N° d'ordre :11/2005-M/MT

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari - Boumediene

U.S.T.H.B / ALGER

Faculté des Sciences Mathématiques

MEMOIRE

Présenté en vue de l'obtention du diplôme de Magister

en : MATHEMATIQUES

Spécialité : PROBABILITES ET STATISTIQUES

Par : TALBI RAFIKA

Sujet :

LE PRECEPTRON MULTICOUCHES APPROXIMATEUR UNIVERSEL

Soutenu publiquement le17 mars 2005, devant le jury composé de :

M.A. BERRACHEDI , *Professeur U.S.T.H.B*	Président
M.M. DJEDOUR, *Professeur U.S.T.H.B*	Directeur de thèse
M.A. AISSANI, *Professeur U.S.T.H.B*	Examinateur
M.O.ANES, *Maître de conférences I.N.P.S*	Examinateur
M.T.LARDJANE, *Chargé de cours U.S.T.H.B*	Examinateur

Table des matières

Liste des figures

INTRODUCTION

Au cours de ces dernières années, l'une des évolutions les plus marquantes des réseaux de neurones formels a été, pour les ingénieurs, l'abandon de la métaphore biologique au profit des fondements théoriques solides dans le domaine des statistiques : On sait à présent que la propriété fondamentale des réseaux de neurones est l'approximation universelle parcimonieuse.

Depuis leur découverte, les réseaux de neurones n'avaient pas de fondements mathématiques mais grâce à *Kolmogorov* [2], ils ont fait une avancée remarquable.

Le théorème de *Kolmogorov* [2] tient ses origines de la fin du $19^{ème}$ siècle où le mathématicien *Hilbert* [3] dressa une liste de 23 problèmes non résolus, et qui devaient représenter un *challenge* pour les mathématiciens du $20^{ème}$ siècle.

Le treizième problème d'*Hilbert* traite de la possibilité de représenter des fonctions à plusieurs variables en termes de superpositions de fonctions ayant un nombre minimum de variables. La conjecture d'*Hilbert* était que : "*Il existe des fonctions continues de trois variables qui ne peuvent pas être représentées par la superposition de fonctions de deux variables* ". La conjecture a été désapprouvée par *Kolmogorov* en 1957 [2], qui montra alors que toute fonction continue à plusieurs variables (dans un domaine fermé et borné) peut être représentée comme la superposition par un petit nombre de fonctions d'une seule variable. Des versions améliorées du théorème de *Kolmogorov* ont été données par *Sprecher* (1965) [4].

En termes de réseaux de neurones, ce théorème dit que toute fonction continue $f(x)$ à n entrées x_i et à une sortie y peut être représentée exactement par un réseau à trois couches ayant $n(2n+1)$ cellules dans la première couche et $(2n+1)$ cellules dans la seconde couche cachée. Chaque cellule dans la première couche cachée calcule une fonction d'une des variables d'entrée x_i donnée par $\Phi_j(x_i)$ où $j = 1, ..., 2n+1$ et où les Φ_j sont des fonctions strictement monotones.

L'activation de la $j^{ème}$ unité dans la seconde couche cachée est donnée par :

$$Z_j = \sum_{i=1}^{n} \lambda_i \Phi_j(x_i) \qquad \text{où } 0 < \lambda_i < 1 \text{ sont des constantes}$$

La sortie y du réseau est donnée par :

$$y = \sum_{j=1}^{2n+1} g(Z_j) \qquad \text{où la fonction } g \text{ est réelle et continue}$$

L'objectif principal de ce travail est donc l'étude du Perceptron multicouches (PMC) comme approximateur universel.

Le chapitre 1 présente les réseaux de neurones à travers des définitions et leurs propriétés. On présente notament plusieurs types et règles d'apprentissage. On y aborde principalement les perceptrons multicouches qui forment une très grande majorité des réseaux. Les méthodes d'apprentissage décrites sont celles de rétropropagation du gradient, la descente stochastique et la descente avec inertie. On propose aussi dans ce chapitre de connaitre les réseaux RBF (Radial Basis Functions). On y retrouve l'architecture de ces réseaux, leur fonctionnement ainsi que leur apprentissage. On trouve aussi d'autres algorithmes d'optimisation tels que l'algorithme de *Levenberg-Marquardt* [5]-[6] et la méthode de *Newton* [7].

Le chapitre 2 aborde l'approximation de fonctions par les réseaux de neurones et plus précisement du perceptron multicouches. On énonce le théorème de *Kolmogorov* et de *Sprecher* sur l'approximation, ainsi que la propriété fondamentale des réseaux de neurones non bouclés à apprentissage supervisé, soit l'approximation parcimonieuse.

Dans la section approximation de fonctions par le Perceptron Multicouches, on résume les résultats fondamentaux sur les capacités d'approximation de fonctions continues par des réseaux de neurones. On énonce aussi le théorème qui montre que les RBF sont des approximateurs universels.

Dans la fin du chapitre, on s'intéresse au choix du nombre d'unités cachées dans un réseau de neurones, donc de fixer le nombre de couches cachées, de déterminer le nombre de neurones par couches cachées, choisir la fonction d'activation et enfin choisir l'apprentissage.

Le chapitre 3 du présent mémoire est consacré à montrer les liens existants entre les statistiques et les réseaux de neurones. On aborde d'abord le problème d'approximation de fonction et la modélisation statistique, ensuite on s'intéresse aux statistiques et théories de l'apprentissage.

Le chapitre 4 est consacré à la complexité des algorithmes d'apprentissage.

Dans le cas des réseaux de neurones, l'important n'est pas seulement l'architecture du réseau mais aussi la définition de l'algorithme d'apprentissage. Le fait d'admettre que les règles d'apprentissage effectuent une optimisation, permet de les appréhender de la même façon que les autres problèmes informatiques. Il permet d'appliquer des outils de la théorie de la complexité.

Dans ce chapitre, on définit les classes de complexité P, NP et $NP-Complet$ et on montre que les problèmes d'apprentissage sont NP complet.

Chapitre 1

Les réseaux de neurones

1.1 Histoire des réseaux de neurones

Les réseaux de neurones sont des modèles mathématiques et informatiques des assemblages d'unités de calcul appelés neurones formels, et dont l'inspiration originelle était un modèle de la cellule nerveuse humaine. Cet héritage de la neurobiologie forme une composante importante de la matiére, et le souci de maintenir une certaine correspondance avec le système nerveux humain a animé et continue à animer une part importante des recherches dans le domaine.

Les réseaux de neurones ont vu le jour il y a une cinquantaine d'années, des efforts combinés de scientifiques divers et aux motivations variées.

Tout commence en 1943, lorsque deux bio-physiciens de l'université de Chicago, *Mc Culloch* et *Pitts* [8], s'inspirant des récentes découvertes en neurobiologie, conçoivent le premier modèle du neurone biologique, baptisé neurone formel ou automate à seuil.

Le neurone formel, aussi fréquemment appelé neurone de *Mc Culloch* et *Pitts*, est décrit de façon très simple comme un sommateur à seuil à sortie binaire, pondérant les signaux reçus des autres neurones.

Les deux biophysiciens américains n'ont pas seulement le mérite d'avoir conçu ce modèle qui est encore aujourd'hui le plus utilisé, mais ils ont également prouvé qu'un système de tel neurones était en principe capable de calculer n'importe quelle fonction, à condition que les poids associés aux connexions entre neurones soient judicieusement choisis.

Un peu plus tard, un neurophysiologiste nommé *Donald Hebb*, propose en 1949 [9] une formulation du mécanisme d'apprentissage, sous la forme d'une règle de modification des connexions synaptiques qui porte encore son nom.

Le fait remarquable est qu'il ne s'agit à l'époque que d'une simple "*intuition*", que les techniques expérimentales ne permettent pas encore de corroborer.

L'intuition de *Donald Hebb*, en ajoutant la dimension dynamique au schéma statique dressé par *Mc Culloch* et *Pitts*, a constitué "*l'étincelle de vie*" qui a permis aux réseaux de neurones artificiels de connaître un véritable essor.

Finalement, c'est en 1958 que *Rosenblatt* [10], combinant les idées de ses prédécesseurs, conçoit le Perceptron , un réseau de neurones artificiels inspiré du système visuel (la partie du cerveau qui reste de loin la mieux connue), possédant une couche de neurones "*perceptive*" et une couche de neurones "*décisionnelle*".

Ce réseau, qui parvient à apprendre à identifier des formes simples et à calculer certaines fonctions logiques, constitue le premier système artificiel exhibant une faculté jusque là réservée au vivant, la capacité d'apprendre par l'expérience, le premier réseau de neurones artificiel proprement dit.

En particulier, *Rosenblatt* démontre un théorème fondamental, qui porte le nom de "**perceptron learning theorem**", qui établit qu'un perceptron, s'il est en mesure de réaliser une tâche, sera toujours capable de l'apprendre en un temps fini (c'est à dire au bout d'un nombre fini de cycles d'apprentissage).

Les travaux de *Rosenblatt* suscitent au début des années 60 un vif enthousiasme chez les scientifiques alors fortement impliqués dans la recherche sur l'intelligence artificielle .

Cet enthousiasme se voit pourtant brusquement refroidi en 1969 lorsque deux scientifiques américains de renom, *Minsky* et *Papert* [11] publient un livre qui, au terme d'une analyse mathématique approfondie, met à jour les limites intrinsèques du perceptron, en particulier son incapacité à résoudre les problèmes non linéairement séparables.

Ce qu'ont démontré *Minsky* et *Papert* c'est qu'un réseau de neurones de type perceptron, c'est à dire ne possédant qu'une couche de sortie, est incapable de résoudre toute une classe de problèmes simples (les problèmes non linéairement séparables).

Certes l'utilisation de couches intermédiaires (ou cachées) de neurones, permettrait de contourner cette limitation, à condition de disposer d'un mécanisme d'apprentissage approprié pour ces neurones additionnels.

Mais c'est précisément ce mécanisme qui à l'époque fait cruellement défaut.

Il faut attendre le début des années 80 pour voir un regain d'intérêt pour les réseaux de neurones artificiels. Celui-ci s'explique tout d'abord par les résultats des travaux complètement connectés

(les réseaux reccurents, avec "*feed-back*", qui constituent la deuxième grande classe de réseaux avec les réseaux de type perceptron, aussi qualifiés de "*feed-forward*") dans la compréhension et la modélisation des processus de la mémoire.

Parallèlement aux travaux de *Hopfield* [12], *Werbos* [13] conçoit un mécanisme d'apprentissage pour les réseaux multicouches de type perceptron : c'est l'algorithme d'apprentissage par "*Back-propagation*" (retro propagation de l'erreur) qui fournit un moyen simple d'entraîner les neurones des couches cachées.

Cet algorithme sera réellement popularisé en 1986 par *Rumelhart* et *al.* [14], ils ont pu ainsi fournir un moyen d'entraîner les réseaux du type "perceptron" muni d'un nombre quelconque de couches cachées.

Depuis la fin des années 80, l'intérêt pour les réseaux de neurones artificiels ne s'est pas démenti, dans tous les milieux et sur tous les fronts.

Du côté théorique, on a pu démontrer rigoureusement un résultat de grande importance affirmant que les réseaux MLP possédant seulement deux couches cachées sont capables d'approximer avec précision arbitraire n'importe quelle fonction.

1.2 Les réseaux de neurones : définitions et propriétés

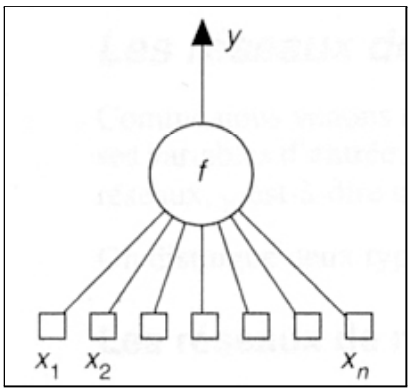
Définition 1.1 : *Un neurone est une fonction non linéaire, paramétrée, à valeurs bornées [15].*

Les variables sur lesquelles opère le neurone sont habituellement désignées sous le terme d'entrées du neurone, et la valeur de la fonction sous celui de sortie. Il est commode de représenter graphiquement un neurone comme indiqué sur la figure 1-1. Cette représentation est le reflet de l'inspiration biologique qui a été à l'origine de la première vague d'intérêt pour les neurones formels, dans les années 1940 à 1970 [8], [11].

La fonction f peut être paramétrée de manière quelconque. Deux types de paramétrages sont fréquemment utilisés :

1. Les paramètres sont attachés aux entrées du neurone : la sortie du neurone est une fonction non linéaire d'une combinaison des entrées $\{x_i\}$ pondérées par les paramètres $\{\omega_i\}$, qui sont alors souvent désignés sous le nom de "poids" ou, en raison de l'inspiration biologique des réseaux de neurones, "poids synaptiques".

   Conformément à l'usage (également inspiré par la biologie), cette combinaison linéaire est appelée potentiel. Le potentiel ν, le plus fréquement utilisé, est la somme pondérée à laquelle



Figure˜1-1: Un neurone formel

s'ajoute un terme constant ou "biais" :

$$\nu = w_0 + \sum_{i=1}^{n-1} \omega_i x_i \tag{1.1}$$

La fonction f est appelée fonction d'activation; l'état x_i du neurone i est une fonction de son activité.

La fonction f peut avoir plusieurs formes différentes. Il est recommandé d'utiliser pour f une fonction "sigmoïde" symétrique par rapport à l'origine, telle que la tangente hyperbolique ou la fonction arctangente.

2. Les paramètres sont attachés à la non linéarité du neurone : ils interviennent directement dans la fonction f ; cette dernière peut être une fonction radiale ou RBF.

   Par exemple, la sortie d'un neurone RBF à non linéarité gaussienne a pour équation :

$$y = \exp\left[-\frac{\sum_{i=1}^{n} (x_i - \omega_i)^2}{2\omega_{n+1}^2} \right] \tag{1.2}$$

   où les paramètres ω_i, $i = 1$ à n sont les coordonnées du centre de la gaussiènne et w_{n+1} est son écart type.

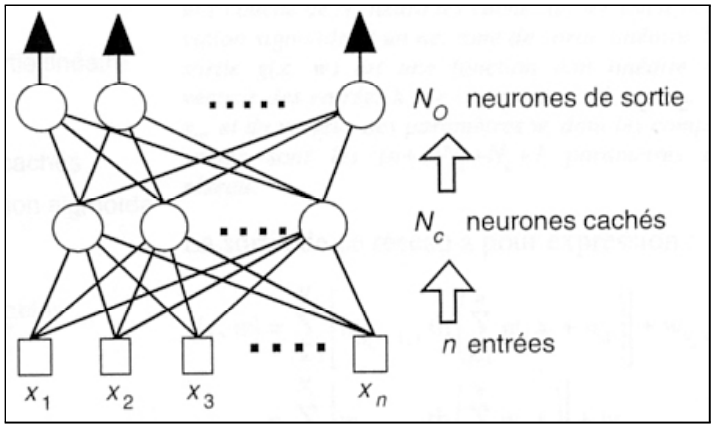
On distingue deux types de réseaux de neurones : les réseaux non bouclés et les réseaux bouclés.

1.2.1 Les réseaux de neurones non bouclés

Définition 1.2 : *Un réseau de neurones non bouclé réalise une (ou plusieurs) fonctions des ses entrées, par composition des fonctions réalisées par chacun des neurones [15].*

"Connectés" entre eux, l'information circulant des entrées vers les sorties sans "retour en arrière" : si l'on représente le réseau comme un graphe dont les noeuds sont des neurones et les arêtes les "connexions" entre ceux-ci, le graphe d'un réseau non bouclé est acyclique. Si l'on se déplace dans le réseau, à partir d'un neurone quelconque, en suivant les connexions, on ne peut pas revenir au neurone de départ.

La représentation de la topologie d'un réseau par un graphe est trés utile . Les neurones qui effectuent le dernier calcul de la composition de fonctions sont *les neurones de sortie*. Ceux qui effectuent des calculs intermédiaires sont *les neurones cachés*, comme le montre la figure 1-2.



Figure˜1-2: Un réseau de neurones

Ce réseau réalise N_O fonctions algébriques des variables d'entrée du réseau. Chacune des sorties est une fonction, réalisée par le neurone de sortie correspondant, des fonctions non linéaires réalisées par les neurones cachés.

Définition 1.3 : *Un réseau de neurones non bouclé à n entrées, N_C neurones cachés et N_O neurones de sortie réalise N_O fonctions non linéaires de ses n variables d'entrée par composition des N_C fonctions déjà réalisées par ses neurones cachés [15].*

Remarques :

1. Les réseaux de neurones non bouclés à couches, dont les neurones cachés ont une fonction d'activation sigmoïde, sont souvent appelés *perceptrons multicouches* (ou MLP, Multi Layer

Perceptron).

2. Les réseaux à une couche cachée de sigmoïde est un neurone de sortie linéaire.

3. Un réseaux de neurones non bouclé sans neurone caché , avec un neurone de sortie linéaire, réalise simplement une fonction linéaire de ses entrées.

1.2.2 Les réseaux de neurones bouclés

Le graphe des connexions dans ce type de réseaux est cyclique, c'est à dire lorsqu'on se déplace dans le réseau en suivant le sens des connexions, il est possible de trouver au moins un chemin qui revient à son point de départ (un tel chemin est désigné sous le terme de "cycle"). La sortie d'un neurone du réseau peut donc être fonction d'elle même. Cela n'est évidement concevable que si la notion de temps est explicitement prise en considération.

Ainsi, à chaque connexion d'un réseau de neurones bouclé (ou à chaque arête de son graphe) est attaché, outre un poids comme les réseaux non bouclés, un retard, multiple entier (éventuellement nul) de l'unité de temps choisie. Une grandeur, à un instant donné, ne pouvant pas être fonction de sa propre valeur au même instant, tout cycle de graphe du réseau doit avoir un retard normal.

1.3 Apprentissage des réseaux de neurones

1.3.1 Présentation

Pour un réseau de neurones, l'apprentissage peut être considéré comme les problèmes de la mise à jour des poids des connexions au sein du réseau afin de réussir la tâche qui lui est demandée.

l'apprentissage est la caractéristique principale des RNA et il peut se faire de différentes manières et selon différentes règles.

1.3.2 Type d'apprentissage

Le mode supervisé

Dans ce type d'apprentissage, le réseau s'adapte par comparaison entre les résultats qu'il a calculé, en fonction des entrées fournies, et la réponse attendue en sortie. Ainsi, le réseau va se modifier jusqu'à ce qu'il trouve la bonne sortie, c'est à dire celle attendue, correspondant à une entrée donnée.

Le renforcement

Le renforcement est en fait une sorte d'apprentissage supervisé et certains auteurs le classent, d'ailleurs, dans la catégorie des modes non supervisés. Dans cette approche le réseau doit apprendre la corrélation entrée/sortie via une estimation de son erreur, c'est à dire du rapport échec/succés. le réseau va donc tendre à maximiser un index de performance qui lui est fourni, appelé signal de renforcement. Le système étant capable ici, de savoir si la réponse qu'il fournit est correcte ou non, mais il ne connait pas la bonne réponse.

Le mode non-supervisé (ou auto-organisationnel)

Dans ce cas, l'apprentissage est basé sur des probabilités. Le réseau va se modifier en fonction des régularités statistiques de l'entrée et établir des catégories, en attribuant et en optimisant une valeur de qualité, aux catégories reconnues.

Le mode hybride

Le mode hybride reprend en fait les deux autres approches, puisque une partie des poids va être déterminée par apprentissage supervisé et l'autre partie par apprentissage non-supervisé.

1.3.3 Règles d'apprentissage

Règle de *Hebb*

Les réseaux de neurones se sont développés grâce à leur capacité d'apprentissage. Le psychologue *Hebb* fût le premier à présenter un apprentissage à partir des neurones artificiels :

"Lorsqu'un axone de la cellule A est suffisament proche pour pouvoir exciter la cellule B et qu'il prend part de manière répétitive ou persistente à cette excitation, alors on doit trouver soit un phénomène de croissance, soit un changement métabolique dans l'une ou l'autre des cellules tel que l'efficacité de la cellule A pour exciter B doit être accrue".

En d'autres termes, la synapse entre la cellule A et la cellule B doit faciliter la transmission de l'influx nerveux lorsque les deux sont excitées en même temps, et au contraire si les cellules A et B ne sont pas activées sur la même excitation, la synapse doit inhiber cette transmission.

Ainsi la synapse reliant deux cellules est affectée d'un poids qui varie pendant l'apprentissage.

Les règles du Delta (ou règle de *Widrow-Hoff*)

Son but est de faire évoluer le réseau vers le minimum de sa fonction d'erreur (erreur commise sur l'ensemble des exemples).

L'apprentissage est réalisé par itération (les poids sont modifiés après chaque exemple présenté), et on obtient le poids à l'instant $t+1$ par la formule :

$$w(t+1) = w(t) + \eta(T-O)E \tag{1.3}$$

où w est le poids, T la sortie théorique et O la sortie réelle, E l'entrée et η un coefficient d'apprentissage (entre 0 et 1) que l'on peut diminuer au cours de l'apprentissage.

Cette règle s'interprète par le fait que l'on modifie le poids proportionnellement à l'erreur (différence entre sortie théorique et sortie donnée) et à l'état d'excitation de l'entrée gérée par le poids. Ainsi, le poids ne sera pas modifié si la sortie est celle voulue, ou si le poids n'a joué aucun rôle dans le résultat.

C'est en fait un cas particulier de l'algorithme de rétropropagation du gradient pour un réseau à une couche.

La rétropropagation du Gradient (backpropagation)

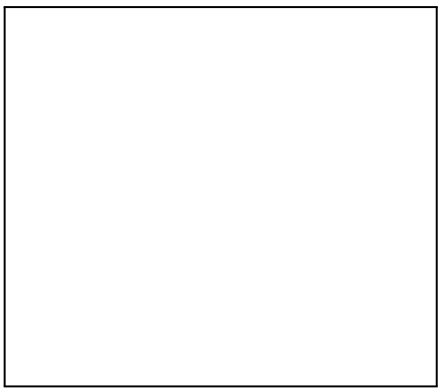
Il a fallu attendre le début des années 80 pour qu'une règle efficace soit mise au point pour l'apprentissage des réseaux multicouches. Cette règle est en fait une généralisation de la règle du delta.

Elle consiste simplement en une descente de gradient, qui est une méthode d'optimisation universelle. On cherche à minimiser une fonction d'erreur (qui représente l'erreur entre la sortie désirée et la sortie obtenue), en suivant les lignes de plus grande pente. Une fonction d'erreur rapportée à une dimension peut se représenter ainsi :

On peut se représenter la descente du gradient comme une bille que l'on poserait sur la courbe, et qui descendrait logiquement la pente (le gradient représente la pente selon chaque dimension).

L'inconvénient de cette méthode est qu'elle va s'arrêter dans le premier minimum local rencontré. C'est pourquoi diverses améliorations ont été apportées.

Diverses solutions permettent de sortir des minima locaux, visant à modifier l'allure locale de l'hyper surface d'erreur seront détaillées dans la section qui suit.



Figure˜1-3: Représentation d'une fonction d'erreur

1.4 Le perceptron multicouches (Multi-Layer Perceptron, MLP)

Un des premiers chercheurs en connexionnisme, *Frank Rosenblatt* [10], s'intéressait à la possibilité d'expliquer les capacités d'association des êtres humains par des capacités équivalentes dans des modèles neuromimétriques. Ainsi, le Perceptron devait modéliser le fonctionnement de la perception humaine. Relié à une caméra, ce réseau de neurones à seuil recevait sur sa couche d'entrée (la rétine) une représentation digitalisée du champ de vision de la caméra. Cette première couche était connectée à une seconde couche (l'aire d'association) qui avait la responsabilité de répérer un ensemble de traits élementaires. Enfin, la couche de sortie (aire de réponse) devait reconnaître l'objet sous l'oeil de la caméra.

Les MLP, ou réseaux à couches, forment la très grande majorité des réseaux. Ils sont intemporels (ce sont des réseaux statiques et non dynamiques).

1.4.1 Structure

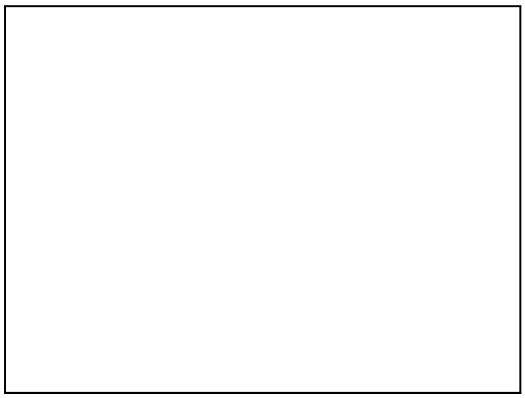
Les neurones sont organisés en couches : chaque neurone est connecté à toutes les sorties des neurones de la couche précédente, et nourrit de sa sortie tous les neurones de la couche suivante (ces réseaux sont d'ailleurs qualifiés de *feedforward*). Pour la première couche, ses entrées sont l'entrée du réseau.

D'ailleurs une couche est souvent rajoutée pour constituer les entrées, appelée couche d'entrée, mais elle n'en est pas une puisqu'elle ne réalise aucun traîtement.

Les fonctions d'entrée et de transfert sont les mêmes pour les neurones d'une même couche, mais peuvent différer selon la couche. Ainsi la fonction de transfert de la couche de sortie est

généralement l'identité.

Un réseau à trois couches (deux couches cachées et une de sortie) et à trois entrées peut donc se représenter ainsi :



Figure~1-4: Un réseau à 3 couches

1.4.2 Apprentissage

Notions d'optimisation

L'apprentissage supervisé des réseaux multicouches est une question d'optimisation, c'est à dire consiste en la minimisation d'un critère d'erreur.

Il existe divers types d'optimisation. Elle peut être différentiable ou non, opérant sur des données linéaires ou non. Dans le cas des réseaux de neurones, on s'intéresse à l'optimisation différentiable non linéaire.

Le but est de trouver un point de l'espace des variables en lequel le critère d'erreur est minimal, c'est à dire un minimum global du critère d'erreur.

Définition 1.4 : *Le point X_0 de l'espace E est un minimum global de la fonction f définie sur E si et seulement si, pour tout X de E, on a :*

$$f(X_0) \leq f(X) \tag{1.4}$$

Définition 1.5 : *Le point X_0 de l'espace E est un minimum local de la fonction f définie sur*

E si et seulement si il existe un voisinage V de X_0 dans E, tel que, pour tout X de V on a :

$$f(X_0) \leq f(X) \tag{1.5}$$

Les minima locaux sont faciles à reconnaître : Une condition nécessaire est que le gradient s'annule en ce point, une condition nécessaire est suffisante est que le gradient s'annule et que la matrice hessienne des dérivées secondes ($\frac{\partial^2 f}{\partial x_i \partial x_j}$) soit définie positive.

En revanche, les minima globaux sont beaucoup plus difficiles à déterminer. En effet, mis à part quelques cas particuliers (fonctions convexes) où la solution est facilement accessible, dans le cas général de fonctions non linéaires, une exploration exhaustive de la fonction est nécessaire pour construire l'enveloppe convexe, ce qui est impensable en dimension quelconque.

Remarque:

Matrice Hessienne = Matrice des dérivées secondes de la fonction coût par rapport aux paramètres.

Ainsi, l'apprentissage de neurones artificiels se réduit à un problème d'optimisation : trouver le minimum d'une fonction d'erreur. On pourra donc mettre à profit cette méthode d'optimisation universelle de descente du gradient, qui constituera la règle de rétropropagation du gradient pour les réseaux multicouches.

Méthodes itératives d'optimisation

Devant la difficulté d'obtenir une solution analytique lorsque les données sont non linéaires, l'émergence de méthodes itératives numériques basées sur une approximation du critère d'erreur (son développement à plusieurs variables de *Taylor*) a été favorisée. On veut donc minimiser à chaque itération la valeur $f(x_t)$. Cela est le cas si l'on a à chaque itération :

$$f(x_{t+1}) < f(x_t) \tag{1.6}$$

Si la variation $\Delta x_t = x_{t+1} - x_t$ est suffisamment faible, on peut approcher la valeur $f(x_{t+1})$ par le développement de *Taylor* à plusieurs variables en x_t :

$$f(x_{t+1}) = f(x_t + \Delta x_t) \approx f(x_t) + (\nabla f \mid_{x_t})^{\mathrm{T}} . \Delta x_t \tag{1.7}$$

où $\nabla f\mid_{x_t}$ est le gradient de f au point x_t.

L'objectif est donc d'avoir le produit scalaire entre le vecteur gradient et le vecteur des variations de x négatif :

$$(\nabla f\mid_{x_t})^{\mathrm{T}}.\Delta x_t < 0 \tag{1.8}$$

On a une décroissance maximale de la fonction f si ce produit scalaire est négatif et minimal, donc si et seulement si les deux vecteurs sont de même direction et de sens opposés.

On obtient pour règle d'apprentissage :

$$x_{t+1} = x_t - \eta \nabla f\mid_{x_t} \tag{1.9}$$

où η est un coefficient compris entre 0 et 1.

On constate donc qu'il faut suivre le gradient, d'où le terme de descente de gradient.

L'apprentissage : un problème d'optimisation

Notations utilisées dans la modélisation du réseau multicouches

- n : nombre de couches du réseau (la pseudo couche d'entrée n'est pas comptée)
- $(m_i)_{0\leq i\leq n}$: nombre de neurones de la couche i

  $q = m_0$: nombre d'entrées du réseau (dimension du vecteur d'entrée $\overrightarrow{I}$)

  $r = m_n$: nombre de sorties du réseau (dimension du vecteur de sortie $\overrightarrow{O}$)
- $(S_{ij})_{\substack{0\leq i\leq n\\ 1\leq j\leq n_i}}$: sortie du neurone j de la couche i (et entrée j des neurones de la couche $i+1$)

  $(I_j)_{1\leq j\leq q} = (S_{0,j})_{1\leq j\leq q}$: entrées (in) du réseau (coordonnées du vecteur d'entrée $\overrightarrow{I}$)

  $(O_j)_{1\leq j\leq r} = (S_{n,j})_{1\leq j\leq r}$: sorties (out) du réseau (coordonnées du vecteur de sortie $\overrightarrow{O}$)
- $(W_{ijk})_{\substack{1\leq i\leq n\\ 1\leq j\leq m_i\\ 1\leq k\leq m_{i-1}}}$: poids (weight), dans le neurone j de la couche i, de la sortie du neurone k de la couche $i-1$
- $(V_{ij})_{\substack{1\leq i\leq n\\ 1\leq j\leq n_i}}$: potentiel du neurone j de la couche i (résultat de la fonction d'entrée)
- $(f_i)_{1\leq i\leq n}$: fonction de transfert de la couche i
- p : nombre d'exemples de la base

  $(E_{k,j})_{\substack{0\leq k\leq p\\ 1\leq j\leq q}}$: exemples de la base (entrées du réseau)

  $(T_{k,j})_{\substack{0\leq k\leq p\\ 1\leq j\leq r}}$: sorties théoriques du réseau pour l'exemple k

On notera également W le vecteur poids du réseau, dont les coordonnées sont l'ensemble des poids W_{ijk} du réseau.

La fonction d'erreur

Contexte : On souhaite modéliser une fonction inconnue $y = f(x)$, et l'on dispose seulement de p échantillons : p observations $\{y_1, y_2, , y_p\}$ de y correspondant à p observations $\{x_1, x_2, , x_p\}$ de x, les x_i et y_i étant entachés de bruit dû à une imprécision de mesure ou à des parasites.

On peut donc définir la relation entre les x_i et y_i par $y = f(x) + e$ où e est un vecteur d'erreur aléatoire.

On supposera de plus que l'erreur moyenne pour une valeur x est nulle (il n'y a pas d'erreur systématique, simplement une erreur accidentelle) :

$$E\left[\varepsilon \mid x\right] = 0 \tag{1.10}$$

et que e et $f(x)$ sont non corrélés :

$$E\left[e.f(x)\right] = 0 \tag{1.11}$$

où $E[.]$ désigne l'espérance mathématique et $E[.|.]$ l'espérance conditionnelle

Il est ainsi possible d'estimer la relation f par $f(x) = E[y|x]$, ce qui est raisonnable pour pouvoir espérer modéliser f. En effet si un décalage constant intervient par exemple sur les mesures, ce ne serait alors plus f qui serait modélisée.

Erreur instantanée : En fournissant alors au réseau une entrée x, celui-ci va fournir une sortie O. Il faut alors estimer l'erreur instantanée qui représente le coût encouru à confondre la sortie du réseau à la réponse désirée y. Celui-ci peut l'être par n'importe quelle distance entre ces deux vecteurs. La distance euclidienne quadratique (correspondant à la fonction de coût des moindres carrées) est usuellement utilisée :

$$C(x, w) = \|y - O\|^2 \tag{1.12}$$

ou encore avec les notations générales :

$$C_k = C(E_k) = \sum_{j=1}^{r} (T_{x,j} - O_{x,j})^2 \tag{1.13}$$

Son intérêt, qui fait qu'elle est largement employée dans tous les domaines, est que les calculs restent simples, qu'elle est infiniment dérivable et que dans le cas d'une distribution normale des erreurs (99% des cas), la minimisation du coût des moindres carrées est celle qui donne le maximum de vraisemblance.

Erreur moyenne : L'erreur moyenne du réseau dans la modélisation de f est alors l'espérance du coût instantané C_k : c'est la moyenne de C_k pour toutes les valeurs de x et y, pondérée par leur probabilité de rencontre. Il faut donc en théorie utiliser la fonction de distribution cumulative de probabilité de x et y considérées comme variables aléatoires :

$$F_X(x) = Pr(X \leq x) \tag{1.14}$$

représente la probabilité pour qu'une réalisation de la variable aléatoire X soit inférieure à x.

En pratique on ne dispose que de nos p échantillons, et on ne connaît pas la fonction de distribution cumulative de probabilité. On est aussi réduit à utiliser une approximation empirique de ce critère d'erreur moyenne, qui est simplement la moyenne du coût sur les p échantillons :

$$C = \frac{1}{p}. \sum_{k=1}^{p} C_k. \tag{1.15}$$

L'algorithme de rétropropagation du gradient On a présenté dans la section "Notions d'optimisations", par une approche intuitive, l'algorithme de rétropropagation du gradient, et on a démontré dans la section "Méthodes itératives d'optimisation", grâce à un développement de *Taylor*, qu'il fallait suivre le gradient. Il reste donc à déterminer ce gradient.

Calcul : On cherche donc, pour chaque poids (i, j et k fixés), à exprimer la pente de la courbe représentative de chaque application partielle associée à la fonction coût $C(W)$ en W (i.e. la dérivée partielle de C par rapport à chacun des poids), en fonction de paramètres connus du réseau (les poids, les potentiels, les fonctions de transfert).

Puisque ce coût total est la somme des coûts instantanés : $C = \sum_{x=0}^{n-1} C_x$, et que ces derniers sont positifs (c'est d'ailleurs pourquoi on préfère le terme coût au terme erreur), le coût total est

minimal lorsque chacun des coûts instantanés est minimal. Il faut donc minimiser chaque coût instantané C_x. Ceci n'est bien entendu rigoureusement valable qu'à condition de ne pas modifier les poids entre deux exemples.

Hypothèses

→Fonction d'entrée (produit scalaire entre le vecteur S_{i-1} et le vecteur W_{ij}) :

$$V_{ij} = \sum_{k=1}^{m_i-1} W_{ijk}.S_{i-1,k} \tag{1.16}$$

→Fonctions de transfert f_i quelconques mais dérivables :

$$S_{ij} = f_i(V_{ij}) \tag{1.17}$$

→Fonction de coût instantané $C_x(W)$ pour chaque exemple x (fonction de coût des moindres carrés) :

$$C_x = \frac{1}{2}.\sum_{j=1}^{r}(T_{x,j} - O_{x,j})^2 \tag{1.18}$$

→On pose de plus pour chaque neurone :

$$\delta_{ij} = -\frac{\partial C_x}{\partial V_{ij}} \tag{1.19}$$

Démonstration

$$\begin{aligned}
\frac{\partial C_x}{\partial W_{ijk}} &= \frac{\partial C_x}{\partial V_{ij}}.\frac{\partial V_{ij}}{\partial W_{ijk}} = -\delta_{ij}.\frac{\partial V_{ij}}{\partial W_{ijk}} \text{ (dérivation des fonctions composées)} \\
\frac{\partial V_{ij}}{\partial W_{ijk}} &= S_{i-1,k} \text{ d'après 1.16} \\
\delta_{ij} &= -\frac{\partial C_x}{\partial V_{ij}} = \frac{\partial C_x}{\partial S_{ij}}.\frac{\partial S_{ij}}{\partial V_{ij}} \\
\frac{\partial S_{ij}}{\partial V_{ij}} &= f_i'(V_{ij}) \text{ d'après 1.17}
\end{aligned}$$

• pour la couche de sortie ($i = n$):

$$\frac{\partial C_x}{\partial S_{nj}} = \frac{\partial C_x}{\partial O_j} = -(T_j - O_j) \text{ d'après 1.18}$$

$$\delta_{nj} = f_n'(V_{nj}).(T_j - O_j)$$

• pour une couche autre que celle de sortie ($i < n$):

$$\begin{aligned}
\frac{\partial C_x}{\partial S_{ij}} &= \sum_{l=1}^{m_{i+1}} \frac{\partial C_x}{\partial V_{i+1,l}} . \frac{\partial V_{i+1,l}}{\partial S_{ij}} \text{ (puisque les } V_{i+1,l} \text{ sont fonctions de } S_{ij}) \\
\frac{\partial V_{i+1,l}}{\partial S_{ij}} &= \frac{\partial}{\partial S_{ij}} \left(\sum_{k=1}^{m_i} W_{i+1,l,k} . S_{ik} \right) = W_{i+1,l,j} \text{ d'après 1.16} \\
\frac{\partial C_x}{\partial V_{i+1,l}} &= -\delta_{i+1,l} \text{ d'après 1.19} \\
\frac{\partial C_x}{\partial S_{ij}} &= -\sum_{l=1}^{m_{i+1}} \delta_{i+1,l} . W_{i+1,l,j} \\
\delta_{ij} &= f'_i(V_{ij}) . \sum_{l=1}^{m_{i+1}} \delta_{i+1,l} . W_{i+1,l,j}
\end{aligned}$$

$$\frac{\partial C_x}{\partial W_{ijk}} = -S_{i-1,k} . \delta_{ij}$$

Résultat Or on a vu en 1.9 que la règle d'apprentissage était :

$$W_{ijk}(t+1) = W_{ijk}(t) - \eta . \frac{\partial C_x}{\partial W_{ijk}}$$

Ce qui donne donc :

$$W_{ijk}(t+1) = W_{ijk}(t) + . S_{i-1,k} . \delta_{ij} \quad \text{avec} \begin{cases} \delta_{nj} = f'_n(V_{nj}) . (T_j - O_j) \\ \delta_{ij} = f'_i(V_{ij}) . \sum\limits_{l=1}^{m_{i+1}} \delta_{i+1,l} . W_{i+1,l,j} \end{cases}$$

Le coefficient η est le coefficient d'apprentissage ou d'adaptation, qui permet d'ajuster les modifications et qui doit être soigneusement choisi. En effet plus il est élevé et plus l'apprentissage est rapide, mais au détriment de la stabilité. En général sa valeur est fixée entre 0 et 1, et une valeur entre 0.3 et 0.7 représente un bon compromis.

Localité de l'optimisation et améliorations Le problème de cette optimisation est qu'elle reste locale, c'est-à-dire que si le réseau se trouve dans un minimum local de la fonction d'erreur, il y restera.

Diverses solutions permettent de sortir de minima locaux, visant à modifier l'allure locale de l'hypersurface d'erreur.

a. La descente stochastique : Comme dans la règle de *Widrow-Hoff*, on minimise itérativement l'erreur due à chaque exemple. Les apprentissages de chaque exemple s'influençant les uns les autres, cela permet de passer sur des petits minima locaux.

Une présentation aléatoire des exemples donne généralement de meilleurs résultats.

b. La descente avec inertie : On introduit ici un moment d'inertie α (terme de momentum), qui correspond dans notre image de la bille, à l'inertie qu'elle acquiert en descendant la courbe : son élan lui permettra de ne pas s'arrêter dans le premier minimum local rencontré.

La règle d'apprentissage devient alors :

$$\Delta W_{ijk}(t) = W_{ijk}(t+1) - W_{ijk}(t) = .S_{i-1,k}.\delta_{ij} + \alpha \Delta W_{ijk}(t-1) \tag{1.20}$$

On peut également calculer dynamiquement les coefficients grâce à la règle suivante :

- Calculer un coefficient de corrélation : $\sum_{i,j,k} \Delta W_{ijk}(t).\Delta W_{ijk}(t-1)$
- S'il est négatif , on pose $\eta = 0.01$ et $\alpha = 0$
- S'il est positif, on pose $\eta = \eta + \delta\eta$ et $\alpha = \alpha + 0.01$

Il peut être nécessaire de borner ces coefficients, et on peut également décider de ne les augmenter que si le coefficient de corrélation est supérieur d'au moins 5% à sa précédente valeur.

Vitesse de convergence et améliorations : Méthodes du second ordre La convergence peut se révéler très lente à proximité des minima si la pente est faible. Des méthodes dites du second ordre, utilisant une approximation quadratique (au second ordre) du critère d'erreur permettent d'approcher beaucoup plus vite le minima grâce à la courbure quadratique de la fonction d'erreur au voisinage immédiat des minima (la convergence s'effectue en une seule itération si la courbure est idéalement quadratique).

La méthode de base est connue sous la dénomination d'optimisation de *Newton*, mais en pratique ne se fiant pas à la pente, cette méthode est sujette à converger vers des points non minimaux où le gradient s'annule, comme les points d'inflexion. C'est pourquoi la méthode de *Levenberg-Macquardt* par exemple combine une optimisation du premier ordre à une optimisation du second ordre au voisinage des minima.

Cependant, une optimisation du second ordre nécessite la matrice hessienne inverse des dérivées partielles du second ordre, dont le calcul et le stockage peut se révéler handicapant pour des réseaux de grande taille. Des méthodes, dites *quasi-Newton*, utilisent une estimation itérative de la matrice

hessienne, réduisant le coût opératoire.

a. Méthode de *Newton :* La méthode de *Newton* utilise la courbure (dérivée seconde) de la fonction coût pour atteindre le minimum plus rapidement. La modification des paramètres est donnée par :

$$W_k = W_{k-1} - H_{k-1}^{-1} \nabla C_{k-1} \tag{1.21}$$

Dans ce cas, le pas est constant et égal à 1. La direction de descente est fonction du hessien et du gradient.

Si $C(W)$ (fonction coût) est quadratique, l'algorithme atteint la solution en une seule itération. Sinon, cette méthode est très efficace au voisinage d'un minimum.

Cependant, pour que la méthode converge vers le minimum, le hessien doit être défini positif. Dans le cas général d'un modèle non-linéaire, cette hypothèse de convergence n'est pas toujours respectée et la méthode peut ne pas converger.

En pratique, elle est peu employée car elle nécessite, en plus, le calcul du hessien à chaque itération. On lui préfère des méthodes plus économiques dites de quasi-*Newton*.

b. Méthode de *quasi-Newton* : Ici, l'inverse du hessien est approché par une matrice M_k définie positive modifiée à chaque itération. La suite des matrices $\{M_k\}$ est construite de manière à converger vers l'inverse du hessien lorsque la fonction de coût $C(W)$ est une quadratique.

La modification des paramètres est donnée par :

$$W_k = W_{k-1} - u_{k-1}.M_{k-1} \nabla C_{k-1} \tag{1.22}$$

où u_{k-1} est le pas (évalué par une méthode de minimisation unidimentionnelle).

A la première itération, la matrice M_0 est prise égale à la matrice identité.

c. Méthode de *Levenberg-Marquardt* : Une autre manière de diminuer le nombre d'itérations d'un algorithme d'optimisation est d'utiliser les dérivées secondes de C .

En effet, le gradient donne une direction vers laquelle se déplacer pour trouver le minimum, mais ne donne pas le pas.

Dans la descente du gradient classique, ce pas est coefficient fixe, et dans la variante adaptative il peut varier à chaque itération.

Mais la dérivée seconde de C est liée au rayon de courbure de la fonction, et permet donc de déterminer ce pas de manière plus fine.

Mais le calcul des dérivées secondes peut-être très long. Tout d'abord, parce que le nombre de dérivées secondes est le carré de celui des dérivées premières, et également parce que la dérivée seconde de C peut être assez complexe.

De nombreux algorithmes, peut-être abusivement appelés algorithmes d'ordre 2, utilisent en fait une approximation des dérivées secondes calculées à partir de dérivées premières. Cependant, ils gardent l'avantage de nécessiter beaucoup moins d'itérations qu'une descente de gradient.

L'agorithme de *Levenberg-Marquardt* fait partie de ces algorithmes, et s'applique au cas particulier où C est une erreur quadratique moyenne. On peut donc l'exprimer sous la forme :

$$C(W) = \left\langle (g(X,W) - y)^2 \right\rangle \tag{1.23}$$

où : g désigne une fonction de deux vecteurs X et W ; $\langle . \rangle$ désigne la moyenne calculée sur un ensemble de couples (x, y).

L'on se place dans le cas où g est une fonction scalaire afin de simplifier la notation, mais la même démarche peut être faite si g est une fonction vectorielle.

Dans la suite de cette section, toutes les dérivées sont en fonction du vecteur W. C'est en effet uniquement ce vecteur que l'on fait varier afin de trouver le minimum de C.

On suppose que l'on se trouve à une itération numéro i, et que l'on cherche à calculer un nouveau vecteur W_i en fonction de W_{i-1}.

Pour cela, on calcule une approximation quadratique $\widehat{C}$ de C à partir d'une approximation linéaire $\widehat{g}$ de g autour du point W_{i-1}. En déterminant le point W pour lequel le gradient de $\widehat{C}$ s'annule, on obtient :

L'expression du vecteur qui minimise l'approximation de C :

$$W = W_{i-1} - H^{-1} d \tag{1.24}$$

avec :

$$\begin{aligned} d &= \langle (g(X, W_{i-1}) - y) \nabla g(X, W_{i-1}) \rangle \\ H &= \left\langle \nabla g(X, W_{i-1}) \nabla g(X, W_{i-1})^{\mathrm{T}} \right\rangle \end{aligned} \tag{1.25}$$

à condition que H soit inversible. La matrice H est une approximation du hessien de C, calculée à

partir du gradient de g.

L'équation précédente pourrait servir dans un algorithme d'optimisation, qui permet de calculer W_i à partir de W_{i-1} au cours de l'itération i. Mais ceci n'est efficace en pratique que si g est effectivement proche d'une droite autour du point W_{i-1}.

Dans le cas contraire, cet algorithme donne de très mauvais résultats.

L'idée de *Levenberg* [5] est donc d'utiliser cette approche quadratique dans les zones où g est quasi-linéaire, et une descente de gradient dans les autres cas. Le pas d'une itération de cet algorithme est calculé de la manière suivante :

$$\text{Pas de } \textit{Levenberg} : \; W_i = W_{i-1} - (H - \lambda I)^{-1} d \tag{1.26}$$

Lorsque λ est faible, cette équation est équivalente à l'équation 1.24, et le nouveau vecteur de paramètres est déterminé avec l'approximation quadratique de C. Lorsque λ est grand, cette équation est équivalente à :

$$\begin{aligned} W_i &= W_{i-1} - \frac{1}{\lambda} d \\ &= W_{i-1} - \frac{1}{\lambda} \langle (g(X, W_{i-1}) - y) \nabla g(X, W_{i-1}) \rangle \\ &= W_{i-1} - \frac{1}{2\lambda} \nabla C(x, W_{i-1}) \end{aligned} \tag{1.27}$$

On obtient ainsi l'équivalence entre l'agorithme de *Levenberg* et la descente de gradient pour λ grand.

Ce qui correspond bien à une descente de gradient. Pour des valeurs intermédiaires de λ, l'algorithme est un mélange entre la descente du gradient et l'approche quadratique basée sur l'approximation linéaire de g.

Ce coefficient λ est modifié à chaque itération.

Si $C(W_i)$ diminue au cours de l'itération, on diminue λ (en le divisant par 10 par exemple), et l'on se rapproche ainsi de la méthode quadratique.

Au contraire, si $C(W_i)$ augmente, cela signifie que nous nous trouvons dans une région où g n'est pas très linéaire, et donc on augmente λ (en le multipliant par 10 par exemple) afin de se rapprocher de la descente du gradient.

Cet algorithme a ensuite été amélioré par *Marquardt* [6]. Le problème de l'itération étant défini

cette fois par :

$$\text{Pas de } \textit{Levenberg-Marquardt} : \; W_i = W_{i-1} - (H - \lambda \text{diag}(H))^{-1} d \tag{1.28}$$

La matrice identité a été remplacée par la diagonale de H.

Le but ici est de modifier le comportement de l'algorithme dans le cas où λ est grand, c'est à dire lorsque l'on est proche d'une descente du gradient. Avec cette modification, on se place plus vite dans les directions vers lesquelles le gradient est plus faible afin d'éviter de passer de nombreuses itérations sur un plateau. Ceci est appelé l'algorithme de *Levenberg-Marquardt*.

En pratique, cet algorithme, en particulier dans les cas des réseaux de neurones, permet de converger avec beaucoup moins d'itérations. Mais chaque itération demande plus de calculs, en particulier pour l'inversion de la matrice H, et son utilisation se limite donc aux cas où le nombre de paramètres à optimiser n'est pas très élevé.

En effet, le nombre d'opérations nécessaires à l'inversion d'une matrice est proportionnel à N^3; N étant la taille de la matrice, et ici également la taille du vecteur W.

1.5 Les réseaux RBF (Radial Basis Function)

1.5.1 Introduction

Les réseaux RBF sont des réseaux à couches qui ont comme origine une technique d'interpolation nommée méthode d'interpolation RBF, employée pour la première fois dans le contexte des réseaux neurométriques par *Broomhead* et *Lowe* [16].

1.5.2 Architecture

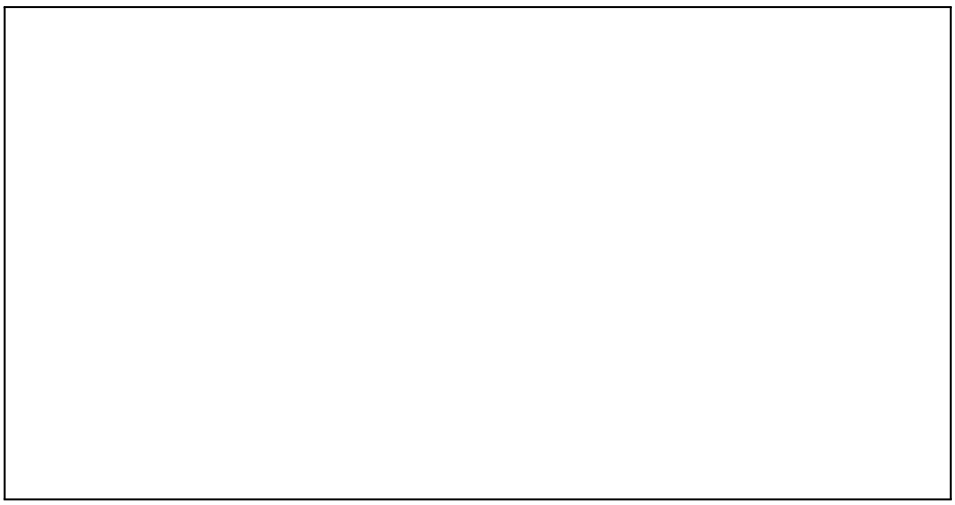
Le réseau RBF est un réseau à deux couches, qui diffère du PMC par la fonction d'activation des cellules cachées qui est de forme gaussienne.

Les neurones de la couche cachée sont reliés aux entrées et ont chacun deux paramètres : un vecteur prototype "C" (centre des gaussiennes) et un coefficient d'étalement "σ" strictement positif.

Les neurones de la couche de sortie sont généralement animés par une fonction d'activation linéaire.

La fonction réalisée par la premiére couche d'un réseau RBF est :

$$f_1(x) = e^{-\frac{\|x-c\|^2}{2\sigma^2}}$$



Figure˜1-5: Schéma général d'un réseau RBF

C définit un point de l'espace d'entrée. La sortie du neurone est égale à 1 pour une entrée x égale à C, puis décroit vers 0 lorsque l'entrée s'éloigne de C. La vitesse de décroissance est réglée par σ : Plus le coefficient est petit et plus la fonction sera concentrée autour du point C_i et proche de 0 ailleurs.

Les neurones de la seconde couche quant à eux calculent la sortie d'un réseau en effectuant une combinaison linéaire des sorties de ceux de la premiére couche et un biais est ajouté au total. La fonction qu'ils réalisent est la suivante:

$$f_2(X) = XW + b \tag{1.29}$$

où X est un vecteur composé des sorties de tous les neurones de la premiére couche , W est un vecteur de poids et b est le biais. W et b sont ajustés lors de l'apprentissage. Chaque sortie du réseau est donné par la formule:

$$S_i(X) = \sum_{j=1}^{m} \omega_{ij} \exp(-\frac{\|x - c_j\|^2}{2\sigma_j^2}) + b_i \tag{1.30}$$

où :

i est le numéro de la sortie (c'est à dire le numéro du neurone de la seconde couche dont on calcule la sortie),

m est le nombre de neurones de la premiére couche,

C_j est le vecteur prototype du neurone numéro j de la premiére couche,

σ_j son coefficient d'étalement,

ω_{ij} $(j = 1, ..., m)$ sont les poids du neurone de sortie i et b_i son biais.

1.5.3 Apprentissage

Lors de l'apprentissage d'un réseau RBF, deux problèmes se posent : la constitution de la première couche (choix du nombre de neurones, choix des centres C_i et de coefficients d'étalement σ_i), et la détermination des poids et biais de la seconde couche.

Répartition des prototypes

Le choix du nombre de neurones sur la première couche et de leurs prototypes correspondants dépend fortement de la réparatition des entrées, c'est à dire des vecteurs x_k de la base d'apprentissage. Pour établir ce choix, différentes méthodes sont applicables.

On peut répartir les prototypes uniformément ou aléatoirement dans l'espace d'entrée, ou bien sélectionner aléatoirement quelques vecteurs de la base d'apprentissage. Ces méthodes sont très simples, mais ne donnent pas de bons résultats car généralement la répartition obtenue n'est pas adaptée à la base. On peut alors utiliser un algorithme de clustering statistique, tel que les k-means, afin de mieux les répartir dans l'espace d'entrée.

K-means est un algorithme itératif permettant de séparer une série de vecteurs dans différents groupes (les clusters) et chaque groupe est représenté par un prototype, placé en son centre.

Le déroulement de l'algorithme k-means est le suivant : il faut tout d'abord choisir le nombre de prototypes désiré, empiriquement ou à l'aide de connaissances a priori sur la base d'apprentissage. Ensuite une série de prototypes initiaux est déterminée par l'une des méthodes précédentes (généralement un tirage aléatoire des exemples est utilisé). Puis chaque itération est réalisée en deux étapes. Chaque vecteur d'entrée de la base est classé dans un cluster en déterminant le prototype le plus proche, puis après avoir parcouru toute la base, la position de chaque prototype est recalculée, en prenant le barycentre de tous les exemples situés dans son cluster. L'algorithme s'arrête lorsque le système est stable, c'est à dire quand aucun vecteur ne change de cluster d'une itération à la suivante. La solution trouvée par l'algorithme k-means est une solution locale, c'est à dire qu'il ne garantit pas de trouver une solution optimale. Cependant, dans la plupart des cas pratiques la solution trouvée est satisfaisante, lorsque le nombre de prototypes décidés est adapté à la base.

Le choix des coefficients d'étalement va influer sur la forme de la fonction réalisée. S'ils sont trop faibles, on verra apparaître des pics autour des points définis par les prototypes, et au contraire

s'ils sont trop grands la fonction pourra difficilement prendre des valeurs différentes près de deux prototypes différents. Il est raisonnable de choisir une valeur qui soit de l'ordre de grandeur du carré de la distance entre deux prototypes voisins.

Apprentissage de la seconde couche

Une fois les prototypes placés, il reste à déterminer les paramètres de la seconde couche, c'est à dire les poids w_i et les biais b_i qui minimisent la fonction de performance du réseau. On peut calculer la sortie de la première couche pour chaque exemple de la base d'apprentissage :

$$z_{k,j} = \exp(-\frac{\|x_k - C_j\|^2}{2\sigma_j^2}) \tag{1.31}$$

où k est le numéro de l'exemple et j celui de la composante de z_k calculée, c'est à dire le numéro du neurone de la première couche dont on calcule la sortie. Le critère à minimiser est l'erreur quadratique moyenne du réseau sur la base d'apprentissage et a donc pour équation :

$$\epsilon = \frac{1}{p}\sum_{k=1}^{p}\sum_{i=1}^{r}(s_i(x_k) - y_{k,i})^2 = \frac{1}{p}\sum_{k=1}^{p}\sum_{i=1}^{r}(z_k.w_i - y_{k,i})^2 \tag{1.32}$$

où $s_i(x_k)$ est la sortie du neurone i de la seconde couche, $y_{k,i}$ est la $i^{ème}$ composante de y_k (c'est à dire la sortie voulue pour l'exemple k sur le neurone i de la seconde couche) et r est le nombre de sorties. p est le nombre d'éléments dans la base d'apprentissage (x_k, y_k). Un algorithme des moindres carrés peut être appliqué afin de trouver les paramètres w_i et b_i qui minimisent ce critère.

1.6 Mise en oeuvre des réseaux neuronaux

Nous allons suivre une démarche reprise par *Wierenga* et *Kluytmans* [17] qui est composé de quatre étapes principales :

1.6.1 Etape 1 : fixer le nombre de couches cachées

Mise à part les couches d'entrée et de sortie, l'analyste doit décider du nombre de couches intermédiaires ou cachées. Sans couche cachée, le réseau n'offre que de faibles possibilités d'adaptation. Avec une couche cachée, il est capable, avec un nombre suffisant de neurones, d'approximer toute fonction continue [18]. Une seconde couche cachée prend en compte les discontinuités éventuelles.

1.6.2 Etape 2 : Déterminer le nombre de neurones par couches cachées

Chaque neurone supplémentaire permet de prendre en compte des profils spécifiques des neurones d'entrée. Un nombre important permet donc de mieux coller aux données présentées mais diminue la capacité de généralisation du réseau. Ici non plus, il n'existe pas de règle générale mais des règles empiriques. La taille de la couche cachée doit être :

- Soit égale à celle de la couche d'entrée [17].

- Soit égale à 75% de celle-ci [19].

- Soit égale à la racine carrée du produit du nombre de neurones dans la couche d'entrée et de sortie [20].

Notons que ce dernier choix réduit le nombre de degrés de liberté laissés au réseau, et donc la capacité d'adaptation sur l'échantillon d'apprentissage, au profit d'une plus grande stabilité/capacité de généralisation.

Une voie de recherche ultèrieure consisterait soit à procéder à l'estimation d'un réseau comportant de nombreux neurones puis à le simplifier par une règle d'apprentissage éliminant les neurones inutiles ; soit à définir une architecture tenant compte de la structure des variables identifiée au préalable par une analyse en composantes principales.

1.6.3 Etape 3 : Choisir la fonction d'activation

En général, on considère la fonction logistique par le passage de la couche d'entrée à la couche cachée. Le passage de cette dernière à la couche de sortie serait soit linéaire, soit logistique selon les types de variables.

1.6.4 Etape 4 : Choisir l'apprentissage

L'apprentissage de rétro-propagation nécessite la détermination du paramètre d'ajustement des poids synaptiques à chaque itération. La détermination du critère d'arrêt est aussi crucial dans la mesure où la convergence peut passer par des minima locaux.

Chapitre 2

Approximation de fonctions par les réseaux de neurones

2.1 L'approximation

Lors d'un apprentissage supervisé, on veut faire correspondre à chaque vecteur d'entrée, un vecteur de sortie.

En d'autres termes, le réseau doit réaliser une fonction vectorielle voulue. Si cette fonction est connue analytiquement, il s'agit d'une tâche d'approximation de fonction. Si on ne dispose que d'un certain nombre de mesures (souvent entâchées de bruit), il s'agit d'une tâche de régression.

2.1.1 Approximateurs universels

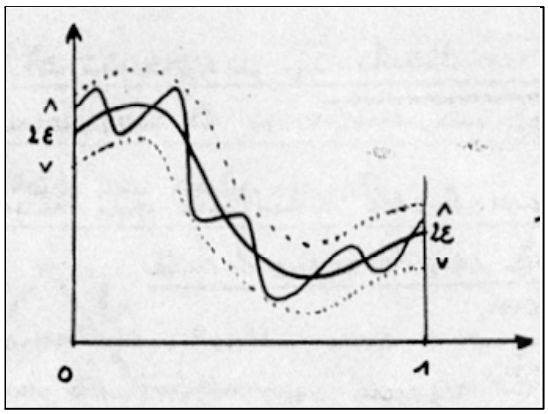
Les approximateurs universels les plus connus sont les polynômes.

En effet, n'importe quelle fonction suffisament continue de plusieurs variables peut-être approchée avec une précision arbitraire (sur un domaine fini de l'espace des variables) par une fonction polynôme.

Cette fonction polynôme peut être déterminée par le développement limité si la fonction est connue, ou par une régression polynômiale si la fonction est inconnue. En fait, l'exemple classique est l'approximation de fonctions d'une variable réelle par des polynômes :

Théorème de Weistrass-Stone

Toute fonction continue sur $[0, 1]$ *peut être approchée uniformément par un polynôme.* figure 2-1.



Figure˜2-1: Approximation d'une fonction f

(Dans une largeur 2ε autour d'une fonction f, on peut toujours trouver un polynôme)

L'approximation de fonction d'une variable réelle peut aussi se faire par des séries de *Fourier*.

- La série de *Taylor* d'une fonction "f" autour d'un point x_0 est :

$$f(x) = \sum_{i=1}^{\infty} a_i (x - x_0)^i \tag{2.1}$$

où les constantes a_i dépendent de la fonction f et de ses dérivées en x_0.

- De même, la fonction "f" peut avoir un développement en série de Fourier de la forme :

$$f(x) = \sum_{i=1} (a_i \cos(ix) + b_i \sin(ix)) \tag{2.2}$$

Les sommes partielles d'ordre n de ces deux exemples peuvent être associées à un réseau de neurones artificiel, comme par exemple :

Ces deux simples exemples montrent que les réseaux de neurones peuvent être utilisés comme des approximateurs universels de fonctions.

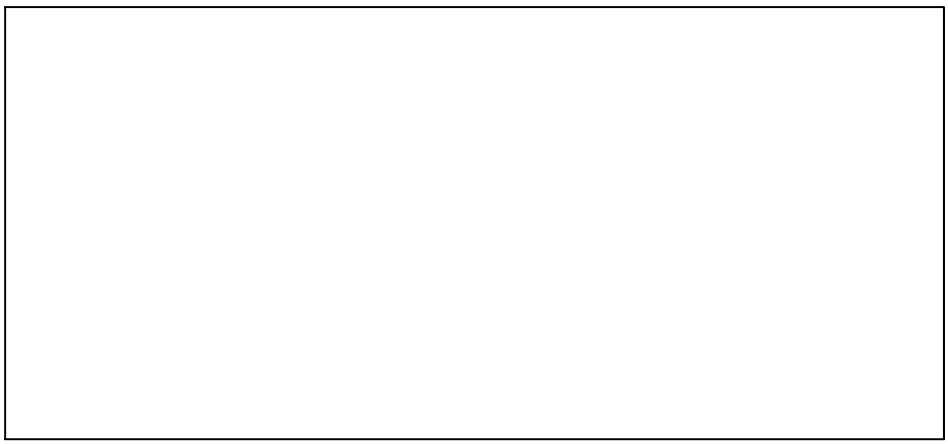
Cependant, il y a lieu de noter que dans l'approximation classique, la fonction f est connue explicitement alors que le réseau de neurones artificiel est "une boite noire" avec des données à l'entrée et des sorties à comparer à des valeurs désirées.

Dans ce cas, on doit approximer une fonction vraie mais inconnue.

Il y a lieu de préciser :

-Quel type de fonctions peut-on approximer par les réseaux de neurones artificiels ?

-Quelle est la complexité des calculs nécessaires ?



Figure˜2-2: Un réseau de *Taylor*

La réponse à la conjecture posée par le $13^{ème}$ problème, de la fameuse liste des 23 problèmes ouverts qu'*Hilbert* [3] a supposé être de la plus grande importance pour le développement des mathématiques, allait fournir une réponse à la première interrogation.

2.1.2 Le $13^{ème}$ problème d'*Hilbert*

Au Congrés International des mathématiques de Paris en 1900, *D. Hilbert* [3] dressa une liste de 23 problèmes non encore résolus et qui doivent constituer un défi à relever pour la communauté des mathématiciens du $20^{ème}$ siècle.

Le $13^{ème}$ problème traite d'un ancien problème de résolution d'équations algébriques : trouver une formule simple pour exprimer les racines d'un polynôme de degré n en fonction des coefficients.

La conjecture d'*Hilbert* était que :

"*Il est probable que la racine du polynôme à coefficients réels de degré 7 soit une fonction de ses coefficients qui n'appartienne pas à la classe de fonctions représentables par des méthodes nomographiques*".

C'est à dire qu'elle ne peut pas être représentée par une composition d'un nombre fini de fonctions continues de deux variables.

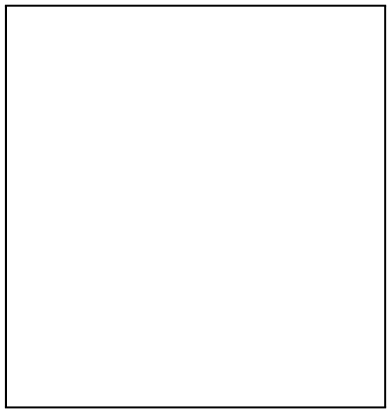
Pour être explicite, on doit montrer que l'équation :

$$f^7 + xf^3 + yf^2 + zf + 1 = 0 \tag{2.3}$$

de degré 7 ne peut pas être résolue en utilisant des fonctions d'un argument (d'une variable).

Cette formulation abstraite peut être mieux comprise en la comparant à des problèmes similaires plus simples.

Les racines de l'équation quadratique : $ax^2+bx+c=0$, par exemple peuvent être calculées par la composition d'un nombre fini d'opérations algébriques, exclusivemnet des opérations arithmétiques élementaires et des fonctions racine carrée, comme le montre le diagramme ci-après :



Figure~2-3: Réseau pour le calcul de des racines de l'équation quadratique

Des formules analytiques similaires pour résoudre les équations algébriques de degré 3, 4 et 5 étaient connues depuis longtemps.

Abel [21] a montré que pour les équations algébriques de degré $n \geq 5$, celà n'était pas possible.

Donc pas de formule algébrique pour trouver les racines pour degré $n \geq 5$. Par contre, les racines de l'équation de degré 7 peuvent être obtenues par un nombre fini de compositions de fonctions non-algébriques.

Une méthode trés populaire, avant l'avancement des ordinateurs, était la méthode nomographique (abaques ou nomogrammes) qui utilisait la représentation graphique des fonctions pour l'approximation des fonctions.

Exemple de résolution de $F(u,v,w) = uvw$

On peut écrire $uvw = mult(mult(u,v),w)$ et par la méthode nomographique, calculer uvw pour $u=3$, $v=1$ et $w=1,5$ (figure 2-4).

Par la méthode nomographique (abaque), il est possible de trouver une approximation de la solution d'une équation donnée.

Cependant, si la conjecture d'*Hilbert* était vraie, cela signifiait que pour les équations algébriques de degré $n \geq 7$, on ne pourra pas trouver une formule à base d'un nombre fini de compositions de

Figure˜2-4: Méthode nomographique pour le calcul de uvw

fonctions de deux variables et par conséquent aucune méthode de composition nomographique ne donnera la solution.

Cela donne une généralisation du théorème d'*Abel* [21].

2.1.3 Le théorème de *Kolmogorov-Arnold-Sprecher*

En 1957, *Kolmogorov* [2] a montré que la conjecture d'*Hilbert* était fausse. *Arnold* [22] arriva au même résultat à la même époque.

Il montra qu'une fonction continue de n variables peut toujours être représentée par un nombre fini de compositions de fonctions d'une variable.

La multiplication, par exemple, peut-être réécrite comme la composition de fonctions d'une variable et d'additions, puisque :

$$xy = \exp(\ln x + \ln y) \tag{2.4}$$

Une version moderne du théorème de *Kolmogorov* est :

Théorème 2.1 : *Soit* $f : I^n = [0,1]^n \to [0,1]$, $n \geq 2$, *une fonction continue.*

Il existe une fonction d'une variable g_q *et* Φ_q *pour* $q = 1, ..., 2n+1$ *et des constantes* λ_p *pour* $p = 1, ..., n$ *tel que :*

$$f(x_1, x_2, ..., x_n) \simeq \sum_{q=1}^{q=2n+1} g_q \left(\sum_{p=1}^{p=n} \lambda_p \Phi_q(x_p) \right) \tag{2.5}$$

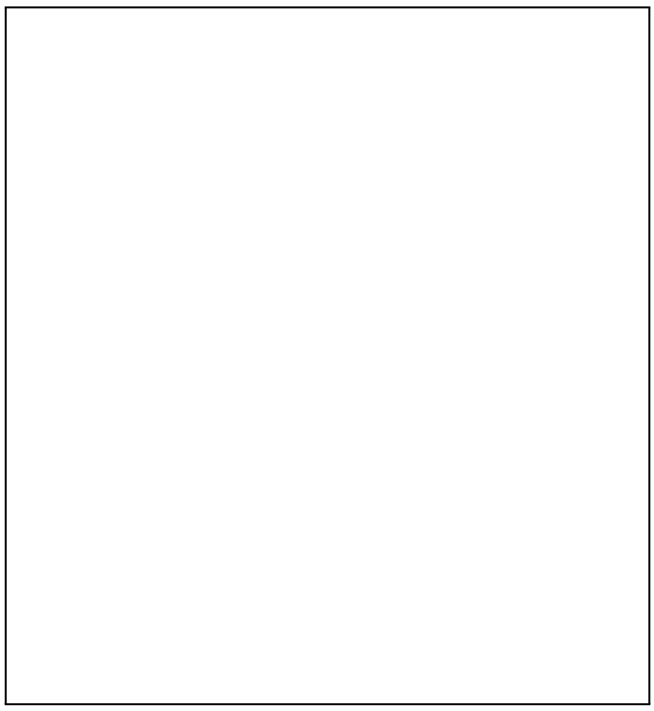
où g_q *sont des fonctions continues d'une variable correctement choisies, et* $\lambda_p \Phi_q$ *sont des fonctions*

continues croissantes indépendantes de f.

Du point de vue des réseaux de fonctions, le théorème de *Kolmogorov* peut être interprété comme affirmant n'importe quelle fonction continue de n variables peut être représentée par un réseau fini de fonctions d'une variable où seule l'addition est fonction de plusieurs variables.

Le réseau est fonction de plusieurs variables.

Le réseau ci-après est un exemple pour la représentation de $f(x_1, ..., x_n)$:



Figure~2-5: Réseau pour calculer une fonction continue f

L'idée fondamentale du théorème est que l'on peut exprimer une fonction continue sur un ensemble compact en termes de sommes et de compositions d'un nombre fini de fonctions d'une variable.

Le théoreme de *Kolmogorov* montre que chaque fonction continue à pusieurs variables peut-être calculée par un réseau à 2 couches cachées et dont les unités calculent les fonctions continues (les fonctions g_q et Φ_{pq})

En 1965, *Sprecher* [4] donna une formulation simplifiée du théoreme de *Kolmogorov.*

2.1.4 Théorème de *Sprecher*

Pour chaque entier $n \geq 2$, il existe une fonction réelle, monotone croissante $\psi(x)$, avec $\psi([0,1]) = [0,1]$ dépendant de n et ayant la propriété suivante :

Pour tout $\delta > 0$, il existe un nombre rationnel ε, $0 < \varepsilon < \delta$, tel que pour toute fonction continue de n variables à valeurs réelles, définie sur I^n peut être représentée par :

$$f(x) = \sum_{j=1}^{j=2n+1} \chi \left(\sum_{i=1}^{n} \lambda^i \psi(x_i + \varepsilon(j-1)) + j - 1 \right) \tag{2.6}$$

où la fonction χ est une fonction réelle, continue et λ une constante indépendante de f.

2.2 Approximation des fonctions par le PMC

En 1987, *Hecht Nielsen* [23] fût le premier à remarquer que le théorème de *Kolmogorov* pouvait être interprété en termes de réseaux de neurones .

Le neurone étant un dispositif avec n entrées (x_1, x_2, ..., x_n) et une sortie y

$$y = g(w_1 x_1 + ... + w_n x_n + w_0) \tag{2.7}$$

avec g la fonction d'activation, $w_{i=1,...,n}$ les poids et w_0 le seuil.

Si on envoie les sorties de quelques neurones vers les entrées d'autres neurones, on obtient alors un réseau de neurones.

Hecht-Nielsen a ainsi noté que le théorème de *Kolmogorov* décrit le réseau suivant :

La première couche est composée de $n(2n+1)$ neurones N_{ij} $1 \leq i \leq n$, $0 < j < 2n+1$ avec une fonction d'activation $\Psi(x)$ et des poids $\omega_i = 1, \omega_0 = -\varepsilon j$

La deuxième couche de $2n+1$ neurones avec une fonction d'activation χ, des poids λ^i et un seuil $\omega_0 = -j$

La troisième couche consiste en un neurone linéaire qui ne fait que sommer les entrées.

Ainsi le théorème, en réalité, prouve qu'une fonction arbitraire peut être représentée par un réseau à trois couches. (2 couches cachées et celle de sortie fait la somme)

Dans le théorème de *Kolmogorov*, pour toute fonction f, il y a une seule fonction qui ajuste f parfaitement ($\varepsilon = 0$).

On veut être capable, pour f donnée et $\varepsilon > 0$, de générer un réseau de neurone pour lequel la fonction $\widetilde{f}$ est ε proche de f, c.a.d pour laquelle pour tout $x_i \in [0,1]$:

$$\widetilde{f}(x_{1,}x_{2.....,}x_{n)} \in [f(x_{1,}x_{2.....,}x_{n)} - \varepsilon, f(x_{1,}x_{2.....,}x_{n)} - \varepsilon] \tag{2.8}$$

L'existence d'un réseau de neurone qui approxime n'importe quelle fonction avec une précision donné, a été prouvé par *Hornik et al.* (mars 1989) [24] en utilisant le théorème de *Weistrass-Stone*.

Il a démontré que la famille des réseaux neuronaux à une seule couche de neurones cachés est un approximateur universel pour toute fonction mesurable.

En fait, ce résultat tient en particulier pour toutes les fonctions continues sur un compact (voir annexe A).

En septembre 1989**,** *Federico Girosi* et *Tomaso Poggio* [25], ont critiqué l'interprétation du théorème de *Kolmogorov* comme étant inapproprié aux réseaux de neurones en soulignant que les fonctions g et ϕ du théorème de *Kolmogorov* sont très fortement non lisses (leurs graphes sont fractales) et ainsi sont très difficiles à calculer.

Le fait que les fonctions g et ϕ soient lisses est important car la représentation doit être lisse afin de la généraliser et pour qu'elle soit stable contre les bruits.

Ils ont utilisé certains résultats de *Vitushkin* (1954,1977) [26] [27] et de *Henkin* [28] pour montrer que les fonctions intérieures Ψ et χ du théorème de *Kolmogorov* sont fortement non lisses.

Théorème de *Vitushkin* (1954) [26]:

Ce théorème montre que malgré que les fonctions Ψ soient continues, elles sont fortement non lisses :

Il y a des fonctions r ($r = 1, 2, .$) fois continument dérivables de $n \geq 2$ variables non représentables par la superposition de fonctions r fois continuement dérivables de moins de n variables.

Il y a des fonctions r fois continuement dérivables de deux variables qui ne sont pas représentables par des sommes de fonction continuement dérivables à une variable.

En juillet 1991, *V. Kurkova* [29] a montré que le théorème de *Kolmogorov* pour la représentation des fonctions à n variables par les sommations et les superpositions des fonctions continues à une variable est approprié dans le contexte des réseaux de neurones.

On constate que si l'on remplace l'égalité de l'équation du théorème de *Kolmogorov* par une approximation seulement, on peut éliminer les difficultés évoquées par *Girosi* et *Poggio* [25].

En mathématiques, les fonctions fortement non lisses rencontrées, sont généralement constantes comme des limites ou des sommes de séries infinies de fonctions lisses.

Dans le contexte des réseaux de neurones, on s'intéresse aux approximateurs de fonctions, le

seul problème concernant la pertinence du théorème de *Kolmogorov* est : comment la construction de *Kolmogorov* peut être modifiée de telle façon que toutes les fonctions à une variable sont des limites de fonctions lisses utilisées dans les réseaux du type perceptron.

Les fonctions utilisées dans les réseaux du type perceptron sont des combinaisons linéaires finies de compositions de transformations affines de la ligne des réelles E avec une certaine fonction sigmoïdale (une fonction $\sigma : E \to [0,1]$ avec $\lim \sigma(t) = 0 (t \to -\infty)$ et $\lim \sigma(t) = 1 (t \to +\infty)$). On les appelle les fonctions escalier du type sigmoïdal.

Les constructions de *Kolmogorov* des fonctions g et ϕ sont parfaitement ajustées pour les fonctions escalier de n'importe quel type sigmoïdal.

Théorème 1 (*Kurkova* 1991) [29]:

Soit n, m des nombres naturels avec $n \geq 2$, $m \geq 2n+1$, et $\sigma : E \to [0,1]$ une fonction sigmoïdale quelconque.

Alors il existe des nombres réels $\omega_{pq} (p = 1....n, q = 1....m)$ et des fonctions Ψ_q qui sont des limites de fonctions escalier du type σ uniformément convergente qui pour chaque fonction continue $f : [0,1]^n \to E$, il existe une fonction continue $\varphi : E_1 \to E_1$ qui est une limite d'une fonction escalier du type σ uniformément convergente tel que pour tout $(x_1.....x_n) \in [0,1]^n$:

$$f(x_1, x_2, ..., x_n) = \sum_{q=1}^{q=m} \varphi \left(\sum_{p=1}^{p=n} \omega_{pq} \Psi_q(x_p) \right) \tag{2.9}$$

Théorème 2 (*Kurkova* 1991) [29]:

Soit $n \geq 2$ un nombre naturel, $\sigma : E \to [0,1]$ une fonction sigmoïdale et $f : [0,1]^n \to E$ une fonction continue

Alors il existe un nombre naturel k , et une fonction escalier du type σ Ψ_{pi} $\varphi_i (i = 1...k, p = 1...n)$, tel que pour tout $(x_1.....x_n) \in [0,1]$ on a

$$\left| f(x_1, x_2, ..., x_n) - \sum_{i=1}^{k} \varphi_i \left(\sum_{p=1}^{p=n} \Psi_i(x_p) \right) \right| < \epsilon \tag{2.10}$$

Ce théorème établit que toute fonction continue peut être approchée par un perceptron avec 4 couches.

Kurkova a pu donner aussi le nombre de couches du perceptron, où elle a estimé le nombre de

neurones à $nm(m+1)$ et $m^2(m+1)^n$ dans la $2^{ème}$ et la $3^{ème}$ couche.

Ainsi, *Kurkova* a soutenu l'appropriation du théorème de *Kolmogorov* aux réseaux de neurones en montrant que les fonctions non lisses peuvent être approximées par des sommes de séries infinies de fonctions lisses. Donc, il est possible de calculer approximativement ψ et χ avec un réseau paramètré.

D'autres résultats tels que (*Funahashi* 1989 [30], *Hecht - Nielsen* 1989 [31], *Hornik et Al.* 1989 [24], *Cybenko* 1989 [32], *Caroll* et *Dickinson* 1989 [33], *Stinchombe* et *White* 1989 [34], 1990 [35], *Hornik* 1991 [18]) établissent que trois couches suffisent pour l'approximation.

2.3 Les réseaux de neurones à une couche cachée sont des approximateurs universels

Des preuves mathématiques rigoureuses pour l'universalité des réseaux de neurones multicouches feed-forward utilisant des unités d'activation, aussi bien de type sigmoïdal que d'autres types plus généraux, ont été données indépendement par *Cybenko* [32], *Hornik* et *al.* [24] et *Funahashi* [30].

La preuve de *Cybenko* est distinguée par étant mathématiquement brève et élégante. Elle est basée sur le théorème de *Hahn-Banach* (*Luenberger*, 1969 [36]).

Ci-dessous est la déclaration du théorème de *Cybenko* :

Soit φ une fonction continue de type sigmoïdale (exemple : $\varphi(\xi) = \frac{1}{1+e^{-\xi}}$)

Alors, pour toute fonction à valeurs réelles sur $[0,1]^n$ donnée, (ou tout autre sous-ensemble compact sur R_n) et $\varepsilon > 0$, il existe des vecteurs W_1, W_2, ..., W_N et une fonction paramètrée $G : [0,1]^n \to R$ telle que :

$$|G(X,W) - f(X)| < \varepsilon \qquad \text{pour tout } X \in [0,1]^n \tag{2.11}$$

où

$$G(X,W) = \sum_{j=1}^{N} \alpha_j \varphi(W_j^T X + \theta_j) \tag{2.12}$$

avec : $W_j \in R^n$ et α_j, $\theta \in R$ sont fixés.

Des résultats de *Hornik* et *al.* (1989) [24] utilisant le théorème de *Stone-Weierstrass*, et *Funahashi* (1989) [30] utilisant une formule d'intégrale d'*Irie* et *Miyake* (1988) [37], ont indépendement prouvé des théorèmes similaires établissant qu'une seule couche cachée d'un réseau de neurones feed-forward est capable d'approximer uniformément toute fonction continue, avec un degré de

précision désiré.

Ceci implique que tout manque de succés dans les applications doit surgir d'un choix inadéquat de paramètres ou un nombre insuffisant de noeuds cachés.

Dans le cas du théorème de *Cybenco*, l'hypothèse sur la fonction d'activation est qu'elle a pour limite 0 en $-\infty$ et 1 en $+\infty$, dans celui de *Funahashi* qu'elle est croissante, non constante et bornée.

Les fonctions non continues, mais mesurables, peuvent aussi être approchées, mais dans un sens moins fort .

Il existe par ailleurs quelques résultats sur le nombre de neurones requis pour approcher une fonction avec une précision fixée, pour certaines classe de fonctions.

Ces résultats affirment donc, pour toute fonction déterministe usuelle, l'existence d'une approximation par un réseau de neurones.

Les réseaux complètement connectés ou à couches, et à neurones cachés sigmoïdaux, remplissent les conditions d'application des théorèmes.

Hornik et *al.* (1990) [38] ont prouvé un autre résultat important, concernant la capacité d'un réseau feed-forward utilisant des fonctions d'activation de type sigmoïdal. Ces réseaux peuvent non seulement approximer des fonctions inconnues, mais peuvent aussi approximer leurs dérivées. En effet, *Hornik* et *al.* ont montré aussi que ces réseaux peuvent approximer des fonctions qui ne sont pas différentiables dans le sens classique mais possédent une dérivée généralisée comme dans le cas des fonctions escalier (piecewise) différentiable.

Utilisant le théorème de *Sun* et *Cheney* (1992) [39], *Light* (1992) [40] a étendu le résultat de *Cybenko* à toute fonction continue f sur R^n, il montra que la fonction sigmoïdale peut être remplacée par toute fonction continue g sur R qui satisfait :

1. $\int_0^{\infty} \left| g(\xi)\xi^{n-1} \right| dt < \infty$
2. $\int_{-\infty}^{+\infty} g(\xi)\xi^{2j} dt = 0 \qquad 0 < j \leq \frac{n-3}{2} \qquad n > 3$
3. $\int_{-\infty}^{+\infty} g(\xi)\xi^{n-1} dt \neq 0$

Un exemple de fonctions d'activation, satisfaisant les conditions ci-dessus, est donné par la famille de fonctions :

$$g_n(\xi) = \frac{1}{(1+\xi^2)^n} \cos\left[n \cos^{-1} \frac{1}{\sqrt{1+\xi^2}} \right] \tag{2.13}$$

L'approximation universelle des réseaux de neurones à une couche cachée avec des fonctions d'activation non-sigmoïdales a été prouvée par *Stinchombe* et *White* (1989) [34].

En 1991, *Hornik* [18] a prouvé qu'une condition suffisante pour que l'approximation universelle puisse être obtenue en utilisant des fonctions d'activation continues, bornées et non-constantes.

Ito (1991) [41] a montré que toute fonction appartenant à la classe de fonctions continues qui décroient rapidement dans R^n (les fonctions $f(x)$ satisfaisant $\lim_{|x|\to\infty} \left| x_1^{k_1} x_2^{k_2} ... x_n^{k_n} f(x) \right| = 0$ pour tout $k_j > 0$) peut être approximée par un réseau de neurones à deux couches.(une couche cachée)

Théorème d'approximation universel *Soit $\phi(.)$ une fonction continue non-constante, bornée et monotone croissante. Soit $I^p = [0,1]^p$. Soit $C(I^p)$ l'espace des fonctions continues sur I^p. Soit $f \in C(I^p)$ et $\varepsilon > 0$ donné, il existe alors un entier M et des constantes réelles α_i, θ_i et w_{ij} pour $i = 1, ..., M$ et $j = 1, ..., p$, et une fonction F définie par :*

$$F(x_1, ..., x_p) = \sum_{i=1}^{M} \alpha_i \phi \left(\sum_{j=1}^{p} w_{ij} x_j - \theta_i \right) \tag{2.14}$$

est une approximation de la fonction $f(.)$ telle que :

$$|F(x_1, ..., x_p) - f(x_1, ..., x_p)| < \varepsilon \tag{2.15}$$

pour tout $(x_1, ..., x_p) \in I^p$.

Ce théorème s'applique directement au PMC.

La fonction logistique utilisée $\frac{1}{[1+\exp(-V)]}$ vérifie les conditions de ϕ.

L'équation 2.15 ci-dessus décrit la sortie d'un PMC avec :

1. Le réseau possède p cellules d'entrée et une couche cachée avec M neurones; les données d'entrée sont : $x_1, ..., x_p$.

2. Le neurone caché i a comme poids $w_{i1}, ..., w_{ip}$ et un biais de θ_i.

3. La sortie du réseau est une combinaison linéaire des sorties des neurones cachés, avec $\alpha_1, ..., \alpha_M$ comme coefficients de la combinaison.

Ce théorème est un théorème d'existence dans la mesure où il fournit une justification mathématique pour l'approximation d'une fonction quelconque continue contrairement à la représentation exacte de la fonction tel que décrit initialement par le théorème de *Kolmogorov*.

L'équation 2.15 généralise l'approximation par les sommes partielles des séries de *Fourier*.

Le théorème ne dit pas qu'une seule couche cachée est optimal en ce qui concerne le temps d'apprentissage ou la facilité d'implémentation. L'algorithme d'apprentissage du PMC, la rétropropagation, est NP complet (voir chapitre 4).

La précision de la meilleure approximation nécessite un nombre de neurones cachés M grand.

L'efficacité des données empiriques d'entrée pour ajuster au mieux l'approximation nécessite un rapport M/N petit, N étant le nombre de vecteurs utilisés par l'apprentissage du réseau.

Si l'on désigne par f_M l'approximation théorique fournie par le réseau choisi et f_M^N la sortie obtenue après apprentissage sur N vecteurs d'entrée, on a les deux quantités à considérer :

$$\begin{aligned} E_1 &= f - f_M \qquad \text{appelé l'erreur d'approximation} \\ E_2 &= f_M - f_M^N \qquad \text{appelé l'erreur de l'estimation} \end{aligned} \tag{2.16}$$

Dans l'approche statistique des réseaux de neurones du PMC, minimiser E1 et E2 donne lieu au dilemme biais/variance.

2.4 Propriété fondamentale des réseaux de neurones non bouclés à apprentissage supervisé : L'approximation parcimonieuse

Propriété 1*:*

Toute fonction suffisament régulière peut être approchée uniformément, avec une précision arbitraire, dans un domaine fini de l'espace de ses variables, par un réseau de neurones comportant une couche de neurones cachés en nombre fini, possédant tous la même fonction d'activation, et un neurone de sortie linéaire [38], [18].

Cette propriété qui n'est qu'un théorème d'existence et ne donne pas de méthodes pour trouver les paramètres du réseau, n'est pas spécifique aux réseaux de neurones.

C'est la propriété suivante qui leur est particulière et en fait tout leur intérêt :

Propriété 2:

Les réseaux de neurones non linéaires par rapport à leurs paramètres sont des approximateurs parcimonieux.

Dans la pratique, le nombre de fonctions nécessaires pour réaliser une approximation est un critère important dans le choix d'un approximateur de référence à un autre.

Le concepteur du modèle doit toujours faire en sorte que le nombre de paramètres ajustables soit le plus faible possible : on dit que l'on cherche l'approximation la plus parcimonieuse.

Propriété fondamentale*:*

*On montre (*Barron *(1993) [42]) que si l'approximation dépend des paramètres ajustables de manière non linéaire, elle est plus parcimonieuse que si elle dépend linéairement des paramètres.*

Plus précisement, on montre que le nombre de paramètres, pour une précision donnée, croit exponentiellement avec le nombre de variables dans le cas des approximateurs linéaires par rapport à leurs paramètres, alors qu'il croit linéairement avec ce nombre pour les approximateurs non linéaires par rapport à leurs paramètres.

La parcimonie est donc d'autant plus importante que le nombre d'entrées du modèle est grand : pour un modèle à 1 ou 2 entrées, on peut utiliser indifférement un modèle linéaire par rapport à ses paramètres (polynôme par exemple) ou un modèle non linéaire par rapport à ses paramètres (réseau de neurones par exemple).

Or, la sortie des réseaux de neurones à fonction d'activation sigmoïde n'est pas linéaire par rapport aux poids du réseau, alors que la sortie des réseaux de RBF à centres et écarts-types fixés est linéaire par rapport aux poids. De même, une approximation par polynôme est linéaire par rapport aux coefficients des monômes.

Ainsi, l'utilisation de neurones cachés à fonction d'activation sigmoïde permet une approximation plus parcimonieuse qu'une approximation polynômiale, où qu'une approximation par fonctions radiales à centres et écarts-types fixes.

Si en revanche, on considère que les centres et écarts types des RBF gaussiennes sont des paramètres ajustables au même titre que les poids des connexions, il n'y a pas à l'heure actuelle, un avantage mathématiquement démontré à utiliser un type de neurones plutôt qu'un autre.

En revanche, des arguments pratiques décisifs peuvent justifier une préférence : connaissance à priori sur le type de non-linéarité souhaitable, rapidité de calcul, facilité d'initialisation de l'apprentissage, etc.

Expliquons qualitativement l'origine de la parcimonie. Considérons un modèle linéaire par rap-

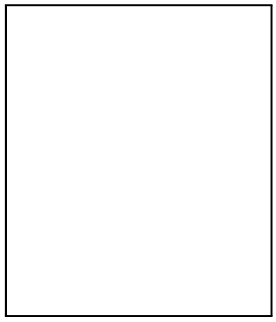
port à ses paramètres , un modèle polynômial par exemple :

$$g(x) = 4 + 2x + 4x^2 - 0,5x^3$$

Le résultat $g(x)$ du modèle est une combinaison linéaire des fonctions : $y = 1$, $y = x$, $y = x^2$, $y = x^3$, avec les poids $w_0 = 4$, $w_1 = 2$, $w_2 = 4$, $w_3 = -0,5$. Ces fonctions ont une forme qui est fixée une fois pour toutes.

Considérons à présent le modèle neuronal représenté sur la figure 2-6, dont l'équation est :

$$g(x) = 0,5 - 2th(10x + 5) + 3th(x + 0,25) - 2th(3x - 0,25)$$



Figure˜2-6: Réseau de neurones non bouclé à 1 variable

Le résultat de ce modèle est aussi une combinaison linéaire des fonctions : $y = 1, y = th(10x+5)$, $y = th(x + 0,25)$, $y = th(3x - 0,25)$. Mais la forme de ces fonctions dépend des valeurs des poids des connexions entre les entrées et les neurones cachés.

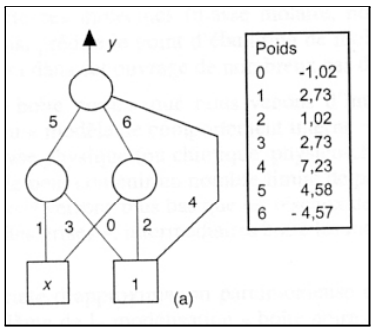
Ainsi, au lieu de combiner des fonctions de formes fixes, on combine des fonctions dont la forme elle-même est ajustée par des paramètres.

On comprend facilement que ces degrés de liberté supplémentaires permettent de réaliser une fonction donnée avec un plus petit nombre de fonctions élémentaires, ce qui est précisement la définition de la parcimonie.

Exemple élémentaire :

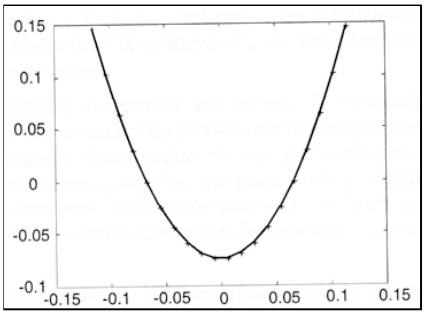
Considérons la parabole d'équations : $y = 16,71x^2 - 0,075$. Nous en prenons 20 échantillons régulièrement espacés, que nous utilisons pour effectuer un apprentissage supervisé d'un réseau à 2 neurones cachés (à fonction d'activation Arctg) représenté sur la figure 2-7 :

Un apprentissage à l'aide de l'algorithme de *Levenberg-Marquardt* fournit, en quelques dizaines



Figure˜2-7: Réseau de l'interpolation

d'itérations les paramètres indiqués sur la figure 2-7.



Figure˜2-8: Points d'apprentissage et sortie du réseau

La figure 2-8 représente les points de l'ensemble d'apprentissage et la sortie du réseau, qui passe par ces points avec une excellente précision.

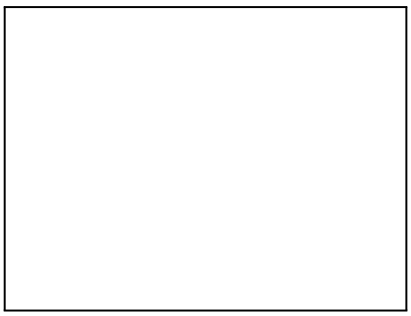
La figure 2-9 représente les sorties des neurones cachés, dont la combinaison linéaire avec le biais constitue la sortie du réseau.

La figure 2-10 montre les points d'un ensemble de tests et la sortie du réseau.

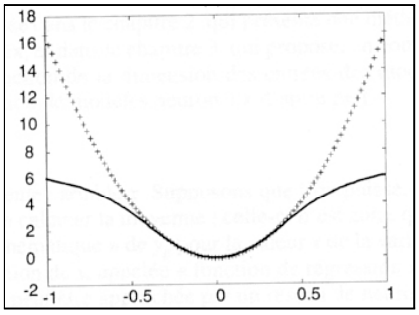
Lorsqu'on s'éloigne du domaine d'apprentissage $[-0,12,+0,12]$, la précision de l'approximation se dégrade, ce qui est normal. On notera la symétrie dans les valeurs des paramètres, qui reflète la symétrie du problème (simulation réalisée à l'aide du logiciel Neuro one^{TM} de NETRALSA [15]).

Remarque :

Bien entendu, approcher une parabole à une variable par un réseau de neurones ne présente aucun intérêt pratique, puisque la parabole a deux paramètres alors que le réseau de neurones en a sept! La seule justification de cet exemple est que, étant mono-dimensionnel, il permet d'utiliser des représentations graphiques.



Figure˜2-9: Sortie des deux neurones cachés (sigmoïdes) après apprentissage.



Figure˜2-10: Test

Conclusion

Lorsqu'on cherche à modéliser un processus à partir des données, on s'efforce toujours d'obtenir les résultats les plus satisfaisants posibles avec un nombre minimum de paramètres ajustables. Dans cette optique *Hornik et al.* [44] ont montré que :

Si le résultat de l'approximation (c'est à dire la sortie du réseau de neurones) est une fonction non linéaire des paramètres ajustables, elle est plus parcimonieuse que si elle est une fonction linéaire de ces paramètres.

De plus, pour des réseaux de neurones à fonction d'activation sigmoïdales, l'erreur commise dans l'approximation varie comme l'inverse du nombre de neurones cachés. Elle est indépendante du nombre de variables de la fonction à approcher. Par conséquent, pour une précision donnée, donc pour un nombre de neurones cachés donné, le nombre de paramètres du réseau est proportionnel au nombre de variables de la fonction à approcher.

Ce résultat s'applique aux réseaux de neurones à fonction d'activation sigmoïdale puisque la sortie de ces neurones n'est pas linéaire par rapport aux poids synaptiques. Cette propriété montre l'intérêt des réseaux de neurones par rapport à d'autres approximateurs comme les polynômes dont

la sortie est une fonction linéaire des paramètres ajusatbles. Pour un même nombre d'entrées, le nombre de paramètres ajustables à déterminer est plus faible pour un réseau de neurones que pour un polynôme.

2.5 L'approximation de fonctions par le RBF

Dans le réseau RBF, chaque unité dans la couche cachée a son propre centroïde, et pour chaque entrée $x = (x_1, ..., x_r)$, le réseau calcule la distance entre x et son centroïde. Les sorties sont des fonctions linéaires de cette distance.

Ainsi chaque noeud dans le réseau RBF calcule une sortie qui dépend d'une fonction radiale symétrique.

Supposons qu'il y a "r" noeuds d'entrées et "m" noeuds de sortie. La fonction de réponse globale a la forme suivante :

$$\sum_{i=1}^{M} w_i K(\frac{x - c_i}{\sigma_i}) = \sum_{i=1}^{M} w_i g(\frac{\|x - c_i\|}{\sigma_i}) \tag{2.17}$$

$M \in N$: ensemble de nombres naturels, est le nombre de noeuds dans la couche cachée ; $w_i \in R^m$: est le vecteur des poids du $i^{ème}$ noeud vers les noeuds sortant ; x est un vecteur d'entrée (1 élément de R) ; K est une fonction radialement symétrique d'une unité dans la couche cachée ; c_i et σ_i sont le centroïde et le facteur d'activation.

La fonction gaussiènne est souvent utilisée comme fonction d'activation, et les facteurs de lissage peuvent être les mêmes ou varier à travers les noeuds.

Propriété d'approximation : *(*Park et Sandberg 1991 [45]*)*

Pour toute fonction d'entrée sortie f, il existe un RBF avec un ensemble de centres $\{c_i\}$ et une largeur commune $\sigma > 0$ tel que la fonction réalisée par le RBF est proche de $f(x)$ pour la norme $L_p(p > 0)$.

Il a été montré que le réseau RBF est un approximateur universel, c'est à dire que le réseau est capable d'approximer n'importe quelle fonction avec une précision donnée pourvu que l'on fournisse un nombre suffisant de neurones, et que l'on utilise un algorithme d'apprentissage adéquat.

Ces réseaux approximent localement les fonctions. En effet, les neurones de la première couche ne produisent pas de valeurs significatives en sortie que dans une certaine zone de l'espace d'entrée (c'est à dire que la sortie est proche de 1 pour une entrée proche de leur prototype est proche de 0 ailleurs).

La seconde couche permet d'associer une certaine sortie à chaque zone. Les réseaux RBF sont particulièrement adaptés lorsque les vecteurs des entrées sont regroupés par zones, tel que dans le cas de l'algorithme des K-means. Dans cette situation, les RBF ont généralement de meilleures performances que les autres solutions neuronales, ou des performances équivalentes pour une complexité moindre.

Par contre, lorsque les vecteurs d'entrée sont répartis plus uniformément, ou lorsque le nombre de dimensions de l'espace d'entrée est important, le nombre de prototypes (centres) nécessaires augmente rapidement et les réseaux RBF perdent leur intérêt.

Enfin un autre avantage des réseaux RBF est leur rapidité d'apprentissage plus grande que pour le perceptron.

Comme le PMC, les RBF peuvent être utilisés dans le cas de problèmes d'approximation de fonctions, et sont capables d'approximer avec un degré de précision quelconque n'importe quelle fonction continue non linéaire.

La différence principale réside dans la nature sigmoïde, alors que les RBF utilisent des fonctions gaussiènnes.

Choisir entre PMC et RBF dépend de la nature du problème à traiter.

La propriété des approximateurs universels pour ces réseaux n'a été que récement prouvée pour des gaussiènnes radiales [46] et plus généralement pour des réseaux RBF [45].

Chapitre 3

Statistiques et réseaux de neurones

3.1 Introduction

Les développements récents dans le domaine des réseaux de neurones artificiels ont montré les liens théoriques étroits existant entre réseaux de neurones et statistiques (*Friedman* 1994 [47], *Vapnik* 1995 [48]).

En pratique, pour développer un modèle de réseaux de neurones, on doit préparer l'expérience, collecter des données, calculer l'estimateur, évaluer les performances, interpréter les résultats..., avec les mêmes précautions que quand on utilise un modèle statistique. A tel point qu'on pourrait presque se demander aujourd'hui, si les réseaux de neurones ne seraient pas tout simplement une classe particulière de modèles statistiques.

La table d'équivalence ci-dessous [49] semblerait même indiquer que la seule différence entre les réseaux de neurones et statistiques tiendrait au vocabulaire.

Réseaux de neurones	**Statistiques**
Apprentissage	Estimation
Poids	Paramètres
Connaissance	Valeurs des paramètres
Apprentissage supervisé	Régression
Classification	Discrimination
Apprentissage non supervisé	Estimation de densité
Clustering	Classification
Réseau de neurones	Modèle
Grand : 100 000 poids	Grand : 50 paramètres
Ensemble d'apprentissage	Echantillon

Tableau 3.1: **Glossaire réseaux de neurones/statistiques. Présentation de R.Tibshirani [45]**

3.2 Approximation de fonction et modélisation statistique

Les problèmes que l'on rencontre en pratique ne sont que très rarement des problèmes d'approximation de fonctions connues.

Dans la très grande majorité des cas, on cherche à établir un modèle à partir de mesures, ou, en d'autres termes, à trouver la fonction qui passe "au plus près" d'un nombre fini de points expérimentaux, généralement entachés de bruit.

On cherche alors à approcher la "fonction de régression" du processus considéré, c'est à dire la fonction que l'on obtiendrait en calculant la moyenne d'une infinité de mesures effectuées en chaque point du domaine de validité du modèle.

Le nombre de points de ce domaine étant lui-même infini, la connaissance de la fonction de régression nécessiterait donc une infinité de mesures en un nombre fini de points.

Les réseaux de neurones, en raison de leur propriété fondamentale, sont de bons candidats pour réaliser une approximation de la fonction de régression. C'est ce qui justifie l'utilisation réelle des réseaux de neurones : la recherche d'une approximation de la fonction de régression à partir d'un nombre fini de points.

L'utilisation des réseaux de neurones entre donc complètement dans le cadre de méthodes statistiques : les méthodes de recherche d'une approximation de la fonction de régression.

De telles méthodes ont été très largement développées pour les fonctions de régression linéaires.

L'apport des réseaux de neurones réside donc dans leur capacité à approcher des fonctions non linéaires.

Ainsi, en raison de leurs propriétés fondamentales, les réseaux de neurones sont capables d'intervenir dans la résolution de nombreux problèmes de modélisation et de classification à partir de mesures.

Ainsi, il peut être avantageux de les mettre en oeuvre pour toute application nécessitant de trouver, par des méthodes statistiques, une relation non linéaire entre les données numériques.

Il va de soi que les méthodes "neuronales" ont des limitations et que leur mise en oeuvre nécessite quelques précautions :

- Tout d'abord, nous avons vu que l'apport des réseaux de neurones réside dans leur capacité à réaliser des approximations de fonctions de régression non linéaires. Avant d'utiliser des réseaux de neurones dans une application, il faut donc s'assurer de la nécessité d'un modèle non linéaire.

En effet, la mise en oeuvre d'un modèle linéaire est généralement plus simple que celle d'un réseau de neurones.

- D'autre part, l'utilisation des réseaux de neurones (et plus généralement des méthodes statistiques) nécessite un échantillon représentatif de la population étudiée. L'apprentissage des réseaux de neurones montrera l'importance de la taille de l'échantillon.

3.3 Statistiques et théories de l'apprentissage

Un réseau de neurones, lorsqu'il apprend, ajuste ses poids synaptiques afin d'aboutir à une description d'un système (d'une fonction, d'une tâche, ...) qu'il utilise pour relier ses entrées et ses sorties. En cela, les réseaux ont de fortes ressemblances avec les techniques statistiques.

Il est souvent fort intéressant de comparer les formalismes et les terminologies à travers les disciplines. Là où le traitement de signal cherche à minimiser des fonctions de correlation, la physique et les mathématiques cherchent à minimiser des fonctions de coût ou d'énergie, les problématiques sont les mêmes. On peut donc être en droit de penser que les solutions, elles aussi, sont les mêmes. Dans de nombreuses applications, les réseaux de neurones se posent évidemment comme une alternative aux techniques statistiques. Un réseau de neurones, et l'apprentissage qui lui est associé, peuvent être caractérisés à l'aide des statistiques. Une règle d'apprentissage peut-être modélisée grâce aux outils statistiques. Une réseau qui a appris peut-être compris et expliqué à travers des grandeurs statistiques.

Un apprentissage est toujours réalisé avec un objectif, celui d'approximer, d'identifier, d'estimer, de classer,...etc. On cherche donc à obtenir le meilleur apprentissage possible, et pour cela, il faut pouvoir caractériser un processus d'apprentissage et analyser sa convergence.

Les méthodes statistiques qui permettent d'analyser une machine d'apprentissage sont regroupées sous la désignation de la théorie d'apprentissage.

3.3.1 Présentation

A partir d'une architecture donnée de réseaux de neurones, il est possible d'engendrer une famille de fonctions paramètrées par les valeurs des coefficients du réseau (ou poids synaptique).

L'objectif de la phase d'apprentissage des réseaux de neurones est de trouver, parmi toutes ces fonctions, celle qui s'approche le plus de la régression (fonction génératrice des exemples). Celle-ci est inconnue (sinon il ne serait pas nécessaire d'utiliser une approximation par réseaux de neurones).

On connaît seulement les valeurs observées (valeurs de la régression à laquelle s'ajoute du bruit) pour plusieurs valeurs prises par les entrées (points de l'ensemble d'apprentissage).

En d'autres termes, on cherche la fonction de régression ($E(Y/X)$: espérance mathématique des valeurs observées Y au point X), et comme le nombre de points est fini, on ne peut en trouver qu'une approximation. Pour trouver cette approximation, il faut définir une fonction de coût qui mesure l'écart entre la sortie du modèle (fonction réalisée par le réseau de neurones) et la sortie désirée.

La fonction de coût est une fonction scalaire qui dépend du vecteur des paramètres du modèle, et des individus de l'ensemble de l'apprentissage.

Dans le cas des réseaux de neurones, le vecteur des paramètres est constitué par les poids du réseau.

Plus la valeur de la fonction coût est petite, plus le modèle reproduit fidelement les observations utilisées pour l'apprentissage.

Les différents algorithmes d'apprentissage cherchent donc à trouver le point dans l'espace des paramètres, pour lequel la fonction de coût est minimale.

A partir d'une fonction de coût dépendant des paramètres et des exemples de l'ensemble d'apprentissage, il faut choisir l'algorithme d'optimisation qui permettra d'estimer le vecteur des paramètres pour lequel la fonction de coût choisie est minimale.

3.3.2 Biais et Variance

Il est important de remarquer que $E[Y/X]$ correspond à la valeur moyenne de Y. Cela implique qu'elle ne sera que rarement égale à un $y \in Y$ en particulier.

Ainsi on peut écrire :

$$y = E[Y/X] + \varepsilon \tag{3.1}$$

où ε est la différence entre l'espérance $E[Y/X]$ et la valeur réelle y. Bien sûr $E[\varepsilon/X] = 0$.

Pour trouver $E[Y/X]$, on procède souvent par approximations successives. On dispose alors d'une estimation "f" de $E[Y/X]$, qui est progressivement améliorée par un calcul itératif.

"f" est généralement une fonction paramétrée par un ensemble W de valeurs ; l'effet du calcul itératif est donc de modifier ces paramètres de façon à améliorer la ressemblance entre "f" et "Y". Alternativement, on peut considérer que l'estimation équivaut à choisir une succession de fonctions f_W parmi un ensemble (ou famille de fonctions) possible f.

Il est clair que cette procédure d'approximation itérative nécessite de mesurer l'adéquation d'une

estimation donnée f_W. Cela implique la détermination en quelque sorte de la distance entre f_W et l'espérance $E[Y/X]$. Or cette espérance est justement la fonction que nous cherchons à approximer. Elle ne nous est donc pas directement accessible pour effectuer la comparaison nécessaire.

On considère alors une fonction qui mesure les valeurs de Y et les valeurs prédites par f_W : $C[Y, f_W(y)]$. En général, on choisit une fonction familière, l'erreur quadratique :

$$C[y, f_W(y)] = [Y - f_W(X)]^2 \tag{3.2}$$

En tenant compte du fait que :

$$E[(Y - E(Y/X)/X] = E[\varepsilon/X] = 0 \tag{3.3}$$

on trouve que :

$$\begin{aligned} E[(Y - f_W(X))^2/X] &= E[(Y - E(Y/X))/X]^2 + E[E(Y/X) - f_W(X)/X]^2 \\ &\quad (\text{Biais})^2 \qquad + \qquad (\text{Variance}) \end{aligned} \tag{3.4}$$

Le premier terme décrit la variation naturellement présente dans les données Y. Le second terme, seule variable de l'équation, exprime la distance moyenne entre la fonction choisie f_W et l'estimation voulue$E[Y/X]$.

Chercher à minimiser l'erreur, revient donc à minimiser " le biais " et " la variance ".

En pratique, le biais et la variance sont antagonistes (diminution de l'un = augmentation de l'autre), il faut donc trouver un compromis, c'est le dilemme : "biais - variance"

Ce dilemme est bien présent en connexionisme, et notamment dans les travaux avec le PMC, où il se traduit par le choix de deux valeurs : la taille de la couche cachée et la taille de l'échantillon d'apprentissage.

En effet, si le PMC est en principe capable d'approximer toute fonction continue, cela n'est vrai que s'il dispose d'un nombre suffisant de neurones cachés. Ainsi, un nombre insuffisant de neurones cachés limite le type de comportement que peut adopter le réseau et introduit de ce fait un biais dans son apprentissage.

En revanche, si le réseau est grand, il y a peu de chance que se produisent des performances adéquates en généralisation.

3.3.3 Convergence uniforme des PMC

En pratique, on dispose que rarement d'une connaissance complète de X et de Y. A défaut, l'estimation doit se faire à partir d'un échantillon, c'est-à-dire d'un ensemble de valeurs $\widetilde{X}$ et $\widetilde{Y}$

Si on suppose que les valeurs de X seront indépendantes, la loi des grands nombre nous assure que :

$$E(\widetilde{X}) \underset{N\to\infty}{\to} E(X) \tag{3.5}$$

où N représente le nombre de données dans l'échantillon, cela implique que :

$$\widetilde{E}\,[Y/X] \underset{N\to\infty}{\to} E[Y/X] \tag{3.6}$$

Intuitivement, puisque f_W a été choisie pour estimer au mieux $\widetilde{E}\,[Y/X]$, et que cette dernière est proche de $E[Y/X]$, c'est-à-dire que la qualité de l'estimation devrait s'améliorer quand la taille de l'échantillon croît.

La convergence de l'estimation du PMC a été démontrée, à condition d'augmenter le nombre de neurones cachés en même temps que la taille de l'échantillon augmente [43].

3.3.4 Généralisation et dimension de *Vapnik* et *Cervonenkis*

Etant assuré de la convergence uniforme des PMC, il devient ensuite important de pouvoir déterminer le nombre d'échantillons nécessaires à un réseau donné pour garantir une généralisation adéquate.

Il est important d'étudier la puissance d'un système d'apprentissage, nous avons vu qu'il existe un compromis à trouver entre le biais du système et la variété des fonctions qu'il peut approximer.

Ce compromis est au coeur du problème de généralisation.

Quand une famille de fonctions est simple, le choix d'une fonction particulière f_W peut être fait à partir d'un petit nombre de données ; inutile de disposer de plus d'observations, car elles n'affecteront pas les résultats de l'estimation.

En revanche, plus le système est puissant, plus grand sera le nombre d'observations nécessaires pour effectuer ce choix, et donc plus il est difficile de s'assurer de la validité de son apprentissage.

Par exemple deux points suffisent pour choisir une famille de droites, trois points sont nécessaires pour une famille de paraboles, et quatre pour une famille de fonctions cubiques.

Il existe donc un lien entre la puissance d'un système, la taille N du corpus d'apprentissage et

la confiance d que l'on peut avoir en ses capacités de génération. Mais quelle est la relation exacte entre ces paramètres ?

En 1971, *Vapnik* et *Cervonenkis* [50] proposent une borne pour d à partir d'une mesure de la puissance du système d'apprentissage.

Soit un sous-ensemble $S \subset R^E$, de cardinalité N : Il y a 2^N façon de partitionner cet ensemble en 2 classes.

Supposons que la famille "f" contienne un choix de fonctions, chacune permettant de répartir les éléments de S en partitions.

Soit maintenant $\phi_f(S)$ le nombre de différents partitionnements possibles de S induits par les éléments de f.

Alors :

$$\phi_f(S) \leq 2^N \tag{3.7}$$

Dans le cas particulier où $\phi_f(S) = 2^N$, on dira que f sépare (" shatters ") l'ensemble S.

Définissons maintenant $\phi_f(N)$ comme le plus grand nombre de partitions que peut effectuer pour tous les sous-ensembles S de cardinalités N possibles :

$$\phi_f(N) = \max_{S \subset R^E} \phi_f(S) \tag{3.8}$$

Ainsi, $\phi_f(N)$ peut-être considéré comme une mesure de la puissance de f, et on a :

$$P[Err(f_W) \succ \varepsilon] \leq 4\phi_f(2N)e^{-e^2\frac{N}{8}} \tag{3.9}$$

où $Err(f_W)$ mesure la différence entre la performance en apprentissage et en généralisation de f_W.

Cette nouvelle relation est importante, car elle permet d'estimer, pour une taille de corpus N donnée, la confiance que l'on peut avoir dans les capacités de généralisation de notre système f.

De plus, en étudiant son comportement quand N tend vers l'infini, elle donne les conditions pour lesquelles un système d'apprentissage est apprenable.

En effet, en 1971, *Vapnik* et *Cervonenkis* [50] ont démontré :

Soit $\phi_f(N) = 2^N$, soit $\phi_f(N) \leq N^{VC} + 1$, où VC est une constante, nommée dimension de *Vapnik-Cervonenkis*.

Si l'on examine l'inégalité 3.9, il est clair que le terme de droite de l'inégalité n'est fini à la limite $N \to \infty$ que quand $\phi_f(N) \leq N^{VC} + 1$

Ce implique qu'un système doit posséder une dimension VC finie pour que ses performances en

généralisation puissent être garanties, c'est à dire pour qu'il soit apprenable.

Ces résultats sont importants pour les recherches en réseaux de neurones, car ils donnent un fondement théorique aux techniques d'apprentissage. En particulier, il a été démontré que les réseaux à couches, quand ils sont mus par des fonctions d'activation à seuil, sigmoïdes exceptionnelles ou tangentielles, ont une dimension VC finie, et donc qu'ils sont apprenables.

L'inégalité de *Vapnik* et *Cervonenkis* [50] a aussi une utilité pour les praticiens du connexionisme. En effet, nous avons vu que la puissance de calcul d'un réseau à couche est essentiellement déterminée par le nombre et la taille de ses couches cachées. Cela implique que pour un corpus d'apprentissage de taille fixe, l'erreur de généralisation doit augmenter avec le nombre de neurones cachés : le réseau est progressivement moins contraint par le corpus d'apprentissage. De ce fait, il est connu en pratique qu'à une performance d'apprentissage égale, les petits réseaux on tendance à mieux généraliser que les grands.

Ainsi, les travaux sur la dimension VC ont une importante primordiale pour le connexionisme. Cependant, ces recherches sont loin d'être complètes. En particulier :

1. Malgré les travaux relatés ici, il n'existe à ce jour aucun moyen sûr pour déterminer la dimension VC d'un réseau de neurones.

2. La taille du corpus prescrite par l'inégalité de Vapnik et de Cervonenkis semble en fait inutilement grande. En effet, plusieurs auteurs obtiennent des résultats empiriques qui suggèrent que les bornes de l'expression ne sont pas précises. Avec pour conséquence que la relation précise entre généralisation et taille du corpus d'apprentissage reste encore à trouver.

Chapitre 4

Complexité des algorithmes d'apprentissage

4.1 Repères historiques et bibliographiques

Le théorème de *Kolmogorov* a été découvert dans les années 80 par *Hecht Nielsen* (1987) [23] et appliqué aux fonctions des réseaux de neurones.

Depuis, plusieurs types de fonctions de réseaux de neurones ont été étudiées, et ainsi plusieurs résultats sur les propriétés d'approximation des réseaux de neurones sont déjà disponibles.

En 1969, *Minsky* et *Papert* [11] ont fait des estimations de la complexité concernant la vitesse de convergence des algorithmes d'apprentissage. Cependant, en ce temps, il n'était pas possible de formuler des résultats plus généraux car les différentes classes de complexité n'ont été définies que durant les années 70 dans les travaux de *Cook* (1971) [51] et d'autres.

En 1972, *Karp* [52] a montré qu'une large classe de problèmes est NP-complet. Dans ses travaux, *Judd* (1980) [53] a présenté les plus importants théorèmes sur la complexité des algorithmes d'apprentissage des réseaux de neurones. Ses résultats ont été étendus ces dernières années.

Il a été montré que quand le nombre de degrés de liberté et les possibilités combinatoires des éléments d'évaluation vont au delà d'un certain seuil, le problème d'apprentissage devient NP-complet.

Les théorèmes d'existence dans ce chapitre ainsi que les estimations de la complexité qui ont été faits pour le problème d'apprentissage peuvent apparaître superflus aux praticiens.

Cependant, dans la filière où les méthodes numériques sont utilisées intensivement, on doit être conscient des limites de l'efficacité des calculs. C'est seulement sur cette base que l'on peut construire

des architectures de réseaux qui rendront plus rapides les solutions des problèmes d'apprentissage.

4.2 Introduction

Avant d'introduire la notion de compléxité aux algorithmes d'apprentissage des réseaux de neurones on utilisera les résultats suivants:

Proposition 4.1 : *Une fonction continue réelle* $f : [0,1] \longrightarrow [0,1]$ *peut être approximée en utilisant un réseau avec des éléments seuil de telle façon que l'erreur approximative totale soit plus petite que tout réel* $\varepsilon > 0$:

$$E = \int_0^1 \left| f(x) - \widetilde{f}(x) \right| dx < \varepsilon \tag{4.1}$$

où $\widetilde{f}$ *dénote la fonction du réseau.*

Corollaire 4.1 : *La proposition précédente est valide aussi pour les fonctions* $f : [0,1] \to (0,1)$ *avec une activation sigmoïdale.*

Le théorème de *Kolmogorov* et la proposition 4.1 (vu précédement) décrivent les propriétés d'approximation des réseaux utilisés pour la représentation des fonctions.

Dans le cas des réseaux de neurones, l'important n'est pas seulement l'architecture du réseau, mais aussi la définition de l'algorithme d'apprentissage.

Le problème d'apprentissage général pour un réseau de neurones consiste à trouver les éléments inconnus d'une architecture donnée.

Les composantes du réseau peuvent être partiellement ou totalement inconnues. Par exemple, la fonction d'activation des unités ou les poids.

Cependant, le problème de l'apprentissage est habituellement contraint de telle manière que les fonctions d'activation soient choisies à partir d'un ensemble fini de fonctions, où certains poids des réseaux sont fixés à l'avance.

La taille du problème de l'apprentissage dépend du nombre de variables inconnues qui sont déterminées par l'algorithme d'apprentissage.

Le temps d'apprentissage requis dans les applications des réseaux de neurones laisse à penser que la résolution des problèmes d'apprentissage est une tâche exigeante.

Il serait utile si le temps de l'algorithme d'apprentissage soit borné par une fonction polynomiale du nombre de variables, mais ce n'est pas le cas.

Il n'y a pas d'algorithmes connus qui résolvent le probléme d'apprentissage en un temps polynômial dépendant du nombre de variables inconnues. De plus, il est peu probable qu'un tel algorithme existe. Dans la terminologie de la théorie de la compléxité on dit que: "*le probléme général d'apprentissage des réseaux de neurones est NP-Complet*"

4.3 Les classes de complexité

La complexité des algorithmes est le temps d'apprentissage nécessaire pour que le réseau soit à même de généraliser à partir des données.

Définitions:

1. Algorithme : Un algorithme **A** est une séquence d'instruction qui transforme une chaine de carctéres (les données) en une autre chaine (résultat)

2. **Probléme (P)** : Un algorithme **A** pour résoudre **(P)** est un algorithme qui résoud tout éxemple **I** de **(P)** on dit instance

3.Taille des données de l'instance I $\mathbf{T}_I$ (encombrement mémoire): nombres de cases mémoire (bit) nécéssaires pour stocker toutes les données de I

Temps d'éxécution de A sur I

f(T_I) : nombre d'opérations éléùentaires (addition, soustraction, multiplication, division, affectation, comparaison ....) effectuées pour résoudre I.

On cherche une borne de $f(T_I)$: si $f(T_I) = O(T_I^k)$ on dit que A est polynomial (efficace) et que (P) est facile

si A est non déterministe (existence d'une instruction non déterministe) et si $f(T_I)$ est polynomial on dit que A est non déterministe polynomial.

Définition 4.1 : *on dit qu'un probléme est dans P si il existe un algorithme polynomial pour résoudre*

Cependant, il y a des problèmes pour lesquels il est encore inconnu si oui ou non ils appartiennent à la classe P.

Il n'a pas été possible de trouver un algorithme de temps polynômial pour résoudre aucun de leurs exemples et il n'a pas également été possible de montrer qu'un tel algorithme n'existe pas. (ce qui les mettrait en fait hors de la classe P).

Définition 4.2 :

On dit qu'un problème est dans NP s'il existe un algorithme non déterministe polynomial pour résoudre.($P \subseteq NP$)

π se réduit polynomialement à π' s'il existe un algorithme polynomial pour π et qui fait appel à π' (mais l'appel à π' est considéré comme une opération élémentaire)

On peut expliquer la différence entre les deux possibilités en étudiant le problème du voyageur de commerce.

Problème du voyageur de commerce.

Ce voyageur doit visiter n villes pour vendre sa marchandise. Il existe entre ces villes un certain nombre de routes.

Existe-t-il un trajet permettant de visiter toutes les villes en moins de b kms? C'est un problème difficile. On ne sait pas s'il existe un algorithme polynômial pour le résoudre. En revanche, si l'on donne un trajet, il est quasi-immédiat de vérifier si ce trajet fait moins de b kms. Le problème est dans NP.

Pour résumer, être dans P c'est trouver une solution en temps polynômial, tandis qu'être dans NP, c'est prouver en temps polynômial qu'une proposition de réponse est une solution du problème. Ainsi, tout problème qui est dans P se trouve dans NP.

Le problème ouvert le plus important dans la théorie de la complexité est : Est ce que les classes P et NP sont identiques ou différentes ?

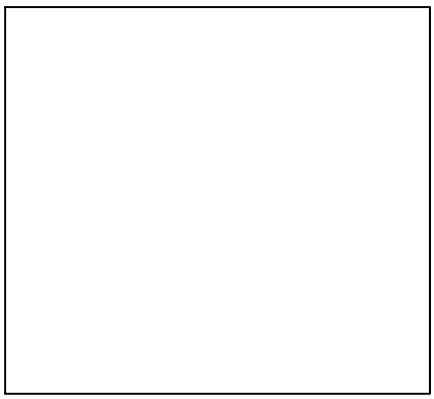
Il est clair que $P \subseteq NP$; Si un problème peut être résolu en un temps polynômial, alors une solution non déterministe peut aussi être vérifiée en un temps polynômial.

Il n'est pas connu s'il y a des problèmes qui appartiennent à la classe NP mais pas à la classe P. Des candidats possibles viennent de la classe NP_C appelée les problèmes $NP-$complets.

Définition 4.3 : *On dit qu'un problème est H NP-complet si* $\forall \pi \in NP$ π se réduit polynomialement à H

La classe NP_C est importante, car si un algorithme avec une complexité polynômiale peut être trouvé et qui peut résoudre tout problème de la classe NP_C, alors tout autre problème dans la classe NP peut aussi être solvable en un temps polynômial.

Cette figure 4-1 montre un diagramme de la classe NP supposant que $NP \neq P$. Les flèches dans le diagramme illustrent le fait que tout problème dans la classe NP peut être transformé en un problème dans la classe NP_C en un temps polynômial.



Figure˜4-1: Représentation schématique des classes NP, P et NP_C.

4.3.1 Les problèmes d'apprentissage *NP* complet

Il a été prouvé que plusieurs problèmes , tels que : le voyageur de commerce ou le problème de satisfaisabilité sont des problèmes NP-Complet.

On peut montrer à l'aide d'un problème appelé satisfaisabilité (SAT) qu'un problème NP complet peut être réduit à un exemple (instance) de problème d'apprentissage pour un réseau de neurones en un temps polynômial.

Définition 4.5 : *(problème SAT)*
Un problème de satisfaisabilité ou SAT est défini par :

1. *Une séquence $X = (x_1, ..., x_n)$ de n variables booléennes (donc prenant leur valeur dans $\{0, 1\}$ ou (vrai,faux)).*

2. *Une formule F définie comme la conjonction de m clauses $C_j = C_1 \wedge ... \wedge C_m$. Chaque clause est définie comme une disjonction des variables x_i ou de leur négation $\overline{x}_i$.*

   On aura ainsi :

   $$C_j = y_1 \vee ... \vee y_k \text{ avec } y_i \in \{x_1, ..., x_n, \overline{x}_1, ...\overline{x}_n\}\,. \tag{4.2}$$

   On dit alors que la clause C_j est d'ordre k.

   Une formule F est satifaisable s'il existe n valeurs pour les variables $\{x_1, ..., x_n\}$ qui rendent la formule F vraie, donc qui rendent toutes les clauses $\{C_1, ..., C_m\}$ vraies. La recherche des ces valeurs particulières correspond à la résolution du problème SAT.

Un exemple de problème SAT

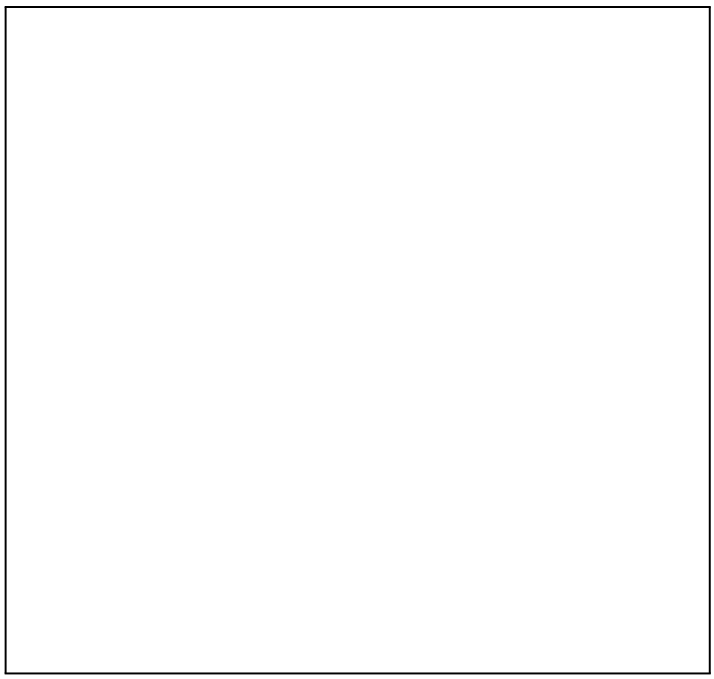
- $X = \{x_1, x_2, x_3\}$: 3 variables booléennes.
- $F = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6$

$C_1 = x_1 \vee \overline{x}_2$; $C_2 = \overline{x}_2 \vee \overline{x}_3$; $C_3 = \overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3$; $C_4 = \overline{x}_1 \vee x_2 \vee x_3$; $C_5 = x_1 \vee x_2 \vee \overline{x}_3$; $C_6 = x_1 \vee \overline{x}_2 \vee x_3$

Un résultat classique de *Cook* [54] (Le théorème de *Cook* établit que le problème de satisfaisabilité (SAT) est NP-complet), montre que le problème de satisfaisabilité est un membre de la classe NP_C.

Ce résultat est également vrai pour des expressions de F avec au plus trois littéraux[1] dans chaque disjonction (ce problème est appelé 3SAT).

On va transformer le problème 3SAT en un problème d'apprentissage d'un réseau de neurones en utilisant le réseau de la figure 4-2.



Figure˜4-2: Réseau équivalent au problème 3SAT

L'activation des unités individuelles sera calculée en utilisant les fonctions seuil. Une expression F (forme normale conjonctive[2]) qui inclue seulement n variables $x_1, ..., x_n$ est donnée.

[1] littéral : lier une variable logique à une valeur (c'est la valeur numérique ou valeur en toutes lettres, par opposition à un nom de variable)

[2] conjonction des disjonctions littéraux

La disjonction en forme normale contient au plus 3 littéraux. On veut trouver les vraies valeurs des variables qui rendent F vraie.

Ce réseau a été construit en réservant pour chaque variable x_i, un poids w_i et une unité de calcul avec un seuil $0,5$.

La sortie de la $i^{ème}$ unité dans la première couche est interprétée comme la valeur vraie affectée à la variable x_i.

Les unités avec un seuil $(-0,5)$ sont utilisées pour nier la vraie valeur de la variable x_i. Les sorties de chacune de ses unités peuvent être interprétées comme $\neg x_i$. La troisième couche (à compter à partir du fond) calcule la disjonction des sorties des unités connectées à eux (les clauses dans la forme normale). Si toute connexion arrivant à une unité à la troisième couche transporte la valeur 1, l'unité déclenche 1. La connexion dans cette figure correspond aux 2 clauses suivantes :

$$x_1 \vee \neg x_2 \vee \neg x_3 \qquad \text{et} \qquad x_2 \vee x_3 \vee \neg x_n$$

La dernière unité calcule la conjonction des disjonctions du réseau.

On suppose que l'expression F contient m clauses. La valeur m est utilisée comme un seuil de la simple unité dans la couche de sortie. l'expression F est vraie seulement si toutes les disjonctions sont vraies.

On a le résultat suivant :

Proposition 4.2 : *Le problème d'apprentissage général pour les réseaux des fonctions à seuil est NP complet.*

Preuve:

L'expression logique F dans la forme conjonctive normale contient n variables et peut être transformée en un temps polynômial en une description d'un réseau du type montré par la figure précédente.

Pour chaque variable x_i, un poids w_i est défini et les connexions aux unités dans la troisième couche sont fixées selon la forme conjonctive normale que nous traîtons.

Ceci peut être fait en un temps polynômial (utilisant un codage approprié) parce qu'il se tient, pour le nombre m des différentes disjonctions possibles dans la formule 3SAT, que $m \leq (2n)^3$.

Après la transformation, le problème de l'apprentissage suivant doit être résolu :

Une entrée $x = 1$ doit produire la sortie $F = 1$.

Seulement les poids du réseau doivent être trouvés :

Si une instantiation[3] A avec des valeurs logiques des variables x_i existe, telle que F devienne vraie, alors il existe des poids w_1, $w_2, ...$, w_n qui résolvent le problème d'apprentissage.

Il est seulement nécessaire de poser $w_i = 1$ si $x_i = 1$. Si $x_i = 0$, on pose $w_i = 0$. Ainsi dans les deux cas $w_i = x_i$. Le contraire est vrai aussi, s'il existe des poids w_1, $w_2, ...$, w_n qui résolvent le problème d'apprentissage, alors l'instantiation $x_i = 1$ si $w_i \geq 0,5$ et $x_i = 0$ sinon, est une instantiation valide qui rend F vraie.

Ceci prouve que la satisfabilité des expressions logiques peut être transformée en un problème d'apprentisage des réseaux de neurones.

Nous devons à présent montrer que le problème d'apprentissage appartient à la classe NP, c'est à dire que la solution peut être vérifiée en un temps polynômial.

Si les poids w_1, w_2,...., w_n sont donnés, alors une seule exécution du réseau peut être utilisée pour vérifier si la sortie F est égal à 1.

Le nombre de pas de calcul est directement proportionnel au nombre n de variables et au nombre m de clauses disjonctives, qui est limité par un polynôme dans n. Le temps nécessaire pour vérifier une instantiation est ainsi lié par un polynôme dans n.

Ceci veut dire que le problème d'apprentissage donné appartient à la classe NP.

Le problème d'apprentissage indiqué ci-dessus paraît être plus difficile que dans le cas typique, parce que plusieurs poids du réseau ont été sélectionnés et fixés à l'avance.

Habituellement, tous les poids et les seuils sont considérés comme des variables. On pourrait penser que si tous les poids peuvent varier librement, ceci ouvre une plus grande région dans l'espace des poids à explorer par l'algorithme d'apprentisasge, et que ceci pourrait réduire la complexité du problème d'apprentissage.

Cependant, il n'en est pas ainsi. La proposition 4.2 est toujours valide même dans le cas où les poids sont libres d'être modifiés.

La même chose est vraie si la fonction seuil est remplacée par la fonction sigmoïdale.

Il a été montré que l'apprentissage d'un réseau avec 3 unités peut être NP-complet pour un certain ensemble d'apprentissage.

[3] Instantiation : produire une version plus définie d'un certain objet en remplaçant des variables avec des valeurs (ou d'autres variables).

4.3.2 Complexité de l'apprentissage avec un réseau "et-ou"

Puisque le problème d'apprentissage général est NP complet, on peut essayer d'analyser des types d'architecture restreinte pour découvrir s'ils peuvent être exécutés dans un temps polynômial.

Comme deuxième exemple, on considère des réseaux qui peuvent seulement calculer les fonctions "ET" ou "OU". Cette restriction limite le nombre de combinaisons disponibles et on peut spéculer que ceci peut aider pour éviter une explosion combinatoire.

Cependant, ce n'est pas le cas, et il peut être montré que le problème d'apprentissage pour que ce type de réseau reste NP complet.

Proposition 4.3 : *Le problème d'apprentissage de l'architecture des réseaux de neurones où les unités peuvent seulement calculer le fonctions "ET" ou "OU" est NP complet.*

4.3.3 Simplification de l'architecture du réseau

La complexité des résultats de la section précédente montre que l'apprentissage des réseaux de neurones est un problème informatique difficile.

Si l'architecture du réseau n'est pas bien définie, et si l'algorithme d'apprentissage doit trouver une configuration valide dans une région large de l'espace des poids, alors on ne peut pas utiliser un algorithme à temps polynômial.

Cette situation rend nécessaire le fait d'améliorer les méthodes d'apprentissage connues pour une taille fixe de l'algorithme.

Une approche plus prometteuse, afin d'éviter l'explosion combinatoire, est de tester différentes sortes d'architectures simples.

Judd [53] a analysé des réseaux de neurones plats (flat) pour essayer de découvrir si, au moins, pour quelques uns de ces réseaux des algorithmes à temps polynômial peuvent être conçus.

La motivation qu'il y a derrière l'analyse théorique des réseaux "plats" est la structure du cortex humain.

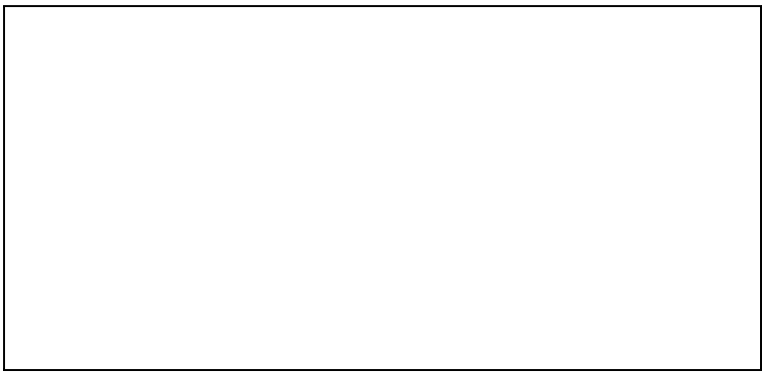
Le cortex ressemble à une structure (peu profonde) à deux dimensions. Ce type d'architecture peut apprendre plus facilement que des réseaux complètement connectés.

Cependant, *Judd* a pu montrer que l'apprentissage dans les réseaux plats à deux dimensions est aussi NP complet.

Un autre type de réseaux à une dimension peut apprendre en un temps linéaire. Et si seulement

le cas moyen[4] est utilisé, le temps d'apprentissage peut être réduit à une constante.

La figure 4-3 montre un réseau à une dimension.



Figure˜4-3: Un réseau de trois couches avec une seule direction

Les entrées viennent du haut, et la sortie est produite par la troisième couche. Les interconnexions sont définies de telle manière que chaque unité propage un signal seulement au voisin de gauche et de droite.

Le $i^{ème}$ bit de l'entrée peut affecter au plus trois bits de la sortie. La sortie de l'unité "C" dépend seulement de l'information des unités "A", "B", "D", "E" et "G". Ces unités constituent, ensemble avec "C", une colonne d'information.

Les unités "D", "E", "F", "G", "H" et "J" constituent les colonnes avoisinantes.

Les deux structures se chevauchent et il est nécessaire de trouver une combinaison correcte pour un algorithme d'apprentisage donné.

Supposons que les entrées et les sorties sont binaires et que les unités du réseau sont des éléments à seuil à deux entrées (qui peuvent calculer 14 différentes fonctions logiques).

Chaque colonne d'information faite de six unités peut assumer n'importe laquelle des 14 configurations. Toutes ne sont pas compatibles avec la configuration de la colonne avoisinante.

Le problème combinatoire peut être résolu en deux étapes : Premièrement, les configurations valides des colonnes sont déterminées en utilisant les "vecteurs" données d'entrée et de sortie. Deuxièmement, seulement les configurations compatibles avec les configurations des "voisins" sont acceptées.

Dans un réseau à une dimension avec "n" bits de sortie, on doit combiner les sorties de n colonnes différentes.

[4] cas moyen : les paires d'entrée-sortie sont sélectionnées aléatoirement.

Judd [53] a montré que ce calcul peut être fait en un temps linéaire.

Quelques expériences d'ordinateur montrent que dans le "cas moyen", c'est à dire quand les paires d'entrée-sortie sont selectionnées aléatoirement, le temps nécessaire pour la coordination des colonnes est limité par une constante.

Le réseau peut apprendre dans un temps constant indépendament de sa largeur.

Bien que le cas unidimensionnel ne peut par être considéré comme trés réaliste, il nous donne un conseil sur une stratégie possible pour réduire la complexité du problème d'apprentissage : si le réseau peut être modularisé et l'échange de l'information entre les modules peut être limité, il y a une chance qu'un algorithme efficace d'apprentissage puisse être trouvé pour de tels réseaux.

4.3.4 L'apprentissage avec conseils

L'exemple des réseaux à une dimension illustre une méthode capable d'arrêter l'explosion combinatoire générée par les problèmes d'apprentissage.

Une autre technique consiste en la considération de quelques propriétés de la fonction à modeler avant la sélection de l'architecture du réseau considéré. Le nombre de degrés de liberté du problème d'apprentissage peut être réduit en exploitant des "conseils" sur la forme ou les propriétés de la fonction qui va être approximée.

Supposons que l'on veuille approximer une fonction f à "n" arguments en utilisant un réseau à trois couches. Le réseau a "n" entrées et une seule unité de sortie.

La fonction primitive dans les unités est la fonction symétrique sigmoïde.

On peut forcer le réseau à calculer exclusivement des fonctions paires pour lesquelles on a :

$$\varphi(x_1, x_2, ..., x_n) = \varphi(-x_1, -x_2, ..., -x_n) \tag{4.3}$$

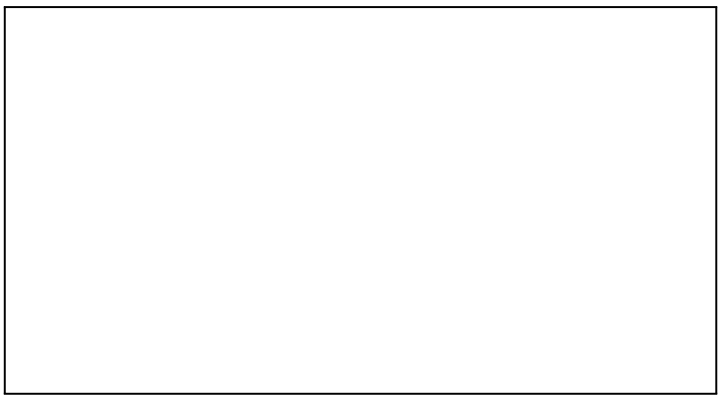
Ceci peut être garanti en répliquant chaque unité et chaque poids dans la seconde couche.

La figure 4-4, montre comment interconnecter les unités dans le réseau.

Les deux entrées "x_i" ou "$-x_i$" mènent vers la même sortie du réseau. Ainsi tout le réseau utilise la même approche (pour chaque variable). Il sera seulement capable de calculer les fonctions paires. Ce qui produit une réduction de la région de recherche de l'espace des fonctions.

Ceci est une méthode générale pour inclure des "conseils" ou des conditions dans un réseau.

Si la rétropropagation est utilisée comme méthode d'apprentissage, on doit aussi veiller à garder l'identité de la duplication de tous les poids dans le réseau. Toutes les corrections des poids avec le même nom sont calculées indépendament, mais sont rajoutées avant de mettre à jour le poids.



Figure˜4-4: Réseau pour les fonctions paires

Dans le cas où les poids ont le même nom mais un signe différent, les corrections du poids sont multipliées par le signe correspondant avant qu'il soit rajouté.

De cette façon, on garde la structure originale du réseau et les poids qui devraient être identiques demeurent ainsi.

D'autres types d'invariance sont plus difficiles à implémenter. Dupliquer le réseau et comparer les valeurs de sorties respectives aide à faire cela.

Supposons que le réseau doit produire les mêmes résultats réels quand deux différentes valeurs de dimension n, x_1 et x_2, sont présentées. La différence entre les deux vecteurs peut reflèter une sorte d'invariance que nous voulons imposer. Le vecteur x_2 peut être égal à $-x_1$ où les composantes de x_2 peuvent être les permutations des composantes de x_1.

On veut calculer une fonction invariante sous de telles modifications des entrées. La figure 4-5 ci-dessous montre la stratégie générale utilisée pour résoudre le problème.

Un simple réseau produit la sortie y_1 pour l'entrée x_1 et la sortie y_2 pour l'entrée x_2.

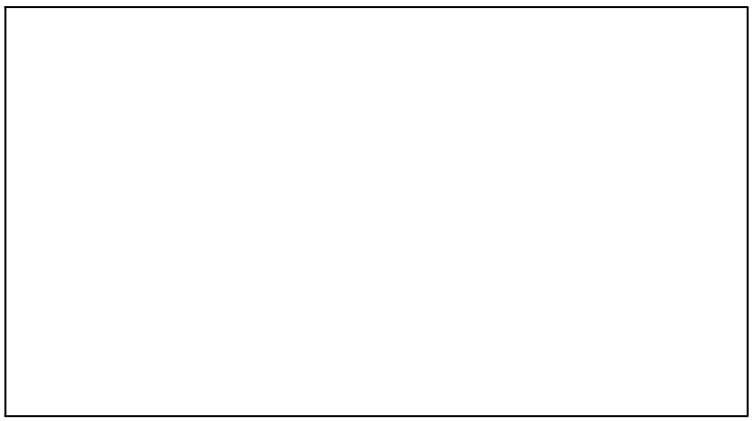
Le réseau peut être forcé de rapprocher y_1 et y_2 en minimisant la fonction $(y_1 - y_2)^2$. Ceci est fait en duplicant le réseau et en exécutant le calcul $\frac{(y_1-y_2)^2}{2}$.

Si l'on veut que le réseau produise la valeur cible t, la différence entre y_2 et t doit aussi être minimisée.

Le côté droit de l'unité de calcul additionnelle, montré dans la figure précédente, montre la différence qui doit être minimisée en considérant que le côté gauche contient la dérivée par rapport à y_2.

Le réseau prolongé de la figure 4-5 peut apprendre avec la méthode de la rétropropagation.

En utilisant la descente du gradient, on essaye de forcer le réseau à produire le résultat correct



Figure˜4-5: Réseau prolongé pour traiter les invariants.

et de respecter l'invariance désirée.

Une approche alternative est de spécialiser les premiéres couches du réseau à la production de l'invariance désirée et les derniéres au calcul de la sortie désirée.

Les conseils, c.a.d, la connaissance du sujet des invariances nécessaires ou désirées de la fonction de réseau, peuvent également être employés pour prétraiter l'ensemble d'apprentissage. Si la fonction du réseau doit (de nouveau) être paire, on peut à nouveau augmenter l'ensemble d'apprentissage en introduisant des paires additionnelles d'entrée-sortie.

Les nouvelles données d'entrées sont obtenues à partir des anciennes juste en changeant le signe, tandis que la valeur des sorties de la cible demeurent constantes.

Un plus grand ensemble d'apprentissage réduit la région faisable de l'éspace de poids dans laquelle une solution au problème d'étude peut être trouvée.

Ceci peut mener à une meilleure approximation de la fonction inconnue si la VC dimension de l'espace recherché est finie.

La technique de l'apprentissage avec conseils essaye de réduire la complexité inhérente du problème d'apprentissage en réduisant le degré de liberté du réseau de neurones.

CONCLUSION

Le travail dont nous avons rendu compte dans le présent mémoire récapitule des résultats fondamentaux, sous forme de théorèmes, sur les possibilités d'approximation des fonctions par le Perceptron MultiCouches (PMC). Le perceptron multicouches étant un des plus populaires et pratiques nombreux modèles de réseaux de neurones.

On a ainsi montré les fondements mathématiques des réseaux de neurones et le succès du modèle statistique.

Ce mémoire est une synthèse d'articles, où nous considérons les implications du théorème de *Kolmogorov* sur l'approximation de fonctions, qui motive l'utilisation des réseaux de neurones feedforward comme approximateurs de fonctions continues

On retrace ainsi l'historique du problème d'approximation de fonctions depuis le théorème de *Kolmogorov,* qui fut une réponse à la $13^{ème}$ conjecture d'*Hilbert* , jusqu'à ce que des preuves mathématiques rigoureuses soient établies par *Cybenko* (1989), *Hornik* et *Al.* (1989) et *Funahashi* (1989) sur l'universalité de l'approximation des réseaux de neurones feed-forward en employant aussi bien des fonctions continues de type sigmoïde que d'autres plus générales.

Le fait saillant est qu'un réseau feed-forward à deux couches avec un nombre suffisant d'unités cachées, une fonction d'activation sigmoïde, et une sortie linéaire est capable d'approximer toute fonction continue $f : R^n \longrightarrow R$.

Nous avons ensuite abordé la notion de complexité des algorithmes d'apprentissage.

Après que *Hecht Nielsen* ait remarqué que le théorème de *Kolmogorov* pouvait être interprété en termes de réseaux de neurones, plusieurs types de fonctions de réseaux neuronaux ont été étudiées, et ainsi plusieurs résultats sur leurs propriétés d'approximation sont disponibles.

En 1969 *Minsky* et *Papert* ont fait des estimations de la complexité concernant la vitesse de convergence des algorithmes d'apprentissage. Mais en ce temps il n'était pas possible de formuler des résultats plus généraux car les différentes classes de complexité n'ont été définies que durant les années 70 dans les travaux de *Cook* (1971).

En 1972, *Karp* a montré qu'une large classe de problèmes est NP-Complet. *Judd* (1980) a présenté les plus importants théorèmes sur la complexité des algorithmes d'apprentissage des réseaux de neurones.

Il a été démontré que quand le nombre de degrés de liberté et les possibilités combinatoires des éléments d'évaluation vont au delà d'un certain seuil, le problème d'apprentissage devient NP-Complet.

Toutefois, à ce jour, il y a des problèmes de recherche reliés à la taille d'échantillonnage, étant donné que le processus d'élaboration d'un réseau de neurones commence toujours par le choix des échantillons. Cette étape est cruciale et aide le concepteur à déterminer le type de réseau le plus approprié pour résoudre son probléme .

La façon dont se présente l'échantillon conditionne le type de réseau, le nombre de cellules d'entrées, le nombre de cellules de sortie et la façon dont il faudra mener l'apprentissage, les tests et la validation.

Bibliographie

[1]

[2] A. N. Kolmogorov : On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSR*, 114, 953-956, 1957.

[3] Hilbert, *Bulletin of the American Mathematical Society* 8, 437-479, 1902.

[4] D. A. Sprecher : On the structure of continous functions of several variables. *Transactions American Mathematical Society*, vol. 115, N3, p 340-355, 1965.

[5] K. Levenberg : A method for the solution for certain non-linear problems in least squares. *Quarterely journal of applied mathematics*, vol. 2, p 164-168,1944.

[6] D. W. Marquardt : An algorithm for least-squares estimation. *Journal of Soc. Indust. Appl. Math.*, Vol. 11, n^o 2, p 431-441. June 1963.

[7] Ciarlet, Analyse numérique matricielle et Optimisation, Masson, Paris 1985.

[8] W. S. Mc Culloch et W. Pitts : A logical calculus of the idea immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, p 115-133al, 1943.

[9] D. O. Hebb : The organization of behaviour. Wiley, New-York, 1949.

[10] R. Rosenblatt, Principles of Neurodynamics. *Spartan Books*. New-York, 1958.

[11] M. Minsky et S. Papert : Perceptrons. Cambridge : MIT Press. (1969)

[12] J. J. Hopfield : Neural and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences*, USA, p 2-554, 1982.

[13] P. J. Werbos : Generalization of back-propagation with application to gas market model. *Neural network*, 1, p 339-356.

[14] D. E. Rumelhart et al. : Learning representations by back-propagating errors. *Nature*, vol 323, 1986.

[15] G. Dreyfus, J.M. Martinez, M. Samuelides, M. B. Gorbon, F. Badran, S. Thiria, L.Herault : *Réseaux de neurones : Méthodologie et applications*. Editions Eyrolles 2004.

[16] D. S. Broomhead et D. Lowe : Multivariate functional interpolation and adaptative networks. *Complex systems*, 2, p 321-355, 1988.

[17] B. Wierenga et J. C. W. H. Kluytmans : *Neural networksen in marketing*. p 9-25, 1994.

[18] K. Hornik : Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4 (2), p 251-257. 1991

[19] V. Venugopal et W. Baets : Neural networks and statistical techniques in merketing research. *Marketing intelligence and planning*, vol. 12, n^o 7, p 30-38. 1994.

[20] R.N. Shepard : Internal representation of universal regularities : A challenge for connexionism (In L. Nadel et al. (Eds)). *Neural computations and mental computations*. MIT Press. 1990.

[21] *"Mémoire sur les équations algébriques"* et *"Démonstration de l'impossibilité de la résolution algébrique qui passent le 4ème degré"* Niels Abel dans ses oeuvres complètes : Jacques Gabay en 1824 et 1826.

[22] V. I. Arnold : On the representation of functions of several variables in the form of superposition of functions of smaller number of variables. *Mathematicheskoe prosveshchenie*, Vyp. 3, p 41-61. 1958.

[23] R. Hecht-Nielsen : Kolmogorov's mapping neural network existence theorem. *IEEE International Conference on Neural Networks*. San Diego. SOS printing, Vol. 2, p 11-14. 1987.

[24] K. Hornik, M. Stinchcombe et H. White : Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, vol. 2, p 359-366,1989.

[25] F. Girosi et T. Poggio : Representation Properties of Networks : Kolmogorov's Theorem is Irrelevant. *Neural Computation* 1, p 465-469, 1989.

[26] A. G. Vitushkin : On Hilbert's thirteenth problem. *Dokl. Alkad. Nauk SSSR* 95, 701-704. 1954.

[27] A. G. Vitushkin : On representation of functions by means of superpositions and related topics. *L'enseignement mathématique*. 1977.

[28] G. M. Henkin : Linear superpositions of continuously differentiable functions. *Dokl. Akad. Nauk. SSSR* 157, 288-290. 1964.

[29] V. Kurkova : Kolmogorov's Theorem Is Relevant. *Neural Computation* 3, p 617-622, 1991.

[30] K. Funahashi : On the approximate realization of continuous mappings by neural networks. *Neural Networks* vol. 2, p 183-192, 1989.

[31] R. Hecht-Nielsen : Neurocomputing. Addison-Wesley. *Reading IMA*. 1989.

[32] G. Cybenko : Approximation by superpositions of a sigmoïdal function. *Math. Control Systems Signals*, 1989.

[33] S. Caroll and B. W. Dickinson : Construction of neural nets using the Radon transform. *In proceedings of the International Joint Conference on Neural Networks*, pp. I, 607-611. IEEE, New York. 1989.

[34] M. Stinchombe and H. White : Multilayer feedforward neural networks are universal approximations. *Neural networks*, 2(3), 359-366. 1989.

[35] M. Stinchombe and H. White : Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks* 3(4), 551-560. 1990.

[36] D. G. Luenberger : *Optimization by vector space method*. John Wiley, New York. 1969.

[37] B. Irie and S. Miyake : Capabilities of three-layered perceptrons. *Proc. Intl. Conf. on neural networks*. IEEE CS Press, CA Vol. 1, 641-648. 1988.

[38] K. Hornik : Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4 (2), p 251-257. 1991

[39] G. Z. Sun and E. W. Cheney : The fundamentals of sets of ridge functions. *Aequationes Math.*, 44, 226-235. 1992.

[40] W. A. Light : *Ridge functions, sigmoidal functions and neural networks in approximation theory V II.* E. Cheney, C. K. Chui and L. L. Shumaker Editions, 163-206. Academic Press, Boston. 1992.

[41] Y. Ito : Representation of functions by superpositions of step functions and their applications to neural network theory. *Neural networks*, 4 (3), 385-394. 1991.

[42] A. R. Barron : Universal approximation bounds for superpositions of a sigmoid function. *IEEE Transactions on Information Theory*, 39, 930-945. 1993.

[43] H. White : Learning in artificial networks : A statistical perspective. *Neural computation* 1, 425-464. 1989

[44] K. Hornilk et al. : Some new results on neural network approximation. *Neural network*, 6(8), 1069-1072. 1993.

[45] J. Park and I. W. Sandberg : Universal approximation using radial basis function networks. *Neural computation* 3 (2), 246-257. 1991.

[46] E. Hartmann, J. Keeler and J. Kowalski : Layered neural networks with gaussian hidden units as universal approximations. *Neural computation* 2, 210-215. 1990

[47] V. Cherkassky, V. Friedman and H. Wechleser : *From statistics to neural networks : Theory and pattern recognition applications.* Berlin : Springer-Verlag. 1994.

[48] V. N. Vapnik : *The nature of statistical learning theory.* New York : Springer. 1995.

[49] T. Hastie : *Discriminant adaptive nearest neighbor classification.* 1994.

[50] V. N. Vapnik and A. Y. Cervonenkis : On the uniform convergence of relative frequencies of events to their probabilities. *Theory of probability and its applications*, 16(2), 264-280. 1971.

[51] S. A. Cook : The complexity of theorem proving procedures. *In 3^{rd} ACM Symposium on theory of computing.* 151-158, Ohio. 1971.

[52] R. M. Karp : reducibility among combinatorial problems. *In complexity of computer computations (Proc. Symposium center).* Yoktown Height, NV. 1972.

[53] S. Judd : On the complexity of loading shallow neural networks. *Journal of complexity*, V 4 , n^o 3, p 177-192. Sept. 1988.

[54] S. A. Cook : Short propositional formulas represent non-deterministic computations. *Information processing letters,* 26 : 269-270. 1988.

[55] R. Rojas : *Neural networks : A systematic introduction.*

Annexe A

Les réseaux de neurones feed-forward sont des approximateurs universels [38]

Définition 2.1 : *Pour tout* $r \in N = \{1, 2, ...\}$

A^r *est l'ensemble des fonctions affines de* $R^r \to R$*, c'est l'ensemble de toutes les fonctions de la forme* $A(x) = wx + b$*, où "*w*" et "*x*" sont des vecteurs dans* R^r

"." dénote le point usuel produit de vecteurs; "b*"* $\in R$ *est un scalaire.*

Dans le contexte actuel, x correspond à l'entrée du réseau ; w correspond aux poids du réseau de l'entrée à la couche intermédiaire ; b correspond au biais.

Définition 2.2 : *Pour toute fonction mesurable (de* Borel*)* $G(.)$ *de* $R^r \to R$ *et* $r \in N$*, soit* $\sum^r(G)$ *la classe des fonctions :*

$$\left\{ f : R^r \to R : f(x) = \sum_{j=1}^{q} \beta_j G(A_j(x)),\ x \in R^r,\ \beta_j \in R,\ A_j \in A^r,\ q = 1, 2, ... \right\} \tag{A.1}$$

Un cas principal se produit quand G est une fonction sigmoïde (squashing). Dans ce cas, $\sum^r(G)$ est la classe familière des "fonctions sorties" pour un réseau feedforward à une seule couche cachée avec une fonction sigmoïde (squashing) à la couche cachée et une fonction sigmoïde (squashing) à la couche de sortie.

Le scalaire β_j correspond aux poids des réseaux des couches cachées vers celles de sortie.

Définition 2.3 : *Une fonction* $\Psi : R \to [0,1]$ *est une fonction sigmoïde (squashing) si elle ne décroit pas.*

$$\lim_{\lambda \to +\infty} \Psi(\lambda) = 1 \text{ et } \lim_{\lambda \to -\infty} \Psi(\lambda) = 0 \tag{A.2}$$

Parce que les fonctions sigmoïdes (squashing) ont plusieurs discontinuités, elles sont mesurables.

Des exemples usuels de fonctions sigmoïdes (squashing) sont :

Les fonctions "seuil" :

$$\Psi(\lambda) = 1_{\{\lambda \geq 0\}} \tag{A.3}$$

(où $1_{\{.\}}$ dénote la fonction indicatrice).

La fonction "ramp" (inclinée) :

$$\Psi(\lambda) = \lambda 1_{\{0 \leq \lambda \leq 1\}} + 1_{\{\lambda > 1\}} \tag{A.4}$$

Squasher cosinus de Gallant *et* White *(1988)* :

$$\Psi(\lambda) = (1 + \cos\left[\lambda + \frac{3\pi}{2}\right])(\frac{1}{2}) 1_{\{-\frac{\pi}{2} \leq \lambda \leq \frac{\pi}{2}\}} + 1_{\{\lambda > \frac{\pi}{2}\}} \tag{A.5}$$

On définit une classe de $\sum \Pi$ fonctions de sortie du réseau :

Définition 2.4 : *Pour toute fonction mesurable* $G(.)$ *de* $R^r \to R$ *et* $r \in N$ *soit* $\sum \Pi^r(G)$ *la classe des fonctions*

$$\left\{ f : R^r \to R : f(x) = \sum_{j=1}^{q} \beta_j \prod_{k=1}^{L_j} G(A_{jk}(x)),\ x \in R^r,\ \beta_j \in R,\ A_{jk} \in A^r,\ l_j \in N,\ q = 1, 2, ... \right\} \tag{A.6}$$

Le résultat général est d'abord prouvé pour $\sum \Pi$ réseaux et par la suite étendu à $\sum$ réseaux. Ces derniers sont un cas spécial des $\sum \Pi$ réseaux pour lesquels $l_j = 1$ pour tout j.

Les notations pour les classes de fonctions à approximer est donnée par la définition suivante.

Définition 2.5 : *Soit* C^r *l'ensemble des fonctions continues de* $R^r \to R$*, et soit* M^r *l'ensemble des fonctions mesurables de Borel de* $R^r \to R$*.*

Notons le σ *domaine de Borel de* R^r *par* β^r*.*

Les classes $\sum^r(G)$ *et* $\sum \Pi(G)$ *appartiennent à* M^r *pour toute fonction mesurable de Borel* G*.*

Quand G *est continue,* $\sum^r(G)$ *et* $\sum \Pi(G)$ *appartiennent à* C^r*. La classe* C^r *est un sous-enesemble de* M^r*.*

Les premiers résultats concernent l'approximation des fonctions dans C^r, et seront étendus ensuite à l'approximation des fonctions dans M^r.

Le rapprochement des fonctions f et g appartenant à C^r ou à M^r est mesuré par une métrique ρ.

Le rapprochement d'une classe de fonctions à une autre est expliqué par le concept de densité.

Définition 2.6 : *Un sous-espace "S" d'un espace métrique (X, ρ) est ρ dense dans le sous-ensemble T, si pour chaque $\varepsilon > 0$ et pour chaque $t \in T$, il existe $s \in S$ tel que :*

$$\rho(s, t) < \varepsilon \tag{A.7}$$

En d'autres termes, un élément de S peut approximer un élément de T à n'importe quel degré de précision.

Dans le théorème ci-dessous, T et X correspondant à C^r ou M^r, S correspond à $\sum^r(G)$ ou $\sum \Pi(G)$ pour un choix spécifique de G, et ρ est choisi d'une manière appropriée.

Le premier résultat est déclaré en termes de la métrique suivante sur C^r.

Définition 2.7 : *Un sous-ensemble S de C^r est dit être uniformément dense sur compact (on compacta) dans C^r si pour chaque sous-ensemble compact $K \subset R^r$, S est ρ_k dense dans C^r, où pour f, $g \in C^r$ $\rho \in C^r$ $\rho_k(f, g) \equiv \sup_{\lambda \in K} |f(x) - g(x)|$.*

Une séquence des fonctions $\{f_n\}$ converge vers une fonction f uniformément sur un compact si pour tout compact $K \subset R^r$ $\rho_k(f_n, f) \to 0$ quand $n \to \infty$.

Le premier résultat:

Théorème 2.2 : *Soit G une fonction continue non-constante de $R \to R$, alors :*
$\sum \Pi^r(G)$ est uniformément dense dans le compact dans C^r.

En d'autres termes, les réseaux $\sum \Pi$ feed-forward sont capables d'une approximation arbitrairement précise à n'importe quelle fonction continue à valeurs réelles à travers un ensemble compact.

L'éxigence de l'ensemble compact tient, quand les valeurs possibles des entrées, x, sont bornées ($x \in K$).

Une propriété intéressante de ce résultat est que la fonction d'activation G peut être n'importe quelle fonction continue non-constante.

Il n'est pas nécessaire qu'elle soit une fonction sigmoïde (squashing), bien que c'est certainement permis.

Définition 2.8 : *Soit μ une mesure de probabilité sur (R^r, B^r). Si f et g appartiennent M^r, on dit qu'elles sont $\mu-$équivalentes si :*

$$\mu\{x \in R^r : f(x) = g(x)\} = 1 \tag{A.8}$$

Prenons μ pour une mesure de probabilité est une question de commodité. Les résultats, en fait, tiennent pour des mesures arbitraires finies. Le contexte n'a pas besoin d'être probabiliste.

Quand même, la mesure μ décrit la fréquence relative de l'occurence de l'entrée "pattern x".

La mesure μ est "l'espace d'environnement des entrées" dans la terminologie de *White* (1988a).

Les fonctions qui sont μ équivalentes diffèrent seulement sur un ensemble de pattern se produisant avec une mesure de probabilité 0, et nous sommes seulement concernés par la distingtion entre les classes de fonctions équivalentes.

La métrique sur des classes de fonctions μ équivalentes utiles pour nos principaux résultats est donnée par lé définition suivante :

Etant donnée une mesure de probabilité μ sur (R^r, B^r), on définit la métrique ρ_μ de $M^r \mathrm{x} M^r$ par :

$$\rho_\mu(f, g) = \inf\{\varepsilon > 0,\ \mu\{|f(x) - g(x)| > \varepsilon\} < \varepsilon\} \tag{A.9}$$

Deux fonctions sont proches dans cette métrique si et seulement si il y a une toute petite probabilité qu'elles diffèrent significativement.

Dans le cas extreme où f et g sont μ-équivalentes, $\rho_\mu(f, g)$ égale à zéro.

Il y a plusieurs manières de décrire ce que signifie pour $\rho_\mu(f_n, f)$ de converger vers zéro.

Lemme 2.1 : *Tous les suivants sont équivalents :*

(a) $\rho_\mu(f_n, f) \to 0$

(b) pour chaque $\varepsilon > 0$, $\mu\{x : |f_n(x) - f(x)| > \varepsilon\} \to 0$

(c) $\int \min\{|f_n(x) - f(x)|, 1\}\, \mu(dx) \to 0$

A partir de *(b)*, nous voyons que la convergence ρ_μ est équivalente à la convergence en probabilité (ou mesure).

En *(b)*, la métrique euclidienne peut-être remplacée par n'importe quelle métrique sur R générant la topologie euclidienne et l'intégrale en *(c)* par n'importe quelle métrique bornée sur R générant la topologie euclidienne.

Par exemple $d(a,b) = |a-b| / (1+|a-b|)$ est une métrique bornée générant la topologie euclidienne et *(c)* est vraie si et seulement si $\int df_n(x), f(x)\mu(dx) \to 0$.

Le lemme suivant établit la convergence uniforme sur le compact à la convergence ρ_μ.

Lemme 2.2 : *Si* $\{f_n\}$ *est une séquence des fonctions dans* M^r *qui converge uniformément sur un compact vers la fonction* f *ainsi* $\rho_\mu(f_n, f) \to 0$.

Le premier résultat est fixé sur les fonctions d'approximation dans M^r. Ils découlent du théorème 3.2 et du lemme 3.2.

Théorème 2.3 : *Pour toute fonction continue non constante* G*, chaque* r*, et chaque mesure de probabilité* μ *sur* (R^r, B^r), $\sum \Pi^r(G)$ *est* ρ_μ*-dense dans* M^r.

En d'autres termes, une seule couche cachée d'un réseau feed-forward $\sum \Pi$ peut bien approximer n'importe quelle fonction mesurable arbitrairement, sans se soucier de la fonction continue non-constante G utilisée, sans se soucier de la diemesion de l'espace des entrée r, et sans se soucier de l'environnement de l'espace des entrées μ. Dans ce sens précis et satisfaisant, les réseaux $\sum \Pi$ sont des approximateurs universels.

La condition de continuité sur G exclut la fonction seuil $\Psi(\lambda) = 1_{\{\lambda \geq 0\}}$. Cependant, pour les fonctions "squashing" la continuité n'est pas nécessaire.

Théorème 2.4 : *Pour chaque fonction sigmoïde (squashing)* Ψ*, pour tout* r*, et toute mesure de probabilité* μ *ou* (R^r, B^r), $\sum \Pi^r(\Psi)$ *est uniformément dense sur le compact dans* C^r *et* ρ_μ*-dense dans* M^r.

A cause de leur structure simple, il est important de savoir que les plus simples réseaux $\sum \Pi$, les réseaux $\sum$ ont les mêmes capacités d'approximation.

Théorème 2.5 : *Pour toute fonction sigmoïde (squashing)* Ψ*, pour tout* r*, et toute mesure de probabilité* μ *ou* (R^r, B^r), $\sum{}^r(\Psi)$ *est uniformément dense sur le compact dans* C *et* ρ_μ*-dense dans* M^r.

En d'autres termes, les réseaux feed-forward standard avec une seule couche cachée peuvent approximer n'importe quelle fonction continue uniformément sur n'importe quel ensemble compact et n'importe quelle fonction mesurable arbitrairement dans la ρ_μ métrique, sans se soucier de la fonction "sigmoïde (squashing)" Ψ (continue ou non), sans se soucier de la dimension de l'espace des entrées r, et sans se soucier de l'environnement de l'espace μ. Ainsi, les réseaux $\sum$ sont aussi des approximateurs universels.

Le théorème 3.5 implique le théorème 3.4 et pour les fonctions sigmoïdes (squashing), le théorème 3.4 implique le théorème 3.3.

La déclaration des résultats dans l'ordre donné reflète l'ordre naturel de leur preuve. De plus, la dérivée du théorème 3.3 comme une conséquence du théorème 3.4 cache sa simplicité.

La structure de la preuve du théorème 3.4 (respectivement 3.5) révèle qu'un résultat similaire tient si Ψ n'est pas restreinte à être une fonction "sigmoïde (squashing)", mais est n'importe quelle fonction mesurable telle que $\sum \Pi^r(\Psi)$ (respectivement $\sum{}^r(\Psi)$) et peut uniformément approximer des fonctions sigmoïdes (squashing) dans un compact. *Stinchombe* et *White* (1989) ont donné un résultat analogue au théorème 3.5 pour les fonctions d'activation des couches cachées non sigmoïdes.

Corollaire 2.1 : *Pour chaque fonction g dans M^r, il ya un sous-ensemble compact K de R^r et une $f \in \sum^r(\Psi)$ telle que pour tout $\varepsilon > 0$, on a $\mu(K) < 1 - \varepsilon$ et pour tout $x \in K$ on a :*

$$|f(x) - g(x)| < \varepsilon \tag{A.10}$$

sans se soucier de Ψ, r, ou μ.

En d'autres termes, il y a un seul réseau feed-forward à couche cachée qui approxime n'importe quelle fonction mesurable à un degré d'exactitude désiré sur l'ensemble compact K des entrées (patterns) que le même degré de précision a été mesuré (probabilité de se produire) 1.

Noter la différence entre le corollaire 3.1. et le théorème 3.2 : Dans le théorème 3.2, g est continue et K est un ensemble arbitrairement compact. Dans le corollaire 3.1, g est mesurable et K doit être choisi spécialement.

Le prochain résultat est relatif à l'approximation dans les espaces L_p. Nous rappelons la définition suivante :

Définition 2.9 : *$L_p(R^r, \mu)$ (ou simplement L_p) est l'ensemble de $f \in M^r$ telles que :*

$$\int |f(x)|^p \, \mu(dx) < \infty \tag{A.11}$$

La norme L_p est définie par :

$$\|f\|_p = \left[\int |f(x)|^p \, \mu(dx) \right]^{1/p} \tag{A.12}$$

La métrique associée sur L_p est définie par :

$$\rho_p(f,g) = \|f-g\|_p \tag{A.13}$$

Le résultat de l'approximation L_p est le suivant :

Corollaire 2.2 : *S'il y a un sous-ensemble compact K de R^r tel que : $\mu(K) = 1$ alors $\sum^r(\Psi)$ est ρ_p-dense dans $L_p(R^r, \mu)$ pour tout $p \in [1, \infty)$ sans se soucier de Ψ, r, ou μ.*

On obtient aussi ce résultat immédiatement :

Corollaire 2.3 : *Si μ est la mesure de probabilité sur $[0,1]^r$ alors $\sum^r(\Psi)$ est ρ_p-dense dans $L_p([0,1]^r, \mu)$ pour tout $p \in [1, \infty)$ sans se soucier de Ψ, r, ou μ.*

Corollaire 2.4 : *Si μ est un multiple de 1 (puts mass 1) sur un ensemble fini de points, alors pour chaque $g \in M^r$ et pour chaque $\varepsilon > 0$, il y'a une $f \in \sum^r(\Psi)$ telle que $\max_{x\in\{0,1\}^r} |g(x) - f(x)| < \varepsilon$.*

En fait, une représentation exacte des fonctions avec un soutien (support) fini est possible avec une seule couche cachée.

Théorème 2.6 : *Soit $\{x_1, ..., x_n\}$ un ensemble de points distincts dans R^r et soit $g : R^r \to R$ une fonction arbitraire. Si Ψ donne 0 et 1, alors il existe une fonction $f \in \sum^r(\Psi)$ avec n couches cachées tel que $f(x_i) = g(x_i)$, $i \in \{1, ..., n\}$.*

Avec quelques modifications fastidieuses, la preuve de ce théorème est établie lorsque Ψ est une fonction sigmoïde (squashing).

Les résultats précédents sont relatifs à des réseaux à une seule couche.

Des résultats analogues sont valables pour des réseaux à plusieurs sorties approximant des fonctions mesurables ou continues de $R^r \to R^s, s \in N$ dénotant $C^{r,s}$et $M^{r,s}$ respectivement .

Nous prolongons $\sum^r$et$\sum \Pi^{r,s}$ réspectivement à réinterprèter β_j comme un vecteur $s \times 1$ dans les définitions 3.2 et 3.4**.**

La fonction $g : R^r \to R^s$ a des élements $g_i, i = 1, ..., n$.

Nous avons les résultats suivants :

Corollaire 2.5 : *Les théorèmes 3.4 et 3.5 et les corollaires 3.1-3.4 restent valides pour les classes $\sum \Pi^{r,s}(\Psi)$ et/ou les fonctions d'approximation $\sum^{r,s}(\Psi)$ dans $C^{r,s}$et $M^{r,s}$, avec ρ_μ remplacé par ρ^s_μ, $\rho^s_\mu(f,g) \equiv \sum_{i=1}^{s} \rho_\mu(f_i, g_i)$ et avec ρ_p remplacé par sa généralisation multipliée et variée.*

Ainsi les réseaux feedforward multicouches à plusieurs sorties sont des approximateurs universels des fonctions à valeur vecteur.

Tous les résultats précédents sont pour des réseaux à une seule couche cachée.

Le dernier résultat décrit les capacités de l'approximation des réseaux multicouches à plusieurs sorties.

Pour simplifier, on considère explicitement le cas des réseaux multicouches $\sum$ seuleument .

Notons la classe des fonctions sorties pour les réseaux feedforward multicouches avec l couches (sans compter la couche d'entrés , mais en comptant la couche de sortie) de $R^r \to R^s$ en utilisant des fonctions sigmoïdes (squashing) Ψ comme $\sum_l^{r,s}(\Psi)$ (notre précéent résultat concerne ainsi le cas ou l=2)

Les lois d'activation pour des éléments de tels réseaux sont :

$$a_{kl} = G_k(A_i(a_{k-1}))\ i = 1, ...q_k\ ;\ k = 1, ..., l \tag{A.14}$$

où a_k est un vecteur $q_k \times 1$ avec a_{ki} élements , $a_0 = x$ par convention , $G_1, ..., G_{l-1} = \Psi, G_l$ est la fonction identité , $q_0 \equiv r$, et $q_l \equiv s$.

Nous avons le résultat suivant :

Corollaire 2.6 : *Le théorème 3.5 et les corollaires 3.1-3.5 restent valables pour les classes multicouches à plusieurs sorties* $\sum_l^{r,s}(\Psi)$ *approximant les fonctions dans* $C^{r,s}$*et* $M^{r,s}$, *avec* ρ_μ *et* ρ_p *remplacés comme dans le corrolaire 3.5, pour* $l \geq 2$.

Ainsi les réseaux $\sum_l^{r,s}$ sont des approximateurs universels des fonctions avec des valeurs vecteurs.

On remarque que tout calcul d'un réseau $\sum_l^{r,s}$ est aussi un approximateur universel puisqu'il contient les réseaux $\sum_l^{r,s}$ comme un cas spécial.

A.1 Conclusion

Ces résultats établissent que les réseaux standard feed-forward multicouches sont capables d'approximer toute fonction mesurable.

Hornik a ainsi établi que de tels réseaux de fonctions sont des approximateurs universels. Ceci implique que tout manque de succés dans les applications doit surgir d'un apprentissage inadéquat, d'un nombre insuffisant d'unités cachées ou d'un manque d'une relation déterministé entre les entrées et la cible (sortie désirée).

Les résultats donnés ici fournissent une base fondamentale pour établir rigoureusement la capacité des réseaux feed-forward multicouches d'apprendre (ie pour estimer avec cohérence) les forces de connexions qui rendent les approximations prouvées ici possibles.

RESUME DE THESE DE MAGISTER

LE PERCEPTRON MULTICOUCHES APPROXIMATEUR UNIVERSEL

ABSTRACT :

Dans ce travail on donne les fondements mathématiques des réseaux de neurones qui ont connu de grandes applications dans les statistiques , on s'intéresse au Perceptron Multicouches en sa qualité d'approximateur universel.

L'approximation de fonctions est une des Utilisations les plus courantes des réseaux de neurones artificiels.

Le cadre général de l'approximation est le suivant : on suppose l'existence d'une relation entre plusieurs variables(les entrées) et une variable de sortie, cette relation étant inconnue , on essaie de construire un approximateur « boite noire » entre ces entrés et cette sortie.

Dans ce mémoire nous présentons des résultats qui montrent que le PMC est un approximateur universel parcimonieux pour les fonctions non linéaire , continues et dérivables.

Nous étudions aussi la complexité des algorithmes d'apprentissage on définit les classe P,NP, et NP complet et on montre que les problèmes d'apprentissage sont NP complet.