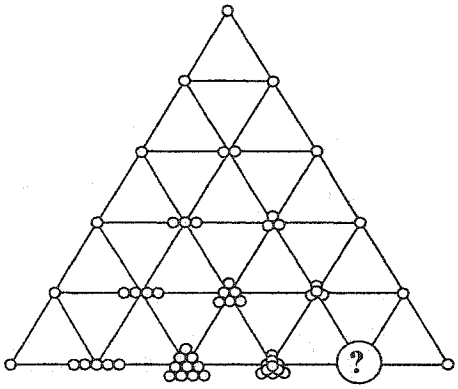


Figure 3.2: Table T. Hariot basée sur le triangle de Pascal.

Lagrange donna le résultat suivant pour k_2 en 1773 :

$$\delta(k_2) \leq \frac{\pi}{\sqrt{12}} = 0.9069 \tag{3.1}$$

Où ρ est une fonction qui donne la densité maximale d'un système k_2, k représente la forme utilisée, dans ce cas un cercle, et le 2 la dimension du système.

En 1831 Gauss montre qu'un treillage cubique à face centrée est le treillage régulier le plus dense pour k_3, mais il laisse ouvert la possibilité qu'un treillage irrégulier le soit encore plus. Tandis que Thue en donne la preuve pour k_2 en 1892 et l'améliore en 1910. En 1953, Fejes Tóth, réduit le problème de k_3 en plusieurs sous problèmes, mais il a fallu attendre l'arrivée des ordinateurs pour que Hales en donne la preuve en 1998, 387 ans après la publication de la conjecture de Kepler. k_4 et k_5 on été trouvés en 1872 et 1877, respectivement par Korkin et Zolotarev. Blichfeldt (1934), Barnes (1957) et Vetcinkin

(1980) trouvent les valeurs de k_6 à k_7. Minkowski généralise la densité pour n dimensions en 1893. Dans les années 40 et 50, des bornes, plus précises que celles de Minkowski, sont émises pour k_9 à k_{12}. Finalement, Rogers améliore la formule de Minkowski en 1947.

Le tableau 3.1 montre les valeurs connues pour le « packing » d'« hypersphères » en n dimensions pour un espace infini. δ_n représente la densité maximale que peut avoir le « packing » d'« hypersphères » dans un espace infini.

Tableau 3.1: Valeurs connues pour le « packing » d' « hypersphères ».

n	δn	Références
2	$\frac{1}{6}\pi\sqrt{3}$	Kepler 1611, 1619, Lagrange 1773
3	$\frac{1}{6}\pi\sqrt{2}$	Kepler 1611, 1619, Gauss 1840
4	$\frac{1}{16}\pi^2$	Korkin et Zolotarev 1877
5	$\frac{1}{30}\pi^2\sqrt{2}$	Korkin et Zolotarev 1877
6	$\frac{1}{144}\pi^2\sqrt{3}$	Blichfeldt 1934, Barnes 1957, Vetcinkin 1980
7	$\frac{1}{105}\pi^3$	Blichfeldt 1934, Barnes 1957, Vetcinkin 1980
8	$\frac{1}{384}\pi^4$	Blichfeldt 1934, Barnes 1957, Vetcinkin 1980

Dans le cas où l'espace serait fini et carré, on aurait l'agencement optimal montré à la figure 3.3.

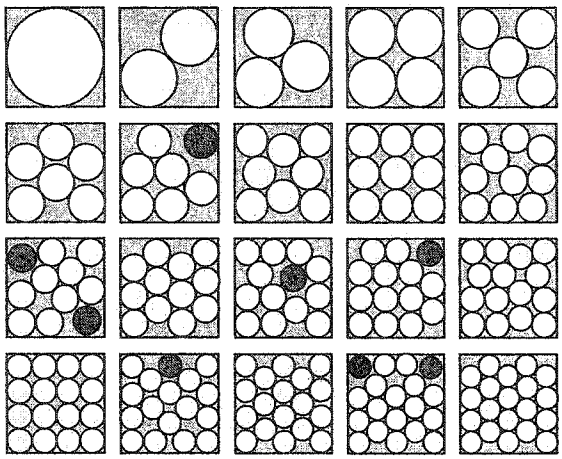


Figure 3.3: Agencement de cercles à l'intérieur d'un carré, les cercles qui sont plus foncés représentent des cercles pouvant bouger librement.

Le tableau 3.2 montre le rayon des cercles qui permettent un « packing » maximal dans un carré unitaire.

Tableau 3.2: Rayon maximal qui permet le « packing » de n cercles à l'intérieur d'un carré.

n	Rayon du cercle	Références	n	Rayon du cercle	Références
2	0.2929...	(élémentaire)	12	0.1400...	Peikert
3	0.2543...	(élémentaire)	13	0.1340...	Peikert
4	0.2500	(élémentaire)	14	0.1293...	Wengerodt
5	0.2071...	(élémentaire)	15	0.1272...	Peikert
6	0.1877...	Graham, Melissen	16	0.1250	Wengerodt
7	0.1745...	Schaer	17	0.1172...	Peikert
8	0.1705...	Schaer et Meir	18	0.1155...	Peikert
9	0.1667	Schaer	19	0.1123...	Peikert
10	0.1482...	Peikert	20	0.1114...	Peikert
11	0.1424...	Peikert	25	0.1000	Peikert

3.1.2 LE « COVERING »

À l'inverse du « packing » on a le « covering ». Au lieu de maximiser le nombre d'objets k_n qui peuvent occuper un espace donné, on cherche à minimiser le nombre d'objets k_n nécessaires afin de couvrir entièrement un espace donné. Comme par exemple le nombre minimum d'antennes nécessaire afin de couvrir toute une surface, ou encore comment placer x antennes de façon à couvrir une certain surface.

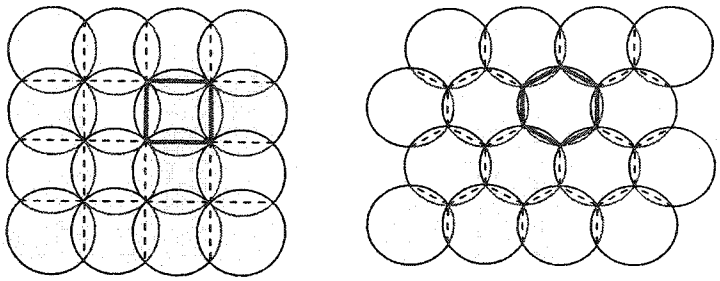


Figure 3.4: Treillage carré et hexagonal pour le « covering ».

Pour un espace infini, le tableau 3.3 montre le recouvrement minimal v_n nécessaire afin de couvrir un espace de n dimensions donnée avec des « hypershères ».

Tableau 3.3: Valeurs connues pour le « covering » d'« hypersphères »

n	vn	Références
2	$\frac{2\pi}{3\sqrt{3}}$	Kershner 1939
3	$\frac{5\sqrt{5}\pi}{24}$	Bambah 1954
4	$\frac{2\pi^2}{5\sqrt{5}}$	Delone et Ryškov 1963
5	$\frac{245\sqrt{35}\pi^2}{3888\sqrt{3}}$	Baranovskii et Ryškov 1975

Le tableau 3.4 montre les valeurs connues pour un carré unitaire avec des cercles.

Tableau 3.4: Rayon minimal qui permet le « covering » de n cercles à l'intérieur d'un carré.

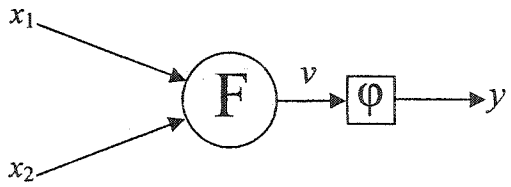
n	Rayon du cercle	Références
2	$\frac{\sqrt{5}}{4}$	(élémentaire)
3	0.5039...	Heppes et Melissen
4	$\frac{1}{2\sqrt{2}}$	Heppes et Melissen
5	0.3262...	Heppes et Melissen
7	0.2743...	Heppes et Melissen

3.2 SIMULATIONS

On a défini le « packing » et le « covering » et on a montré la difficulté qu'il existe de trouver, non seulement la position optimale, mais aussi le rayon optimal. On a également développé un nouveau type de neurone qui permet d'avoir des frontières non linéaires et disjointes avec les autres neurones de la même couche. Ce nouveau neurone semble donc bien adapté au problème de « packing » et « covering ».

La première étape serait de choisir une fonction F dérivable qui définit la forme de l'objet qui servira à paver, couvrir, stocker, etc. Le vecteur d'entrée représentera alors les points tests.

On traitera dans cette recherche une seule forme, les cercles. On aura donc un neurone, qu'on appellera « circulaire », de la forme suivante :



$$F = (x_1 - \omega_1)^2 + (x_2 - \omega_2)^2 - r^2$$

Figure 3.5: Neurone circulaire.

où x_1 et x_2 représenteront les coordonnées des points tests de la surface à couvrir et F la fonction représentant le cercle.

L'équation (2.2) devient alors :

$$y = \frac{1}{1 + e^{\left(x_1^2 + x_2^2 - 2\omega_1 x_1 - 2\omega_2 x_2 + \omega_1^2 + \omega_2^2 - r^2\right)}} \tag{3.2}$$

Les poids de x_1^2 et x_2^2 ont pu être fixés à 1 d'après le lemme 2. Il doit également être mentionné que $\omega_1^2 + \omega_2^2 - r^2$ représente le biais du neurone, et donc une troisième constante puisqu'elle est recalculé a chaque itération et non modifier lors de la rétro-propagation de l'erreur. On remarque également que la fonction F ressemble à l'équation (1.11) de Flake, mais elle en diffère de deux manières. La première étant d'un facteur (-2) et l'addition d'une constante (+2), mais ceci a peu d'impact sur la forme de la courbe.

La deuxième est plus importante, r^2, elle permet de vérifier si le point (x_1, x_2) est à l'intérieur du cercle de rayon r ou pas.

Le neurone, figure 3.5, sera placé à l'intérieur d'un réseau de neurones de deux couches, figure 3.6, la première sera composée de k neurones « circulaires », où k représente le nombre d'objets utilisés. La deuxième sera composée d'un seul Perceptron, on verra plus loin le pourquoi de ce choix.

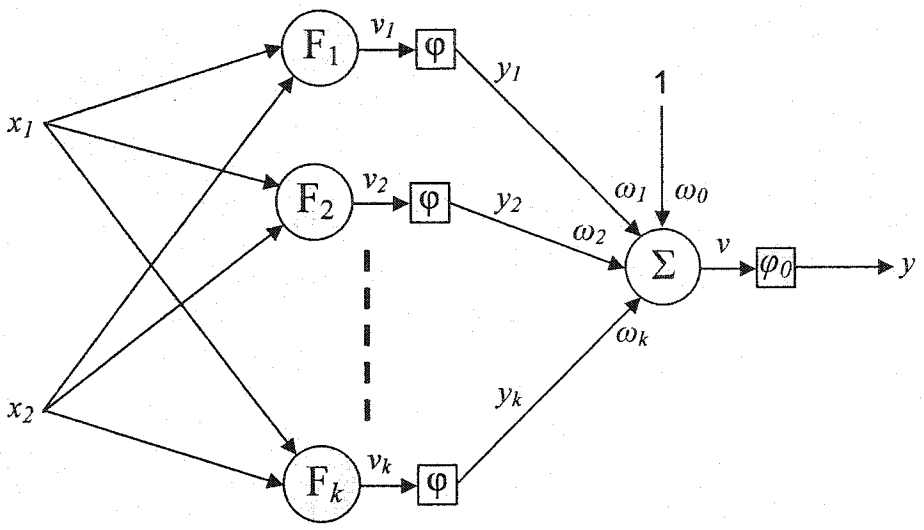


Figure 3.6: Réseau de neurones pour le « packing » et le « covering ».

où F_k est une fonction de la forme :

$$F_k = (x_1 - \omega_{1k})^2 + (x_2 - \omega_{2k})^2 - r^2 = x_1^2 + x_2^2 - 2\omega_1 x_1 - 2\omega_2 x_2 + \omega_1^2 + \omega_2^2 - r^2 \quad (3.3)$$

On remarque que la forme développée de F_k comporte 2 poids constants, ceux de x_1^2 et x_2^2. Or ceci est permissible d'après le lemme 3 à condition qu'il y ait au moins une entrée supplémentaire ajoutée au neurone. Dans ce cas on a rajouté 2 nouvelles entrées $(-2\omega_1 x_1)$ et $(-2\omega_2 x_2)$, ce qui élimine le problème de ces deux poids fixes. Les trois dernières valeurs peuvent être considérées comme le biais, dans quel cas lui aussi devient un poids qui doit être forcé à une valeur dictée par ω_1, ω_1 et r^2. Mais encore une fois, ceci est permissible d'après le lemme 1, puisqu'il stipule qu'un seul poids fixe n'entrave pas la recherche d'une solution optimale.

φ est la fonction d'activation des neurones de la première couche et elle sera de la forme d'une tangente hyperbolique de paramètre A :

$$\varphi = \tanh(Av_k) \tag{3.4}$$

Conséquemment, la sortie v sera positive (1) si le point (x_1, x_2) n'est pas couvert par le cercle de centre $(\omega_{1k}, \omega_{2k})$ et de rayon r, et négative (-1) dans le cas contraire. Ceci servira à bien différencier les deux cas, c'est-à-dire qu'on pourra se servir du signe pour inhiber ou renforcer la sortie du neurone de la deuxième couche. Donc une simple somme des sorties indiquera s'il n'y a plus de cercles qui couvre le point, ou non. Ce qui explique l'utilisation d'un simple Perceptron à la deuxième couche.

La fonction d'activation du neurone de la deuxième couche, φ_0, sera de la forme d'une sigmoïde de paramètre a :

$$\varphi_0 = \frac{1}{1+e^{(-ay)}} \tag{3.5}$$

Les poids ω_1 à ω_k seront forcés à prendre la valeur $1/k$ de façon à s'assurer que les k cercles aient la même influence pour déterminer si oui ou non un point test est couvert. Si aucun cercle ne couvre le point, alors la sortie y sera fortement positive. Dans le cas contraire, si tous les cercles couvrent le point, alors la sortie y sera fortement négative.

On rencontre toutefois un problème lorsque moins de la moitié des cercles couvrent le point, car même s'il est couvert la sortie demeure positive, ce qui implique que le point n'est pas couvert. Il faudrait donc ajouter un biais ω_0 de manière à décaler la sigmoïde vers la gauche. Puisque le nombre de cercles est connu, il devient alors simple de trouver la valeur de ω_0.

La valeur minimale que peut prendre y est :

$$Min = k(-1)\left(\frac{1}{k}\right) + \omega_0 = \omega_0 - 1 \tag{3.6}$$

La valeur maximale sera elle égale à :

$$Max = k(1)\left(\frac{1}{k}\right) + \omega_0 = \omega_0 + 1 \tag{3.7}$$

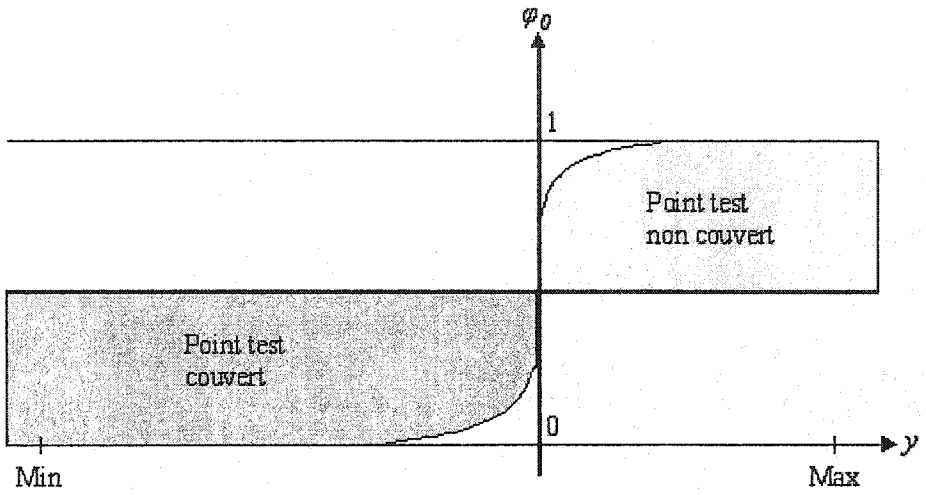


Figure 3.7: Frontière de décision à la deuxième couche.

On aura un cas limite lorsqu'un seul cercle couvrira le point test et que celui-ci sera sur sa circonférence. Si le point bascule d'un côté ou de l'autre de la circonférence du cercle, la sortie devra rester exacte. On a donc :

$$\sum_{i=1}^{k} y_i \omega_i + \omega_0 = \frac{(k-1)}{k} + \omega_0 = 0 \quad \Rightarrow \quad \omega_0 = -\frac{(k-1)}{k} \tag{3.8}$$

On remarque que tous les poids de la deuxième couche sont fixes, or ceci est permissible d'après la proposition 1. La valeur qu'on leur attribue provient de l'information à priori, celle-ci étant l'équiprobabilité d'un cercle de couvrir un point et le nombre de cercles.

Finalement, la sortie du réseau sera « 0 » si le point test n'est pas couvert par un des cercles et « 1 » sinon. Ceci implique que l'équation d'erreurs sera donc la suivante :

$$e = -y \tag{3.9}$$

Il est maintenant possible d'appliquer le réseau au problème de « packing » et de « covering ».

3.2.1 SIMULATIONS SUR LE « PACKING » D'UN CARRÉ

Comme il a été vu au début du chapitre, le « packing » d'un carré est connu pour plusieurs valeurs (Tableau 3.2), on s'est alors basé sur celles-ci lors des simulations. Ces dernières ont été faites avec les paramètres suivants, les valeurs entre parenthèses sont pour $n = [16, 20]$:

- Nombre d'itérations : 5 (10)
- Nombre de points tests par itération : un carré de 50x50 (100x100) points
- Nombre de points tests pour la validation : un carré de 100x100 points

- Constante d'apprentissage : de 1 à 0.01
- Condition d'arrêt : 100%

Voici les résultats des simulations :

Tableau 3.5: Valeurs obtenues lors de la simulation du problème de « packing » pour les valeurs de 1 à 9, 11, 16 et 20.

Nbr. de cercles	Rayon max.	Couverture max.	Couverture en simulation
1	0,5000	78,54%	77,00%
2	0,2929	53,90%	52,94%
3	0,2543	60,95%	59,84%
4	0,2500	78,54%	77,20%
5	0,2071	67,37%	66,12%
6	0,1877	66,41%	65,10%
7	0,1745	66,96%	65,51%
8	0,1705	73,06%	71,43%
9	0,1667	78,54%	77,06%
11	0,1424	70,07%	68,06%
16	0,1250	78,54%	75,32%
20	0,1114	77,97%	72,18%

Dans la figure 3.8, les dessins de droite représentent les positions optimales, tandis que les dessins de gauche représentent les positions « optimales » qui ont été trouvées par le réseau de neurones.

On remarque que le réseau parvient à trouver des solutions « optimales » pour le « packing » d'un carré avec des cercles qui sont très proches de la réalité, l'erreur étant généralement de moins de 2%. Au niveau des arrangements on remarque qu'il est difficile au réseau de trouver l'arrangement optimal lorsqu'il y a beaucoup d'objets à

placer. Une plus petite constante d'apprentissage et un plus grand nombre de points tests par itération pourraient probablement rectifier ce problème.

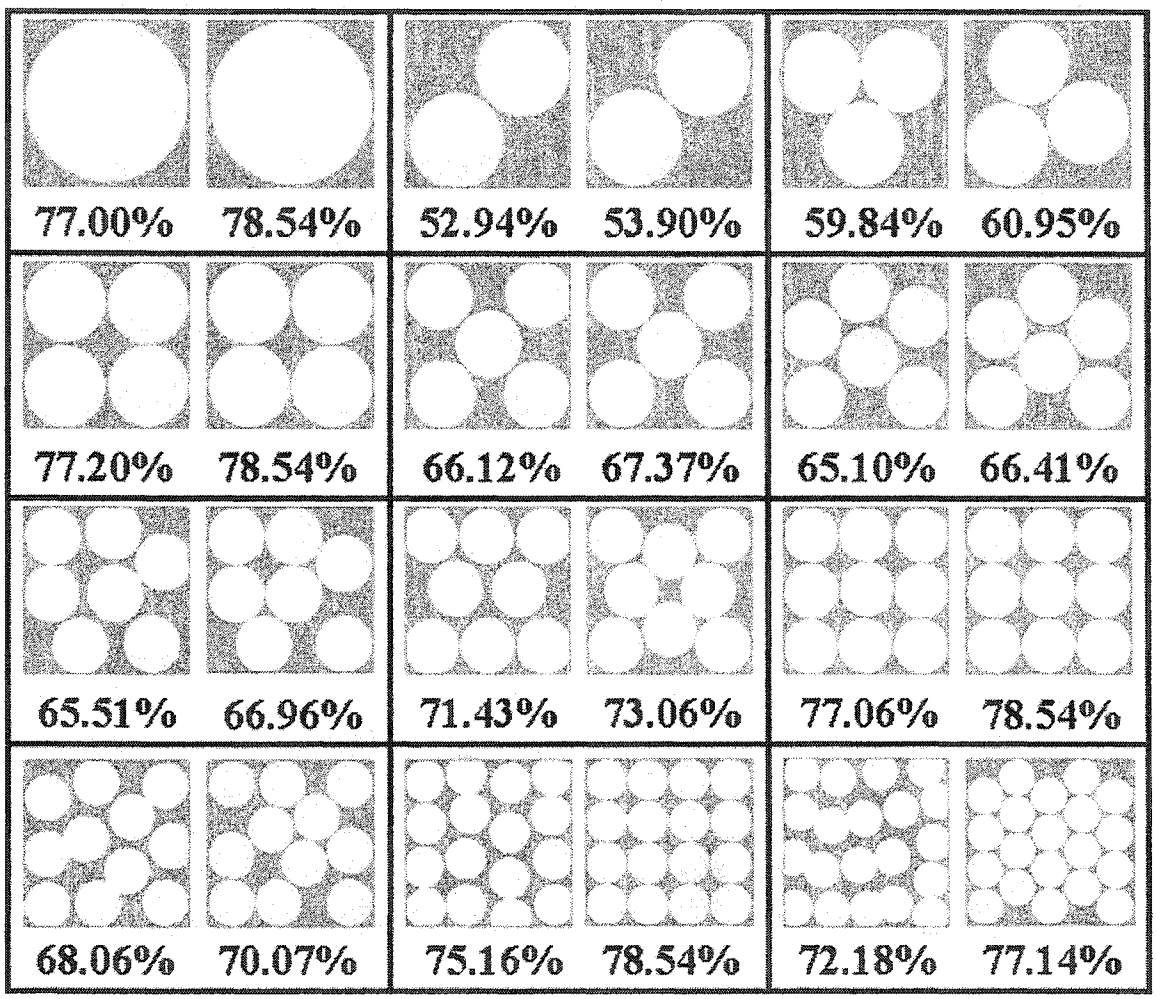


Figure 3.8: Résultats des simulations sur le « packing » pour les valeurs 1 à 9, 11, 16 et 20 d'un carré.

3.2.2 SIMULATIONS SUR LE « COVERING » D'UN CARRÉ

Comme pour le « packing », des valeurs optimales pour le « covering » ont été vues au début de ce chapitre et basées sur celles-ci plusieurs simulations ont été effectuées. Ces dernières ont été faites avec les paramètres suivants :

- Nombre d'itérations : 5
- Nombre de points tests par itération : un carré de 50x50 points
- Nombre de points tests pour la validation : un carré de 100x100 points
- Constante d'apprentissage : de 1 à 0.01
- Condition d'arrêt : 100%

Voici les résultats des simulations :

Tableau 3.6: Valeurs obtenues lors de la simulation du problème de « covering » pour les valeurs de 1 à 5 et 7.

Nbr. de cercles	Rayon min.	Couverture max.	Couverture en simulation
1	0,7071	100%	99,91%
2	0,5590	100%	99,93%
3	0,5039	100%	99,88%
4	0,3536	100%	99,94%
5	0,3262	100%	99,96%
7	0,2743	100%	99,69%

Dans la figure 3.9, les dessins de droite représentent les positions optimales, tandis que les dessins de gauche représentent les positions « optimales » qui ont été trouvées par le réseau de neurone.

On remarque que le réseau parvient à trouver des solutions « optimales » pour le « covering » d'un carré avec des cercles qui sont très proches de la réalité, l'erreur étant généralement inférieurs à 0.1%. Au niveau des arrangements on remarque que le réseau a pu trouver l'arrangement optimal dans tous les cas.

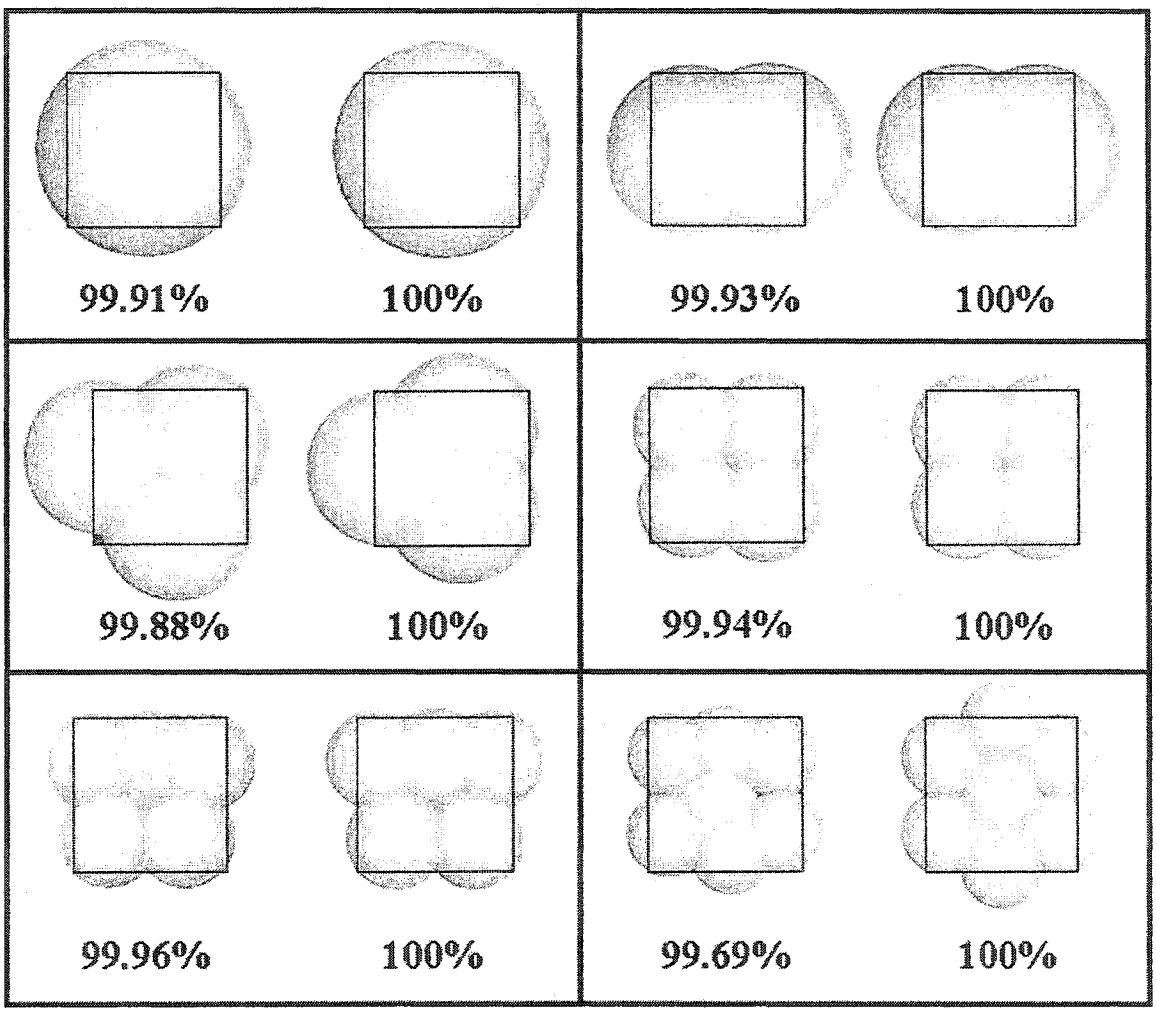


Figure 3.9: Résultats des simulations sur le covering pour les valeurs 1, 2, 3, 4, 5 et 7 d'un carré.

Le réseau pourrait donc également donner un aperçu du cas où il y aurait 6 cercles pour couvrir le carré à 100%. La figure 3.10 montre deux arrangements vers lequel le réseau

semble converger, le rayon des cercles étant 0.2917, c'est-à-dire une valeur située entre celles de n égale à 5 et 7.

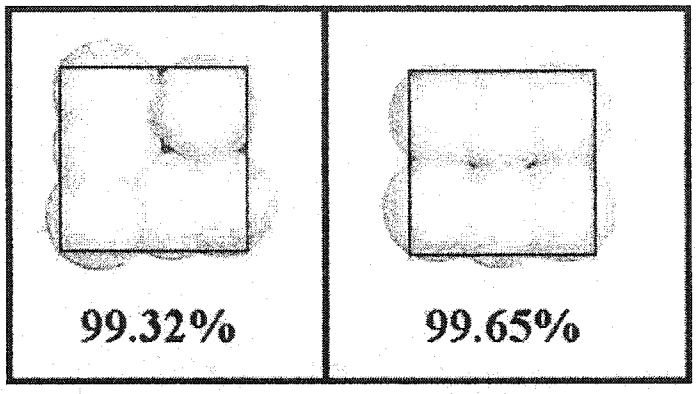


Figure 3.10: Arrangements possibles pour la couverture d'un carré unitaire avec 6 cercles.

Les simulations ont donc montré que notre réseau MLP modifié avec les lemmes 1, 2 et 3 permet de trouver des solutions optimales aux problèmes de « packing » et de « covering » avec une erreur négligeable.

3.3 AUTRES AMÉLIORATIONS ET OPTIMISATION HEURISTIQUE DE L'APPRENTISSAGE

3.3.1 PREMIERE AMELIORATION

La première amélioration consiste à changer la fonction d'activation de la deuxième couche, seulement pour le « packing », pour une sigmoïde modifiée (Équation 3.10), de paramètre a_0, multipliée par une tangente hyperbolique de paramètres a_1, a_2 et a_3.

Le « packing » a la particularité d'avoir en fait trois états : le point n'est pas couvert, le point est couvert par un seul cercle, le point est couvert par plus d'un cercle. Ce dernier ne cause pas de problème lorsqu'on travaille avec du « covering », puisqu'on permet la superposition des objets, mais dans le cas du « packing » il ne doit pas y avoir de recouvrement, c'est-à-dire il ne doit pas y avoir plusieurs cercles qui couvrent un point. C'est pour cela qu'une fonction d'activation à trois états pourrait être préférable. Si le point n'est pas couvert il y a une erreur positive, si le point est couvert par un seul cercle l'erreur est nulle et si plus d'un cercle couvre le point, l'erreur est négative. De cette façon si plus d'un cercle couvre le point lors du « packing » le réseau saura qu'il faut séparer les deux cercles. On propose donc la fonction d'activation suivante :

$$\varphi_0 = \tanh(a_0 v)\left(\frac{-1}{1+e^{a_1\left(a_2^2\left(y+\frac{r}{2}\right)^2 - a_3^2\left(\frac{r}{2}\right)^2\right)}}+1\right) \tag{3.10}$$

Dont la dérivée est la suivante :

$$\frac{\partial 7}{\partial v} = a_0\left(1-\tanh^2(a_0 v)\right)\left(\frac{-1}{1+e^{a_1\left(a_2^2\left(v+\frac{r}{2}\right)^2 - a_3^2\left(\frac{r}{2}\right)^2\right)}}+1\right) + \frac{2a_1a_2^2\tanh(a_0 v)\left(v+\frac{r}{2}\right)e^{a_1\left(a_2^2\left(v+\frac{r}{2}\right)^2 - a_3^2\left(\frac{r}{2}\right)^2\right)}}{\left(1+e^{a_1\left(a_2^2\left(v+\frac{r}{2}\right)^2 - a_3^2\left(\frac{r}{2}\right)^2\right)}\right)^2} \tag{3.11}$$

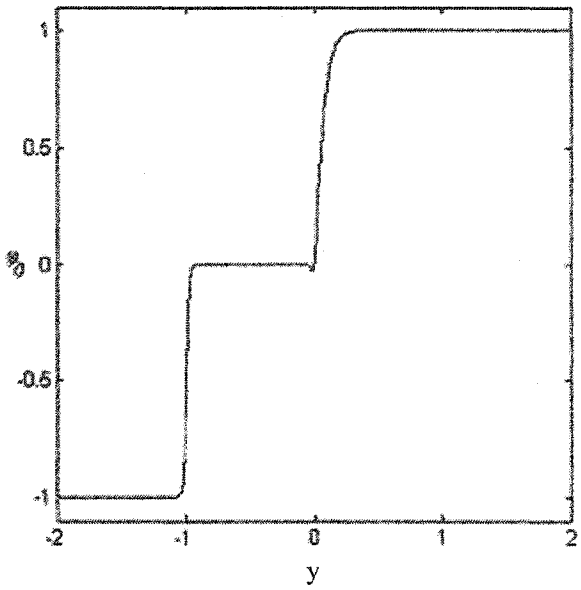


Figure 3.11: Forme de la fonction d'activation à trois états distincts.

La figure 3.11 illustre graphiquement cette fonction d'activation pour un cercle de rayon 1; les paramètres utilisés ont été $a_0 = 10$, $a_1 = 1$, $a_2 = 10$, $a_3 = 10$ et $r = 1$.

3.3.2 DEUXIEME AMELIORATION

La deuxième amélioration serait de mieux contrôler les paramètres a et A des fonctions d'activation. Regardons d'abord la variation de $\Delta\omega$ par rapport à la distance entre le point test et le centre du cercle. On a :

$$\Delta\omega_{ij} = \eta \frac{\partial E}{\partial \omega_{ij}} \qquad \text{où} \qquad \frac{\partial \mathrm{E}}{\partial \omega_{ij}} = \frac{\partial \mathrm{E}}{\partial e}\frac{\partial e}{\partial v}\frac{\partial v}{\partial y}\frac{\partial y}{\partial v_i}\frac{\partial v_i}{\partial y_i}\frac{\partial y_i}{\partial \omega_{ij}} \qquad (3.12)$$

Tableau 3.7:Résumé des équations utilisées lors des étapes 3 et 4 de l'algorithme d'apprentissage.

Équation	Dérivée Partielle	Valeurs [Min,Max]
$v_i = (x_1 - \omega_1)^2 + (x_2 - \omega_2)^2 - r^2$	$\frac{\partial v_i}{\partial \omega_j} = -2(x_j - \omega_j)$	$[-2D_{\max}, 2D_{\max}]$
$y_i = \tanh(-Av_i)$	$\frac{\partial y_i}{\partial v_i} = A(1 - \tanh^{-1}(Av_i))$	[0, A]
$v = \sum \frac{y_i}{k}$	$\frac{\partial v}{\partial y_i} = \frac{1}{k}$	$1/k$
$y = \frac{1}{1 + \exp^{-av}}$	$\frac{\partial y}{\partial v} = \frac{a \exp^{-av}}{(1 + \exp^{-av})^2}$	$\left[0, \frac{a}{4}\right]$
$e = -y$	$\frac{\partial e}{\partial y} = -1$	-1
$E = \frac{1}{2}e^2$	$\frac{\partial E}{\partial e} = e$	[0, 1]
$\frac{\partial E}{\partial \omega_j} = \frac{\partial E}{\partial e}\frac{\partial e}{\partial y}\frac{\partial y}{\partial v}\frac{\partial v}{\partial y_i}\frac{\partial y_i}{\partial v_i}\frac{\partial v_i}{\partial \omega_j}$	$\frac{\partial E}{\partial \omega_j} = \frac{2ea\exp^{-av} A(1 - \tanh^{-1}(Av_i))(x_j - \omega_j)}{k(1 + \exp^{-av})}$	$\left[\frac{-AaD_{\max}}{2k}, \frac{AaD_{\max}}{2k}\right]$

Le tableau 3.7 montre les équations et leurs limites utilisées lors des étapes 3 et 4 de l'algorithme d'apprentissage. Pour des raisons de simplicité on se limitera ici à une fonction d'activation sigmoïdale pour la deuxième couche, dans le cas où la fonction d'activation à trois états était utilisée il faudrait simplement recalculer la dernière dérivée avec sa valeur minimum et maximale.

Regardons de plus près la variation finale de $\partial E/\partial \omega$, celui-ci pouvant être représenté comme étant le produit de MG et MD :

$$MG = \frac{2ea\exp^{-av} A\left(1 - \tanh^2(Av_i)\right)}{k\left(1 + \exp^{-av}\right)^2} \tag{3.13}$$

$$MD = (x_1 - \omega_1) \tag{3.14}$$

Le membre de droite (*MD*) représente la distance entre le point test et le centre du cercle, et de ce fait quatre cas se présentent :

1. *MD* est négatif et le point est à l'extérieur d'un des cercles (|*MD*| – Rayon > 0).

2. *MD* est négatif et le point est à l'intérieur d'un des cercles (|*MD*| – Rayon < 0).

3. *MD* est positif et le point est à l'extérieur d'un des cercles (|*MD*| – Rayon > 0).

4. *MD* est positif et le point est à l'intérieur d'un des cercles (|*MD*| – Rayon < 0).

Dans les cas où le point test est à l'intérieur d'un des cercles, la valeur de l'erreur est nulle (ou presque); $\partial E/\partial \omega$ est nulle, donc il peut être négligé. Dans les cas contraires le cercle doit être déplacé afin de couvrir le point, le signe de *MD* indiquant le sens. Mais quelle doit être la distance de ce déplacement?

On sait que la valeur de |*MD*| indique la distance entre le centre du cercle et le point, mais cette distance est plus grande que le minimum requis. On doit également noter qu'un cercle donné ne connaît pas la position des autres cercles, on doit donc considérer les deux cas possibles :

1. le cercle en question est le cercle le plus près.

2. le cercle en question n'est pas le cercle le plus près.

Sachant qu'il y aura plusieurs itérations, il n'est pas nécessaire que le point soit couvert lors de la première itération, mais qu'il y ait au moins un cercle qui s'en approche. Puisqu'un cercle donné ignore la position des autres cercles et que le point doit être couvert lors d'une itération future, il en résulte que chaque cercle doit considérer qu'il est le plus près et doit donc s'en approcher un petit peu à chaque itération jusqu'à ce qu'il soit couvert par un des cercles.

On veut également que le déplacement du cercle le plus près soit la plus grande, relativement à sa distance initiale. C'est-à-dire que le déplacement d'un cercle soit en fonction de sa distance du point test, d'où le membre gauche (*MG*) de l'équation. La figure 3.12 montre la forme de *MG* pour les valeurs suivantes:

$$e = 1 \qquad A = 10 \qquad a = 10 \qquad k = 1$$

Regardons de plus près la forme de la fonction *MG*. Notons tout d'abord que l'axe des y représente la couverture du point par tous les cercles et l'axe des v_i représente la couverture du point par un cercle en particulier.

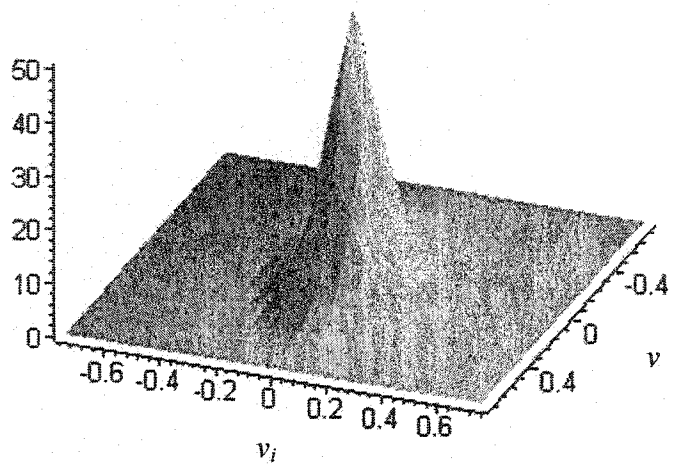


Figure 3.12: Forme de la fonction *MG*.

Puisque l'axe des v représente la couverture de tous les cercles et qu'un v négatif implique que le cercle est couvert ($e = 0 \Rightarrow MG = 0,\ \forall v_i$), alors on peut déjà éliminer toute la partie négative de l'axe des v. Il faut noter que, puisqu'on utilise une sigmoïde, il est possible d'avoir une valeur non nulle pour e même si le point est couvert. Dans ce cas la partie négative où *MG* ne serait pas nulle pourrait être vue comme un renforcement de l'espace déjà couvert. En ce qui concerne la partie positive, on remarque que plus la valeur de v est élevée plus la valeur de *MG* diminue. Cela sert à éviter l'apprentissage de points erronés.

L'axe des v_i représente quant à lui la distance d'un cercle par rapport au point test. Plus particulièrement, la partie positive représente un point test à l'extérieur du cercle avec la valeur de *MG* représentant l'importance de l'apprentissage du point non couvert, tandis que la partie négative représente un point test à l'intérieur du cercle avec une valeur de

MG relative au renforcement nécessaire de l'espace autour du point test. On remarque que dans les deux cas, plus on s'éloigne de l'origine, plus la valeur de *MG* diminue. Dans le cas où v_i est négatif, cette diminution est due au fait que le point étant de plus en plus près de l'origine du cercle, le renforcement est donc moins important. Dans le cas où v_i est positif, cette diminution représente l'importance de couvrir ou non le point de façon à éviter l'apprentissage de points erronés.

Cette composition des deux axes montre bien le partage que le réseau de neurones fait de l'erreur, il y a une erreur globale représentée par l'axe des v et une erreur locale représentée par v_i.

Finalement, il est à noter que la variation de y est plus grande que celle de v_i, et que la valeur maximale est $(Aa)/(2k)$.

MD représente l'erreur totale, il est donc normal que pour un cercle donné le déplacement soit proportionnel à cette erreur, c'est-à-dire que $\partial E/\partial\omega$ soit une proportion *MG* de *MD*. Il ne reste qu'à rajouter la constante d'apprentissage η qui permet un meilleur contrôle du déplacement, ce qui permet de raffiner l'apprentissage.

On peut donc réécrire l'équation (3.12) :

$$\Delta\omega = \eta MG * MD \tag{3.15}$$

Où ηMG représente le pourcentage P de la distance sur laquelle le centre du cercle doit se déplacer pour rejoindre le point test. On peut alors définir la valeur de η :

$$\eta = \frac{P}{MG} \tag{3.16}$$

Donc pour un déplacement maximal voulu, c'est-à-dire un P_{max} donné, on a :

$$\eta = \frac{2kP_{\max}}{Aa} \tag{3.17}$$

Il suffit de modifier P_{max} pour obtenir la valeur de η nécessaire pour avoir un apprentissage plus ou moins raffiné.

3.3.3 TROISIEME AMELIORATION

La troisième amélioration serait d'augmenter la dimension du problème. Il serait possible d'extrapoler le réseau pour le « packing » et « covering » à plus que deux dimensions. Dans le cas où on rajouterait seulement une dimension, c'est-à-dire qu'on passerait en trois dimensions, il suffit de rajouter une entrée pour représenter cette nouvelle dimension. L'équation (3.3) deviendrait la suivante :

$$F = \left(x_1 - \frac{\omega_1}{2}\right)^2 + \left(x_2 - \frac{\omega_2}{2}\right)^2 + \left(x_3 - \frac{\omega_3}{2}\right)^2 - r^2 \tag{3.18}$$

3.3.4 OPTIMISATION HEURISTIQUE

Voyons de plus près ce que le réseau fait. Dans la figure 3.13, le réseau doit couvrir un rectangle. On remarque que certains cercles se chevauchent et d'autres, possiblement les mêmes, dépassent les limites du rectangle.

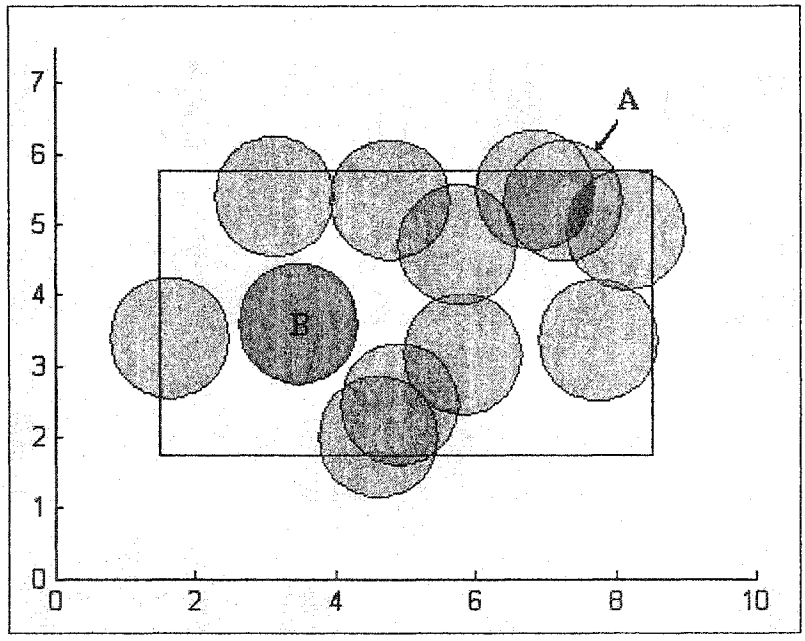


Figure 3.13: Exemple d'un cercle inutile (A) et d'un cercle utile qui devrait être rajouté.

Le réseau, d'après l'équation (3.15), bouge les cercles vers les points tests proportionnellement à la distance entre son centre et celui-ci. Or, il est évident que ce n'est pas très optimal. Le cercle A à la figure 3.13 sert très peu, il serait préférable de le

bouger directement à la position du cercle B que de bouger tous les cercles d'une petite fraction de la distance. Cela ne veut pas dire qu'il est inutile de bouger les cercles en fonction de la distance qui les séparent du point test, c'est plutôt une combinaison des deux méthodes qui serait optimal : des petits déplacements pour raffiner la couverture et des sauts pour l'accélérer et pour éviter d'avoir des objets qui ne servent à rien. Il serait également intéressant d'ajouter des cercles au fur et à mesure que le besoin se fait sentir, puisqu'il est parfois impossible de savoir combien il en faudrait.

3.3.4.1 L'AJOUT D'OBJETS

L'ajout d'un objet, c'est-à-dire d'un neurone, est assez simple. Lorsque le réseau reçoit un point test qui n'est pas couvert par les objets déjà existants, il est alors possible d'accélérer le recouvrement en ajoutant un objet à la position du point test. Mais, le réseau ne doit pas ajouter un objet si celui-ci est par la suite rejeté, car il ne couvre pas suffisamment de surface.

Donc, il faut non seulement que le point test soit non couvert, mais il faut également qu'il soit suffisamment loin des autres objets déjà présents. Lorsque le réseau reçoit un point test et que celui-ci n'est pas couvert, il devra, avant de rajouter un objet, vérifier qu'il n'est pas trop près d'un autre objet. Si les conditions sont remplies, alors le réseau pourra rajouter un objet, le centre de celui-ci étant le point test. Le cercle B représente

cet ajout dans la figure 3.13. On a choisi le centre, pour ne pas avantager une région par rapport à une autre, puisque la surface à couvrir est inconnue du réseau.

3.3.4.2 La suppression d'objets

La suppression d'un objet est un peu plus difficile que l'ajout. Le problème réside dans les conditions pour lesquelles un objet doit être supprimé. Plusieurs options peuvent être satisfaisantes.

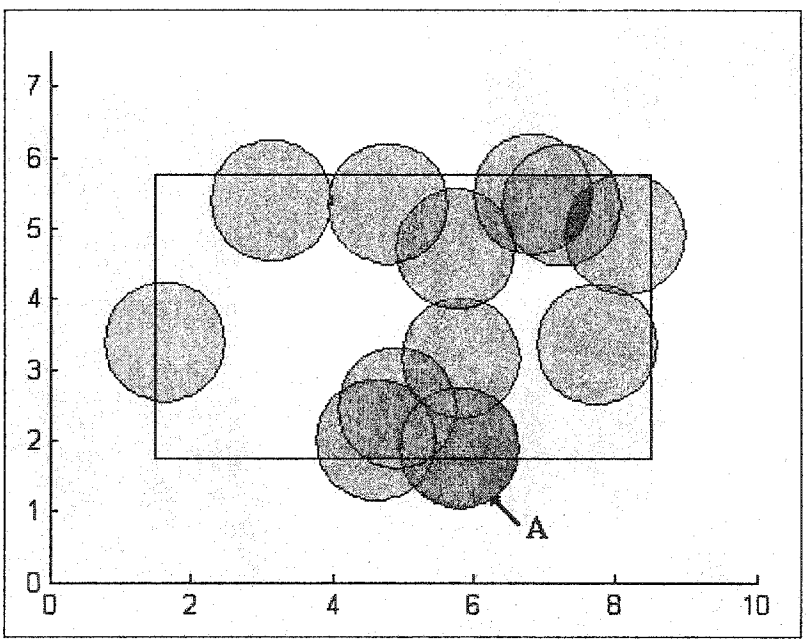


Figure 3.14: Exemple d'un cercle que l'on voudrait supprimer.

La figure 3.14 montre les recouvrements d'une couleur plus foncée, le cercle A représente celui qu'on veut supprimer. Il faut toutefois quantifier ce recouvrement.

Une option serait de calculer analytiquement l'aire de la lunule que deux cercles se partage. La figure 3.15 montre la surface commune de deux cercles, l'aire de la surface hachurée sera égale à 4 fois l'aire de la surface ombragée, on n'a donc qu'à calculer l'aire de cette dernière.

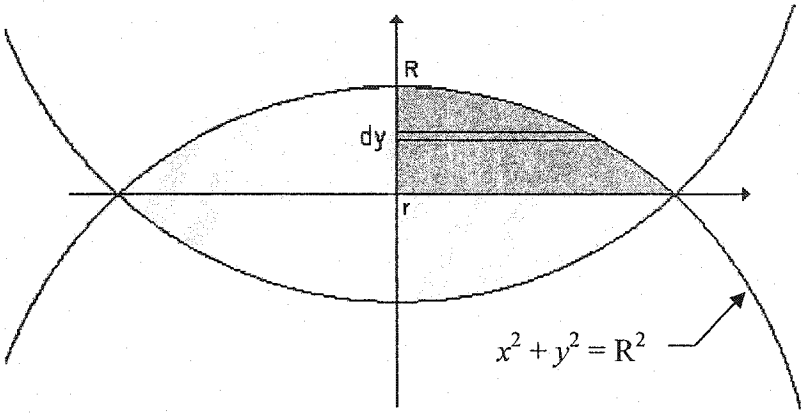


Figure 3.15: Lunule formée par deux cercles

$$AO = \int_{r}^{R} \sqrt{R^2 - y^2}\, dy = \frac{\pi R^2}{4} - \frac{1}{2}\left(r\sqrt{R^2 - r^2} + R^2 \sin^{-1}\left(\frac{r}{R}\right)\right) \tag{3.19}$$

$$AH = 4 * AO = \pi R^2 - 2\left(r\sqrt{R^2 - r^2} + R^2 \sin^{-1}\left(\frac{r}{R}\right)\right) \tag{3.20}$$

où AO est l'aire ombragée et AH est l'aire hachurée.

Pour simplifier les calculs, et puisque l'apprentissage se fait de façon approximative, on ne calculera pas les surfaces triplement ou plus recouvertes. C'est-à-dire, si une surface

est recouverte de trois cercles, celle-ci sera évaluée deux fois. La condition doit être maintenant implantée dans l'algorithme d'apprentissage.

On veut supprimer un objet qui ne respecte pas la (ou les) condition(s), c'est-à-dire qu'on veut que le lien entre le neurone de cet objet et la deuxième couche soit coupé. Puisqu'il n'est plus présent il ne doit pas affecter le résultat de la couche qui suit. Ceci peut se résumer sous forme mathématique comme suit :

$$v_k = \begin{cases} \left(x_1 - \frac{\omega_{1k}}{2}\right)^2 + \left(x_2 - \frac{\omega_{2k}}{2}\right)^2 - r^2 & \text{si} \quad TAH \leq \Gamma \\ \text{rien} & \text{sinon} \end{cases} \tag{3.21}$$

où Γ est le seuil du pourcentage de recouvrement qu'un cercle peut avoir et TAH le pourcentage de recouvrement du cercle k par d'autres.

On a encore une fois changé la sortie du neurone, au lieu d'avoir une fonction en sortie, on a une condition. On doit donc vérifier l'effet de ce changement sur l'algorithme d'apprentissage. Si le lien est conservé, le réseau reste inchangé, conséquemment l'algorithme reste inchangé. Dans le cas contraire, si le lien est coupé *y* égal rien, c'est-à-dire que la (ou les) neurone(s) en aval ne tiendrait(ent) pas compte de ce neurone, celui-ci n'affecte donc pas la sortie du réseau. Il n'affecte donc pas l'erreur directement, ce qui

n'implique aucun changement lors de l'algorithme de rétro-propagation. Mais, indirectement son absence diminue la probabilité qu'un point à l'intérieur de sa circonférence soit couvert. Il en résulte qu'il doit être possible d'affecter une erreur à un objet supprimé de façon à ce qu'il ne le soit plus. Or l'erreur du réseau n'est pas directement liée à l'objet supprimé, il faut donc trouver une autre erreur basée sur une information directement liée à celui-ci. On sait que l'objet fut supprimé car il ne remplissait pas la (ou les) condition(s), il semble donc naturel que cet échec soit l'erreur recherchée. Il faut maintenant utiliser cette erreur pour modifier les poids du neurone supprimé pour qu'il ne le soit plus.

Une fois le neurone détaché du réseau, ce qui constitue un échec à la (ou aux) condition(s), il devient lui-même un nouveau réseau. Pour quantifier cette erreur il suffit de modifier l'équation (3.21) de la manière suivante :

$$v_k = \begin{cases} \left(x_1 - \frac{\omega_{1k}}{2}\right)^2 + \left(x_2 - \frac{\omega_{2k}}{2}\right)^2 - r^2 & \text{si} \quad \text{sigmoïde}(TAH - \Gamma) \leq \frac{1}{2} \\ \text{rien} & \text{si} \quad \text{sigmoïde}(TAH - \Gamma) > \frac{1}{2} \end{cases} \tag{3.22}$$

On peut maintenant appliquer les règles de l'algorithme de rétro-propagation pour la correction des poids du neurone qui s'est détaché, c'est-à-dire celui dont le lien s'est coupé avec le reste du réseau en aval, en utilisant la condition elle même, l'équation

(3.22). Dans le cas contraire, le neurone reste attaché au réseau initial et sa correction dépend de la rétro-propagation de l'erreur de ce même réseau initial.

Une variante pourrait aussi être utilisée. Posons que les deux cercles problématiques de la figure 3.16 ne remplissent pas la (ou les) condition(s).

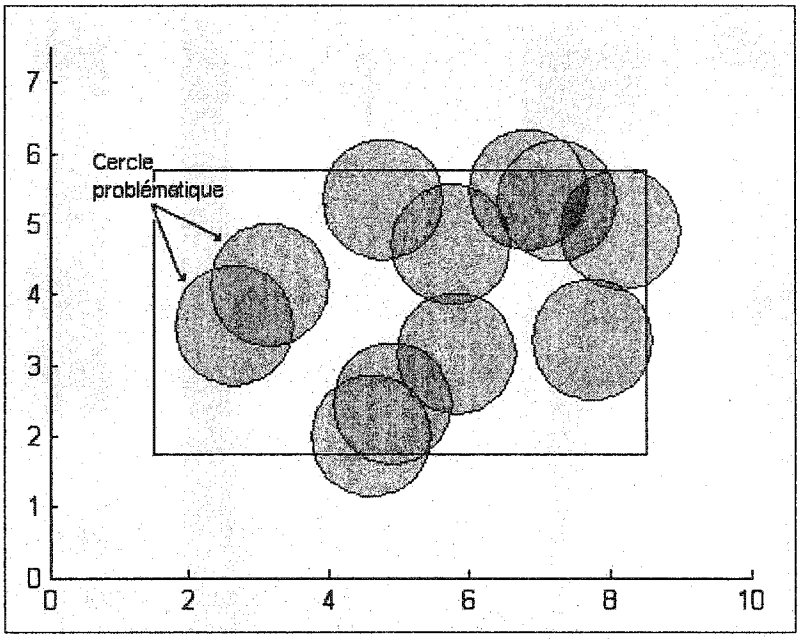


Figure 3.16: Exemple de deux cercles problématiques.

Si le problème devait être résolu par un humain, celui-ci garderait probablement au moins un cercle, mais il aurait de la difficulté à choisir lequel. Quatre cas sont possibles : les deux sont supprimés, les deux sont conservés, un des deux est conservé et l'autre est supprimé et vice versa. Chacun de ces quatre cas peut devenir la solution optimale, ou pas. Une solution serait d'assigner à chacun une probabilité P, qui déterminera sa chance d'être sélectionnée. Cette probabilité devra reposer sur l'information connue, c'est-à-dire

la surface du cercle et l'état du recouvrement de celui-ci, puisque la (ou les) condition(s) sont représentées par une sigmoïde (2.2) la transformation devient simple.

$$v_k = \begin{cases} \left(x_1 - \frac{\omega_{1k}}{2}\right)^2 + \left(x_2 - \frac{\omega_{2k}}{2}\right)^2 - r^2 & \text{avec} \quad P = 1 - \text{sigmoïde}(TAH - \Gamma) \\ \text{rien} & \text{avec} \quad P = \text{sigmoïde}(TAH - \Gamma) \end{cases} \tag{3.23}$$

Pour le moment la probabilité dépend de l'aire que le cercle couvre à lui seul et d'un seuil Γ. Ce seuil a un impact important sur la décision prise par le neurone, or il est déterminé sans aucune base mathématique. Il faudrait alors retirer ce seuil de façon à ce que les cercles se gèrent tout seuls.

$$v_k = \begin{cases} \left(x_1 - \frac{\omega_{1k}}{2}\right)^2 + \left(x_2 - \frac{\omega_{2k}}{2}\right)^2 - r^2 & \text{avec} \quad P = 1 - TAH \\ \text{rien} & \text{avec} \quad P = TAH \end{cases} \tag{3.24}$$

Donc plus un cercle est couvert par d'autres cercles, plus la probabilité qu'il soit supprimé augmente. Mais, cette probabilité ne prend pas en considération combien de cercles couvrent cette surface et de quelle façon ils le font.

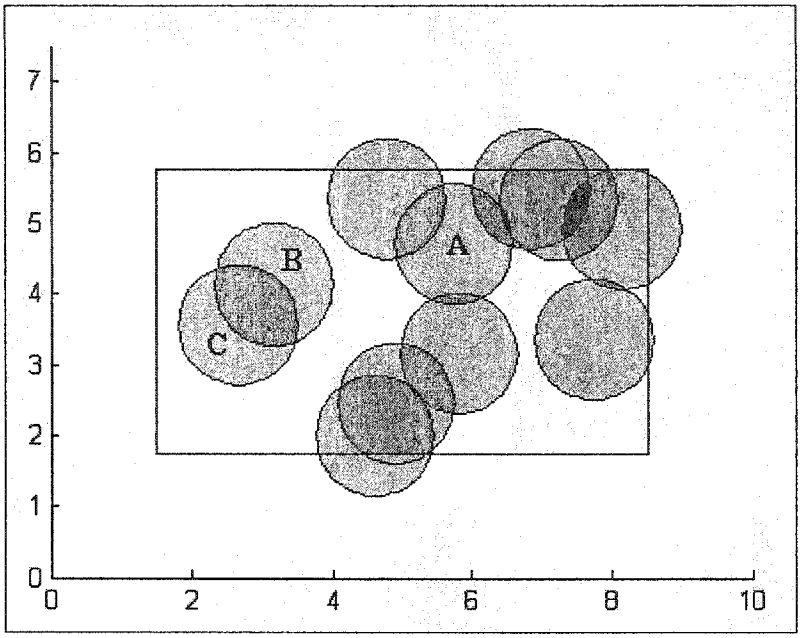


Figure 3.17: Exemple d'un problème où plusieurs cercles pourraient être supprimés (A, B ou C).

Posons (figure 3.17) comme hypothèse que le cercle A couvre la même surface que les cercles B et C. Ils ont alors la même probabilité d'être supprimés, or le cercle A semble plus utile que les deux autres. En effet, la disparition d'un seul des trois cercles qui le couvre diminuerait sa probabilité d'être supprimé, tandis qu'il faut qu'un des deux cercles dans l'autre cas soit supprimé pour que la probabilité que l'autre soit supprimé diminue. Il semble donc naturel que la probabilité que le cercle A soit supprimé est plus petite que celles des deux autres.

On remarque également que le cercle C est plus loin des autres cercles que le cercle B. Encore une fois, il semble plus naturel que le cercle C ait une plus petite probabilité

d'être supprimé que le cercle B. Donc, la distance des cercles entre eux devrait avoir un impact sur la probabilité qu'un cercle soit supprimé.

La suppression d'objet est un problème complexe, qui nécessite une étude plus approfondie, mais elle semble nécessaire afin d'éviter certain minimum local. Il est également important de noter que toutes les améliorations qui ont été vue doivent être appliquées en fonction du problème, chaque cas ayant ses propres particularités.

3.4 CONCLUSIONS

Il a été démontré, dans le lemme 1, qu'un neurone peut avoir au plus un seul poids constant sans restreindre le processus d'apprentissage. Il ne peut cependant être forcé à la valeur 0 en théorie, par contre en pratique ceci ne cause aucun problème puisqu'il est toujours possible de trouver une droite qui sépare correctement les deux classes (lemme 2). Dans les cas où plusieurs valeurs doivent être fixées à des constantes il sera également possible au réseau de trouver une solution qui satisfait le problème à condition de lui rajouter un nombre suffisant d'entrées supplémentaires, comme il a été défini au lemme 3. Une autre option serait de rajouter une couche, et comme il a été montré à la proposition 1, ceci peut avoir deux effets : le premier a pour effet de laisser le réseau trouver l'hyperplan qui sépare correctement les deux classes; le deuxième effectue une transformation spatiale sur les entrées de façon à ce que l'hyperplan fixe de la deuxième couche puisse séparer correctement les deux classes.

Ces modifications permettent de simplifier la structure du Perceptron, qui à son tour permet une plus grande variété de frontières, donc un plus grand contrôle du réseau de neurones. Ce contrôle peut se traduire au niveau du nombre de couches, du nombre de neurones par couche ou encore au niveau des entrées et des poids.

L'application de ces lemmes permet de résoudre, avec de très bons résultats, les problèmes de « packing », l'erreur étant généralement de moins de 2%, et de « covering », l'erreur étant généralement inférieur à 1%. Dans les deux cas le réseau a été capable de trouver la solution optimale pour des cercles à l'intérieur d'un carré. Le réseau a également été capable de servir à trouver des solutions différentes pour un problème aléatoire; comme il a été montré dans l'exemple du « covering » d'un carré unitaire avec 6 cercles, où l'erreur n'a été que de quelques dixièmes de pourcent.

Plusieurs améliorations peuvent être apporté au réseau, tel l'ajout et la suppression d'objets, ceux-ci permettant au réseau de contrôler lui-même sa structure de façon à résoudre le problème, c'est-à-dire qu'il peut rajouter des neurones ou encore contrôler l'état (ouvert ou fermé) des liens entre les neurones qui le compose. D'autres améliorations telles que la fonction d'activation à trois niveaux (-1, 0, 1) et une règle qui permet de contrôler la distance maximale qu'un objet peut se déplacer peuvent également servir à augmenter la performance du réseau.

Il reste cependant des avenues à explorer pour améliorer l'application des réseaux de neurones aux problèmes de « packing » et « covering », principalement au niveau de la surface à couvrir (aléatoire, 3D, etc.) et de la forme des objets qui peuvent être utilisées pour couvrir la surface (triangle, carré, hexagone, etc.). Il y a également des améliorations à apporter au niveau des décisions que le réseau doit prendre sur l'ajout, la suppression et le recouvrement des objets.

On peut alors dire que notre réseau de neurones convient bien aux problèmes de « packing » et de « covering », il réussit à trouver une solution optimale avec un erreur négligeable.

BIBLIOGRAPHIE

[1] ASAI H., NAKAYAMA T. et NINOMIYA H., Tiling Algorithm with Fitting Violation Function for Analog Neural Array, *Neural Networks, IEEE International Conference on,* Volume: 1, 3-6 Jun (1996)., vol. 1, 565-570.

[2] CONWAY J.H. et SLOAN N.J.A., Sphere Packing, Lattices and Groups, *second edition, Springer-Verlag, New-York,* (1993).

[3] FLAKE G. W., Square Unit Augmented, Radially Extended, Multilayer Percepttons, *in Neural Networks: Tricks of the Trade (G. B. Orr & K.-R. Milllet, eds.), Berlin: Springer Verlag,* (1998)., vol. 1524 of Lecture Notes in Computer Science, 145-163.

[4] HALES T. C., An Overview of the Kepler Conjecture, *Manuscrits,* (1998).

[5] HAYKIN S., Neural Networks : a Comprehensive Foundation, *second edition, Prentice Hall, Inc., New Jersey,* (1999).

[6] JACOB E.G., JOSEPH O'R., Handbook of Discrete and Computational Geometry, *CRc Press LLC,* (1997).

[7] KAMIURA N., TANIGUCHI Y. et MATSUI N., A Functional Manipulation for Improving Tolerance Against Multiple-Valued Weight Faults of Feedforward Neural Networks, *Multiple-Valued Logic, 2001 Proceedings 31st IEEE International Symposium on,* (2001)., 339-344.

[8] KOLEAN J. F. et HEWETT R., Prediction of Lake Inflows with Neural Networks, *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, Volume: 1, (2000)., vol. 1, 572-576.

[9] LEE Y.C., DOOLEN G., CHEN H.H., SUN G.Z., MAXWELL T., LEE H.Y. et GILES C.L., Machine Learning Using Higher Order Correlation Networks, *Physica D*, (1986)., 22-D :276-306.

[10] MANABE S. et ASAI H., A Neuro-Based Optimization Algorithm for Tiling Problems with Rotation, *Neural Networks, 1999, IJCNN '99, International Joint Conference on*, Volume: 6, Jul (1999)., vol. 6, 3750-3753.

[11] McCULLOCH W.S. et Pitts W., A Logical Calculus of the Ideas Immanent in Nervous Activity, *Bulletin of Mathematical Biophysics*, (1943)., vol. 5, 115-133.

[12] PAO Y.H., Adaptive Pattern Recognition and Neural Networks, *Addison-Wesley Publishing Company, Inc., Reading, Massachussettes*, (1989).

[13] PEREZ C.A., GONZALEZ G.D., SALINAS C., Genetic Selection of Non-Linear Product Terms in the Inputs to a Linear Classifier for Handwritten Digit Recognition, *Systems Man and Cybernetics, 2001 IEEE International Conference on*, Volume: 4, (2001)., vol. 4, 2337-2342.

[14] ROGERS C.A., Packing and Covering, *Cambridge University Press*, (1964).

[15] ROSENBLATT F., The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, *Psychological Review*, (1958)., vol. 65, 386-408.

[16] RUMELHART D.E., HINTON G.E. et WILLIAMS R.J., Learning Representations of Back-Propagation Errors, Nature (London), (1986a)., vol. 323, 533-536.

[17] RUMELHART D.E., McCLELLAND J.L. et le PDP Research Group, Parallel Distributed Processing: Exploration in the Microstructure of Cognition, *MIT Press.*, (1986)., 2 vols.

[18] TAKEFUJI Y. et LEE K.-C., A Parallel Algorithm for Tiling Problems, *Neural Networks, IEEE Transactions on*, Volume: 1, Issue: 1, Mar (1990)., 143 -145.

[19] TURING A.M., Computing Machinery and Intelligence, Mind, (1950)., vol. 59, 433-460.

[20] YAMAMOTO H., NAKAYAMA T., NINOMIYA H., ASAI H., Application of Neuro-Based Optimization Algorithm to Three Dimensional Cylindric Puzzles, *Neural Networks, 1997, International Conference on*, Volume: 2, 9-12 Jun (1997)., vol. 2, 1246-1250.

[21] ZONG C., Sphere Packing, *Springer-Verlag, New-York*, (1999).