République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université M'hamed Bougara de Boumerdes

Faculté des Sciences
Département des Mathématiques

Mémoire de fin d'études

En vue de l'obtention du diplome de master en Recherche Opérationnelle

Option :

Recherche Opérationnelle, Optimisation et Management Stratégique

Thème

Optimisation dans les réseaux de neurones, cas de perceptron multicouche

Réalisé par :

M^{lle}. Abdellahoum Khouloud
M^{lle}. Boucheneb Amira

Le jury composé de :

M^{me}. Drici W.	M.C.B	*U.M.B.B.*	**Président**
M. Issaadi B.	M.C.A	*U.M.B.B.*	**Promoteur**
M. Cheurfa F.	M.A.A	*U.M.B.B.*	**Co-encadreur**
M^{me}. Ouatiki S.	M.C.B	*U.M.B.B.*	**Examinateur**
M^{me}. Fass R.	M.A.A	*U.M.B.B.*	**Examinateur**

Année universitaire 2020/2021

Remerciment

Nous remercions Allah le tous puissant, qui nous a donné la force, la patience, le courage et la volonté d'achever ce travail. Et qui nous permis de concilier entre nos vie familiale, professionnelle et nos études.

Nos remerciements les plus sincères à notre promoteur ***Mr Issaadi.B*** *pour toute sa confiance et l'aide précieuse qu'il nous a apporté tout au longue de la réalisation de ce travail.*

Ce mémoire n'aurait jamais pu voir le jour sans le soutien actif des membres de notre famille, surtout nos parents qui nous ont toujours encouragé moralement et à qui on tient à les remercier.

Nous exprimons nos sincères remerciements aux membres du jury de nous avoir honorées en acceptant d'examiner et évaluer notre travail.

Nous tenons à remercier sincèrement ***Mr Cheurfa.F****, en tant que Co-encadreur, pour ses remarques et suggestions pour améliorer la qualité de ce mémoire.*

Nos remerciements vont en général à tous les enseignants de la faculté des sciences M'Hamed Bougara, et plus exceptionnellement à ceux du département des MATHEMATIQUES.

Nous adressons nos remerciemments à ***Mr YACINE Ferani****, Chef département des MATHEMATIQUES, pour ses multiples conseils, sa générosité et son soutien immense.*

En fin on tient à exprimer vivement nos remerciements avec une profonde gratitude à toutes les personnes qui ont contribué de près au de loin pour l'élaboration de ce travail.

Dédicaces

Je remercie en premier lieu, Allah de m'avoir donné la confiance, la sagesse, la patience et la force d'achever ce travail, ainsi que l'audace pour surmonter toutes les difficultés.

Je dédie ce mémoire

À tous les membres de ma famille pour tous les sacrifices consentis, pour leur soutien, qui n'ont pas cessé de m'encourager durant toutes mes années d'études pour que j'arrive à ce jour-là. Ils sont les plus chers à mon cœur, avec toute ma gratitude.

À tous mes amis, promotion ROOMS 2020 et 2021

À tous mes enseignants de la spécialité recherche opérationnelle.

À ceux qui ont cru en l'accomplissement de ce travail.

À celui animé par le désir d'apprendre et de découvrir.

À tous ceux qui me sont chers.

Khouloud

Tout d'abbord,je remercie Allah qui m'a apprécié et m'a permis de terminer ce travail .
Je dédie ce mémoire à

Mes trés chers parents,ma source de vie,d'espoire,et de motivation, qui m'ont permettent d'être ce que je suis et qui n'ont jamais cessés de croire en moi.

Mon mari avec amour et gratitude.

Ma soeur et mes frères.

Toute la promotion ROOMS 2020/2021 et tout les enseignants de département mathématique.

Tous mes proches amis. Toute la famille. A tous ceux qui me sont chers et à tous les gens qui ont été là pour moi...

Ma trés chère binome Khouloud ,celle avec qui j'ai partagé une partie importante de ma vie.

Vous chers lecteurs

Amira

Résumé

PMC, qui est l'un des classificateurs les plus couramment utilisés, est une topologie de réseau de neurones supervisée. L'algorithme de rétropropagation est utilisé pour minimiser l'erreur entre la sortie du réseau et la valeur cible. Selon le processus de classification, la structure PMC et les paramètres d'apprentissage, qui sont utilisés dans l'algorithme de rétropropagation, sont nécessaires pour décider d'augmenter la précision du test. Généralement, ces variables sont choisies au hasard, donc trouver les valeurs qui donnent une précision de test maximal est un processus qui prend du temps. Dans ce mémoire de master, les paramètres d'apprentissage de l'algorithme de rétropropagation et de la structure du réseau sont optimisés pour réussir un processus de mise à jour de poids plus rapide et efficace en utilisant trois algorithmes d'optimisation heuristiques différents, ABC, AG et RS. Les deux ensembles de données utilisées contiennent des données de capteurs d'activité humaine. Pour deux jeux de données, trois algorithmes sont comparés et des résultats de test détaillés sont fournis. Il est observé que, bien que RS soit l'algorithme le plus rapide parmi les algorithmes choisis, ABC affiche les performances les plus élevées pour la précision des tests du classificateur PMC.

Mots clés : Perceptron multicouche, algorithmes d'optimisation heuristiques, classification des données.

Abstract

MLP which is one of the most commonly used classifier, is a feed-forward, supervised neural network topology. Back propagation algorithm is used for minimizing the error between network output and the target value. According to classification process, MLP structure and learning parameters, which are used in back propagation algorithm, are needed to decide for increasing the test accuracy. Commonly these variables are chosen randomly, so finding the values that give maximum test accuracy is a time-consuming process. In this master thesis, learning parameter of back propagation algorithm and network structure are optimized to success a faster and efficient weight-update process by using three different heuristic optimization algorithm, ABC, GA and SA. Both of the used two datasets contain human activity sensor data. For two datasets, three algorithms are compared and detailed test results are given. It is observed that, although SA is the fastest one among chosen algorithms, ABC shows the highest performance for test accuracy of MLP classifier.

Keys words : Multi-layer perceptron, heuristic optimization algorithm, data classification.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Introduction générale

Nous entendons beaucoup parler de ce qu'on appelle l'intelligence artificielle, qui est devenue le sujet de conversation du monde moderne en raison du grand développement dont il a été témoin dans divers domaines, car elle est devenue l'une des formes les plus importantes de développement de la recherche scientifique et entretient une relation étroite avec l'apprentissage automatique.

Ces réseaux de neurones artificiels sont des algorithmes d'apprentissage et des programmes de systèmes informatiques qui en principe, dépendent de la simulation du travail des neurones du cerveau humain pour traiter des données et accomplir des tâches dans divers domaines, la théorie des réseaux neuronaux a travaillé pour mieux déterminer le fonctionnement des neurones dans le cerveau.

Le développement de réseaux de neurones artificiels découle du désir des chercheurs de comprendre et d'imiter les capacités du cerveau, de la mémoire, de l'apprentissage et du traitement parallèle pour construire des systèmes artificiels capables de remplacer les humains dans l'exécution de tâches complexes. Les réseaux de neurones sont considérés comme l'un des models mathématiques et des algorithmes d'apprentissage les plus importants dans le domaine de l'intelligence artificielle et sont connus pour leur capacité à apprendre et à généraliser [33].

L'intelligence artificielle est l'ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence humaine, elle correspond donc à un ensemble de concepts et de technologies plus qu'à une discipline autonome constitue. Souvent classée dans le groupe des sciences cognitives, elle fait appel à la neurobiologie computationnelle (particulièrement aux réseaux neuronaux), à la logique mathématique (partie des mathématiques et de la philosophie) et à l'informatique. Elle recherche des méthodes de résolution de problèmes à forte complexité logique ou algorithmique.

Les réseaux de neurones sont apparus dans les années cinquante mais n'ont reçu cependant un intérêt considérable qu'à partir des années quatre-vingt avec l'apparition de l'algorithme de rétropropagation et grâce aux résultats théoriques et pratiques obtenus au cours des dernières décennies, ils sont devenus un outil très utilisé dans divers domaines ; ils ont prouvé leur efficacité dans la reconnaissance de formes, l'identification des paramètres, la commande des procédés et le traitement de signal.... Les réseaux de neurones demeurent toutefois un sujet d'un grand intérêt pour les chercheurs qui désirent améliorer leurs performances et étendre leur champ d'applications.

L'identification des paramètres du réseau de neurones, appelés poids de connexion, est souvent obtenue par l'utilisation de l'algorithme de rétropropagation, back propagation algorithme BP, basé sur la méthode de la descente du gradient, dont l'objectif est la minimisation de l'erreur d'apprentissage. Cependant, la surface de l'erreur est souvent complexe et présente des caractéristiques peu satisfaisantes pour réaliser une descente du gradient ce qui crée des inconvénients, tels que : la lenteur de la convergence, la sensibilité aux minima locaux et la difficulté à régler les paramètres d'apprentissage.

Les métaheuristiques, souvent inspirées à partir des systèmes naturels, sont apparues dans les années quatre-vingt et forment une famille d'algorithmes d'optimisation visant à résoudre des problèmes d'optimisation complexes, pour lesquels on ne connaît pas de méthode classique efficace. Parmi les métaheuristiques les plus connues, on y trouve : les algorithmes génétiques, les colonies

d'abeilles, la recherche taboue et le recuit simulé.
Dans ce mémoire on s'intéresse aux réseaux de Neurones Multicouches PMC, qui sont souvent utilisés, et ceci est dû à leurs simplicités et leurs propriétés d'approximation universelle, ils sont utilisés pour la classification, en raison de leur capacité à résoudre des problèmes non linéaires avec beaucoup de succès. Pour une tache de classification, le réseau neuronal artificiel cherche une structure de réseau avec un ensemble de paramètres à l'aide d'un algorithme d'apprentissage.

Parmi les algorithmes les plus connus en réseaux de neurones on a le Perceptron Multicouche (PMC), il est aussi l'un des réseaux de neurones les plus utilisés actuellement, pour la classification supervisée notamment en raison de leur capacité à résoudre des problèmes non linéaires avec beaucoup de succès.

Le travail a pour sujet l'étude des réseaux de neurones, plus particulièrement la formulation mathématique du modèle.On a présenté dans ce mémoire concerne l'optimisation par un réseau de neurones MLP, a fin de classifier des données ,on se concentre sur les algorithmes d'optimisation utilisés dans cette étude, les ensembles de données sont présentés et le processus d'optimisation est expliqué. Enfin, dans la dernière section, les résultats sont donnés. en utilisant un algorithme Génétique un autre Algorithme dit colonies d'abeilles,et la méthode de recuit simulé et on a fait une comparaison des résultats de ces méthodes pour voir le meilleur algorithme qui donne des meilleurs performances avec un taux d'erreur minimum, notre objectif est d'exploiter les avantages de chacun d'entre eux pour aboutir à un algorithme d'optimisation permettant un bon entrainement du réseau de neurone MLP pour une meilleur classification de données. Les résultats issus de cette étude étaient très performants.

Ce mémoire est organisé comme suit :
Le premier chapitre on a présenté une introduction sur les réseaux de neurones artificielles ,comme on a mentionné un rappelle sur leurs principales notions théoriques,il commence par un historique sur ces réseaux aprés une présentation de la structure du neurone biologique et du neurone formel, puis il accède aux formalisme et à l'apprentissage des réseaux de neurones et il s'achève avec l'utilité et les domaines d'application des réseaux de neurones.
Le deuxième chapitre on a présenté le perceptron l'un des algorithmes des réseaux de neurone qui suit un apprentissage supervisé et on définit aussi quelque algorithmes utilisés dans l'apprentissage supervisé et les plus interéssantes règles d'apprentissage utilisés pour la modification des poids synaptique.
le troisième chapitre on a présenté le perceptron multicouche PMC on a commencé par sa définition ,structure ,et fonctionnement aprés on a parlé de son apprentissage, on a défint l'algorithme d'aprentissage qu'il suit dans la modification des poids synaptique qui est l'algorithme de rétropropagation de gradient .
Le dernier chapitre on a parlé de classification supervisé l'un des applications des réseaux de neurones,on a présenté les méthodes de classification ainsi que leur différentes techniques et domaine d'applications puis on a ajouté quelque application de classification par un réseau de perceptron multicouche à l'aide du logiciel Matlab. Aprés on a définit quelques algorithmes utilisés dans l'optimisation d'un réseaux multicouche PMC .Puis nous avons utilisé une méthode efficace pour optimiser le Perceptron multicouche par des algorithmes d'optimisation. Les trois algorithmes sont comparés et des résultats de test détaillés sont fournis.

Finalement, et pour clôturer ce mémoire, on a une conclusion générale qui récapitule les travaux de l'ensemble des chapitres.

Chapitre

1 Introduction aux réseaux de neurones artificiels

Sommaire

1.1 Introduction

Ces dernières années, l'utilisation du terme de réseaux de neurones artificiels a considérablement augmenté, car ils a été témoin d'un développement remarquable dans divers domaines depuis sa découverte et le début de recherches ultérieures, où le premier neurone artificiel a été proposé par Mc Culloch et Pitts en 1943 qui l'a inspiré par la façon dont les systèmes de neurones biologiques traitent les données .

Dans ce chapitre on va parler beaucoup plus sur les réseaux de neurones artificielle et sa grande utilité dans l'apprentissage automatique qui est un pilier très important en intelligence artificielle.

1.2 Historique

— 1890 : W. James, célèbre psychologue américain introduit le concept de mémoire associative, et propose ce qui deviendra une loi de fonctionnement pour l'apprentissage sur les réseaux de neurones connue plus tard sous le nom de loi de Hebb.
— 1943 : Mc Culloch et Pitts laissent leurs noms à une modélisation du neurone biologique (un neurone au comportement binaire). Ceux sont les premiers à présenter le premier neurone formel et à montrer que des réseaux de neurones formels simples peuvent réaliser des fonctions logiques, arithmétiques et symboliques complexes (tout au moins au niveau théorique). présentent le premier neurone formel.
— 1949 :D. Hebb,physiologiste américain propose un mécanisme d'apprentissage et il a expliqué le conditionnement chez l'animal par les propriétés des neurones eux-mêmes. Ainsi, un conditionnement de type pavlovien tel que, nourrir tous les jours à la même heure un chien, entraîne chez cet animal la sécrétion de salive à cette heure précise même en l'absence de nourriture. La loi de modification des propriétés des connexions entre neurones qu'il propose explique en partie ce type de résultats expérimentaux. Hebb .
— 1958 : Rosenblatt présente le premier réseau de neurones artificiels : le Perceptron. Il est inspiré du système visuel, et possède deux couches de neurones : perceptive et décisionnelle. il a construit le premier neuroordinateur basé sur ce modèle et l'applique au domaine de la reconnaissance de formes. Notons qu'à cet époque les moyens à sa disposition sont limités et c'est une prouesse technologique que de réussir à faire fonctionner correctement cette machine plus de quelques minutes.
— 1960 : B. Widrow, un automaticien, développe le modèle Adaline (Adaptative Linear Element). Dans sa structure, le modèle ressemble au Perceptron, cependant la loi d'apprentissage est différente. Celle-ci est à l'origine de l'algorithme de rétropropagation de gradient très utilisé aujourd'hui avec les Perceptrons multicouches. Les réseaux de type Adaline restent utilisés de nos jours pour certaines applications particulières. B. Widrow a créé dès cette époque une des premières firmes proposant neuro-ordinateurs et neuro-composants, la "Memistor Corporation". Il est aujourd'hui le président de l'International Neural Network Society (INNS) sur laquelle nous reviendrons au chapitre Informations pratiques.
— 1969 : M. Minsky et S. Papert publient un ouvrage qui met en exergue les limitations théoriques du perceptron. Limitations alors connues, notamment concernant l'impossibilité de traiter par ce modèle des problèmes non linéaires. limites du Perceptron ; besoin d'architectures plus complexes, mais comment effectuer leur apprentissage.
— 1967-1982 : Toutes les recherches ne sont, bien sûr, pas interrompues. Elles se poursuivent, mais déguisées, sous le couvert de divers domaines comme : le traitement adaptatif du signal,

la reconnaissance de formes, la modélisation en neurobiologie, etc. De grands noms travaillent durant cette période tels : S. Grossberg, T. Kohonen, ... dont nous reparlerons.

— 1982 : J. J. Hopfield est un physicien reconnu à qui l'on doit le renouveau d'intérêt pour les réseaux de neurones artificiels. A cela plusieurs raisons : Au travers d'un article court, clair et bien écrit, il présente une théorie du fonctionnement et des possibilités des réseaux de neurones. Il faut remarquer la présentation anticonformiste de son article. Alors que les auteurs s'acharnent jusqu'alors à proposer une structure et une loi d'apprentissage, puis à étudier les propriétés émergentes ; J. J. Hopfield fixe préalablement le comportement à atteindre pour son modèle et construit à partir de là, la structure et la loi d'apprentissage correspondant au résultat escompté. Ce modèle est aujourd'hui encore très utilisé pour des problèmes d'optimisation. D'autre part, entre les mains de ce physicien distingué, la théorie des réseaux de neurones devient respectable. Elle n'est plus l'apanage d'un certain nombre de psychologues et neurobiologistes hors du coup. Enfin, une petite phrase, placée en commentaire dans son article initial, met en avant l'isomorphisme de son modèle avec le modèle d'Ising (modèle des verres de spins). Cette idée va drainer un flot de physiciens vers les réseaux de neurones artificiels. Notons qu'à cette date, l'IA est l'objet d'une certaine désillusion, elle n'a pas répondue à toutes les attentes et s'est même heurtée à de sérieuses limitations. Aussi, bien que les limitations du Perceptron mise en avant par M. Minsky ne soient pas levées par le modèle d'Hopfield, les recherches sont relancées.
— 1983 : La Machine de Boltzmann est le premier modèle connu apte à traiter de manière satisfaisante les limitations recensées dans le cas du perceptron. Mais l'utilisation pratique s'avère difficile, la convergence de l'algorithme étant extrêmement longue (les temps de calcul sont considérables).
— 1986 : Rétropropagation (Rumelhart et McClelland) Rumelhart popularise l'algorithme de rétropropagation du gradient, conçu par Werbos, Dès cette découverte, nous avons la possibilité de réaliser une fonction non linéaire d'entrée/sortie sur un réseau en décomposant cette fonction en une suite d'étapes linéairements séparables,cet algorithme permet d'entraîner les couches cachées des réseaux multicouches. Nouvelles architectures de réseaux de neurones applications reconnaissance de l'écriture reconnaissance et synthèse de la parole vision (traitement d'images).
— 1990 : Société de l'Information Nouvelles applications recherche et filtrage d'information sur le Web extraction d'information, veille technologique Multimédia (indexation, ...) datamining besoin de combiner plusieurs modèles.[11]

Au fil du temps,le modèle de perceptron multicouche a continu à développer notamment avec l'apparition de nouvelles fonctions d'activations telle que la fonction sigmoïde,la fonction tangente hyperbolique,ou encore la fonction rectifieur,ces fonctions ont aujourd'huit totalement remplacés la fonction Heavside qu'on a vus car ils offrent de bien meilleur performarance,dans les années 90 on commença à développer les premieres variantes de perceptron multicouche ,le célèbre Yann LeCun inventa les premiers réseau de neurone convolutifs qui sont capable de reconnaitre de traiter des images en introduisant au début de ces réseau des filtres mathématiques qu'on appelle convolusion et pouling. c'est également durant ces années qu'on a vu les premiers réseaux de neurones récurents qui sont une variante de perceptron multicouche et qui permettre de traiter efficasement des problèmes de série temporelle comme la lecture de textes ou la reconaissance vocale.

La question qui se pose dans nos tete c'est, si tout ça existe dans les années 90 pourquoi vouloir attendre aussi lontemps avant de voir émerger les technologies qu'on a aujourd'huit et bien il ya deux grandes raisons à cela :

• première que pour bien fonctionner un réseau de neurone on doit etre entrainer le avec une trés grosse quantités de données dépassant par fois millions voir les dizaine de millions de données et malheureusement dans les années 90,et bien ne disposer pas de tant de données des millions ou bien de dizaine de millions de données toutes bien classés et bien repertoriés ,il a falu attendre l'arrivé d'internet ,les smart phones et des objets connectés pour commencer à collecter de grandes quantités de données d'images et de sens qui vont etre exploité pour le deep learning.

• La deuxième raison il a falu attendre sii lontemps avant de pouvoir réellement utiliser des réseaux

de neurone c'est par ce que la puissance les ordinateurs des années 80 et 90 ne le permettent tout simplement pas(une grande puissance de calcul est nécessaire) car entrainer un réseau de neurone sa demande pas mal du temps et pas mal de puissance d'ou il a fallu attendre qu'on dispose d'excelents GPE et CPE (applications avancées) pour enfin obtenir de bons résultats.

En faite le deep learning n'a réellement pris son envol qu'en 2012 dans une compétition de vision nomée ImageNet une équipe de chercheurs muni par Georffrey Hinton développa un réseau de neurone capable de reconnaitre n'importe qu'elle image avec une meilleur performance que tout les autres algorithmes de l'époque,de puis ce jour tout le monde ne parle plus que de machine learning et de deep learning.

1.3 Le neurone biologique

Avant de parler sur le neurone biologique on va définir le cerveau humain qui a une grande liaison avec les réseaux de neurones artificielle et l'inteligence artificielle.

Le cerveau humain est le principal du système nerveaux des etre humains ,il est constitué d'environ 170 milliards de cellules dont 86 milliards de neurones en moyenne, qui peuvent chacun former de 5 à 60 000 synapses.

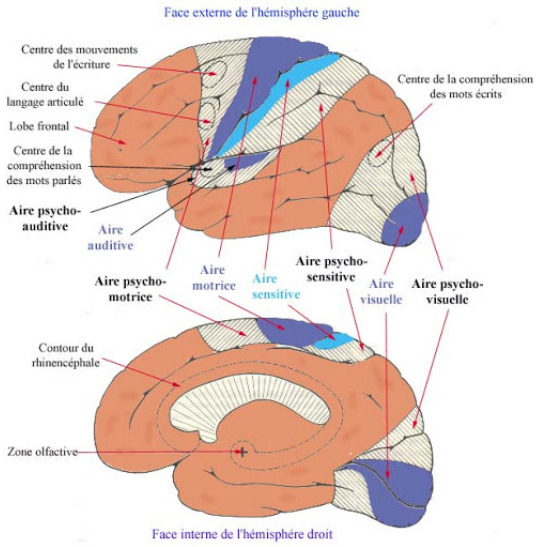


FIGURE 1.1 – Cerveau humain

Un neurone biologique est une cellule excitable constituant l'unité fonctionnelle de la base du système nerveux, spécialisée dans la communication et le traitement d'informations par des impulsions électriques (potentiels d'action).
Les neurones sont reliés entre eux par des liaisons appelées axones. Ces axones vont eux-mêmes jouer un rôle important dans le comportement logique de l'ensemble. Ils conduisent les signaux électriques de la sortie d'un neurone vers l'entrée (synapse) d'un autre neurone voir la figure 1.2. Les neurones font une sommation des signaux reçus en entrée et en fonction du résultat obtenu vont fournir un courant en sortie . Ils ont deux propriétés physiologique :

— l'excitabilité, c'est-à-dire la capacité de répondre aux stimulations et de convertir celles-ci en impulsions nerveuses.

— conductivité, c'est-à-dire la capacité de transmettre les impulsions.

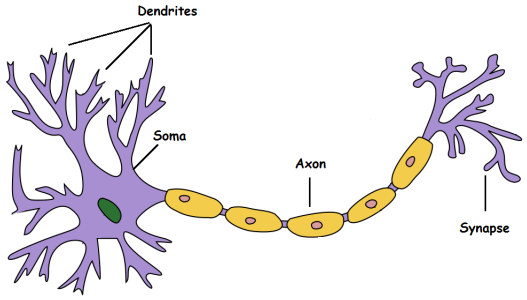


FIGURE 1.2 – Structure d'un neurone biologique

1.3.1 Structure des neurones :

Les neurones sont des cellules nerveuses décomposables en 4 parties principales :
— Un **corps cellulaire** ou péricaryon comportant le noyau, c'est l'unité de traitement.
— Très nombreuses ramifications de type **dendritiques** (d'où proviennent les informations) Les dendrites sont des prolongements du corps cellulaire des neurones.Les dendrites ont pour fonction de recevoir et de conduire l'influx nerveux (signal) provenant d'autres cellules nerveuses, vers le corps cellulaire du neurone.
— Un **axone** (par où sont diffusées les informations). L'axone est la partie où passent les messages accumulés dans le corps de la cellule.
— Des **synapses** ce sont des points de connexion par où passent les signaux de la cellule, par lesquelles la cellule communique avec d'autres neurones.

L'axone, ou fibre nerveuse, est le prolongement du neurone qui conduit le signal électrique du corps cellulaire vers les zones synaptiques. Les différentes dendrites de neurones entrent en contact avec l'axone pour transmettre l'information de cellule à cellule via des structures spécialisées : les synapses.

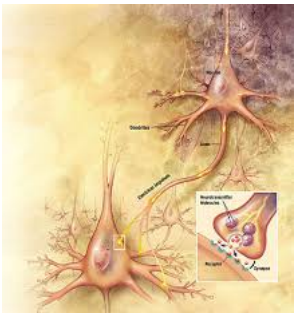


FIGURE 1.3 – Synapse

1.3.2 Fonction des neurones

Les neurones ont pour rôle de faire circuler les informations entre l'environnement et l'organisme, ou au sein de l'organisme. Un neurone est une cellule excitable, c'est-à-dire qu'un stimulus peut entraîner la formation dans la cellule d'un signal bioélectrique ou influx nerveux, qui pourra être transmis à d'autres neurones ou à d'autres tissus pour les activer (des muscles, des glandes sécrétrices...). Les neurones sont au nombre de 100 milliards dans le cerveau humain et sont donc capables

de créer un réseau incroyablement complexe, avec parfois plus de 100.000 synapses par neurone.

1.4 Formalisme d'un neurone élémentaire formel

La notion du neurone formel est inspirée des neurones bioloiques et leurs liaisons (les synapses) du cerveau humain. Le neurone formel imaginé par Warren McCulloch et Walter Pitts en 1943.

1.4.1 Le neurone formel

Un neurone formel est une représentation mathématique et informatique d'un neurone biologique. Le neurone formel est l'unité élémentaire des réseaux de neurones artificiels dans lesquels il est associé à ses semblables pour calculer des fonctions arbitrairement complexes, utilisées pour diverses applications en intelligence artificielle. Il s'inspire du neurone biologique et donc, comme lui, il réalise les opérations suivantes :
- réception des informations des neurones précédents
- somme pondérée
- réponse

Mathématiquement, le neurone formel est une fonction algébrique non linéaire et bornée, il possède généralement plusieurs entrées et une sortie qui correspondent respectivement aux dendrites et au cône d'émergence du neurone biologique (point de départ de l'axone). Les actions excitatrices et inhibitrices des synapses sont représentées, la plupart du temps, par des coefficients numériques (les poids synaptiques) associés aux entrées. Les valeurs numériques de ces coefficients sont ajustées dans une phase d'apprentissage.

Dans sa version la plus simple, un neurone formel calcule la somme pondérée des entrées reçues, puis applique à cette valeur une fonction d'activation, généralement non linéaire. La valeur finale obtenue est la sortie du neurone. à plusieurs variables et à valeurs réelles, dont la valeur dépend de paramètres appelés coefficients ou poids. Les variables de cette fonction sont habituellement appelées « entrées » du neurone, et la valeur de la fonction est appelée « sortie ».

1.4.2 Structure du neurone formel

— Un réseau de neurones est en général composé d'une succession de couches dont chacune prend ses entrées sur les sorties de la précédente.
— Chaque couche est composée de neurones, prenant leurs entrées sur les neurones de la couche précédente.
— À chaque synapse est associé un poids synaptique, de sorte qu'ils sont multipliés par ces poids, puis additionnés par les neurones de niveau , ce qui est équivalent à multiplier le vecteur d'entrée par une matrice de transformation.
— Mettre l'une derrière l'autre les différentes couches d'un réseau de neurones reviendrait à mettre en cascade plusieurs matrices de transformation et pourrait se ramener à une seule matrice, produit des autres, s'il n'y avait à chaque couche, la fonction de sortie qui introduit une non linéarité à chaque étape.
— Ceci montre l'importance du choix judicieux d'une bonne fonction de sortie : un réseau de neurones dont les sorties seraient linéaires n'aurait aucun intérêt.

— Au-delà de cette structure simple, le réseau de neurones peut également contenir des boucles qui en changent radicalement les possibilités mais aussi la complexité.
— De la même façon que des boucles peuvent transformer une logique combinatoire en logique séquentielle, les boucles dans un réseau de neurones transforment un simple dispositif de reconnaissance d'entrées en une machine complexe capable de toutes sortes de comportements.

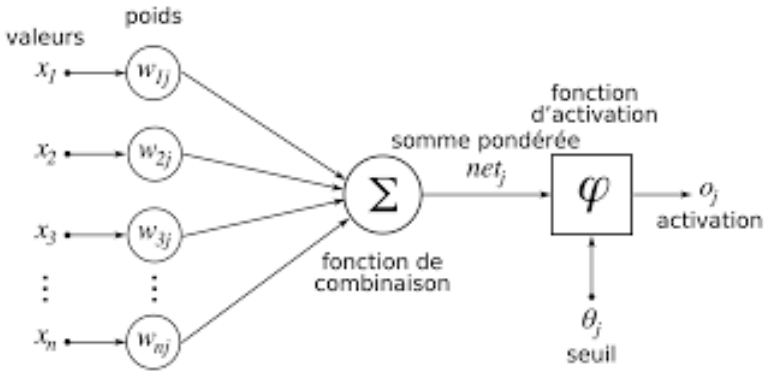


FIGURE 1.4 – structure d'un neurone formel

Le neurone calcule la somme de ses entrées x, pondérée par les poids w, puis cette valeur passe à travers la fonction d'activation ϕ pour produire sa sortie o.

1.4.3 Formulation mathématique

On considère le cas général d'un neurone formel à m entrées, auquel on doit donc soumettre les grandeurs numériques (ou signaux, ou encore stimuli)notées x_1 à x_m.

Un modèle de neurone formel est une règle de calcul qui permet d'associer aux m entrées une sortie : c'est donc une fonction à m variables et à valeurs réelles.

Neurone formel avec 2 entrées et une fonction d'activation à seuil.

Dans le modèle à chaque entrée est associé un poids synaptique, c'est-à-dire une valeur numérique notée de w_1 pour l'entrée 1jusqu'à w_m pour l'entrée m .

La première opération réalisée par le neurone formel consiste en une somme des grandeurs reçues en entrées, pondérées par les coefficients synaptiques, c'est-à-dire la somme :

$$w_1x_1 + ... + w_mx_m = \sum_{j=1}^{m} x_1x_j \tag{1.1}$$

À cette grandeur s'ajoute un seuil w_0 .Le résultat est alors transformé par une fonction d'activation non linéaire (parfois appelée fonction de sortie), . La sortie associée aux entrées x_1 à x_m est ainsi donnée par :

$$\varphi(w_0 + \sum_{j=1}^{m} w_1x_j)$$

qu'on peut écrire plus simplement :

$$\varphi(\sum_{j=0}^{m} w_jx_j)$$

En ajoutant au neurone une entrée fictive x_0 fixéé à la valeur 1. Dans la formulation, la fonction d'activation est la fonction de Heaviside (fonction marche d'escalier), dont la valeur est 0 ou 1.

Dans ce cas, on préfère parfois définir la sortie par la formule suivante qui est :

$$\varphi(\sum_{j=1}^{m} w_j x_j - w_0)$$

justifie le nom de seuil donné à la valeur w_0.

En effet, si la somme $\sum_{j=1}^{m} w_j x_j$ dépase w_0 la sortie du neurone est 1, alors qu'elle vaut 0 dans le cas contraire : w_0 est donc le seuil d'activation du neurone, si on considère que la sortie 0 correspond à un neurone « éteint ».

Donc un neurone est avant tout un opérateur mathématique, dont on peut calculer la valeur numérique par quelques lignes de programme informatique.

1.4.4 Modélisation d'un neurone formel

La modélisation consiste à mettre en œuvre un système de réseau de neurones sous un aspect non pas biologique mais artificiel, cela suppose que d'après le principe biologique on aura une correspondance pour chaque élément composant le neurone biologique, donc une modélisation pour chacun d'entre eux. On peut résumer cette modélisation à l'aide de ce tableau 1.1, qui nous permettra de voir clairement la transition entre le neurone biologique et le neurone formel.

Réseaux de neurones Biologique	Réseaux de neurones Artificiels
Soma	Neurone
Dendrite	Entrée
Axone	Sortie
Synapse	Poids

TABLE 1.1 – La transition entre le neurone biologique et le neurone formel.

Cette figure montre la structure d'un neurone artificiel. Chaque neurone artificiel est un processeur élémentaire. Il reçoit un nombre variable d'entrées en provenance de neurones amonts. A chacune de ces entrées est associée un poids w abréviation de weight (poids en anglais) représentatif de la force de la connexion. Chaque processeur élémentaire est doté d'une sortie unique, qui se ramifie ensuite pour alimenter un nombre variable de neurones avals. A chaque connexion est associée un poids.

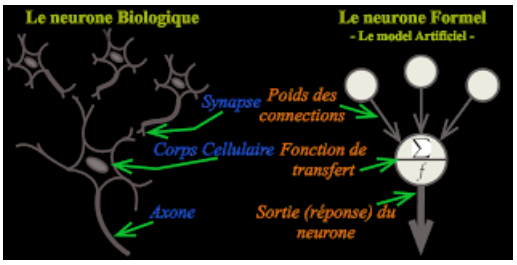


FIGURE 1.5 – Neurone biologique / Neurone artificiel.

1.4.5 Fonctions d'activation d'un neurone formel

Le terme de "fonction d'activation" vient de l'équivalent biologique "potentiel d'activation", seuil de stimulation qui, une fois atteint entraîne une réponse du neurone. La fonction d'activation est souvent une fonction non linéaire. Un exemple de fonction d'activation est la fonction de Heaviside, qui renvoie tout le temps 1 si le signal en entrée est positif, ou 0 s'il est négatif.
La fonction d'activation est souvent une fonction non linéaire. Un exemple de fonction d'activation est la fonction de Heaviside.

1.4.5.1 Fonction de Heaviside

En mathématiques, la fonction de Heaviside (également fonction échelon unité, fonction marche d'escalier), du nom d'Oliver Heaviside, est la fonction indicatrice de . C'est donc la fonction H (discontinue en 0) prenant la valeur 1 pour tous les réels positifs et la valeur 0 pour les réels strictement négatifs :

$$\forall x \in \mathbb{R}, H(x) = \begin{cases} 0 \quad si \quad \text{x<0} \\ Undef \quad si \quad \text{x=0} \\ 1 \quad si \quad \text{x>0.} \end{cases}$$

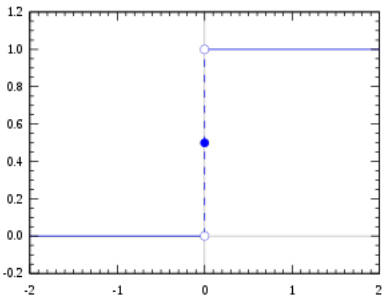


FIGURE 1.6 – Fonction de Heavisite

Dans les réseaux de neurones modernes, des fonctions avec de meilleures propriétés mathématiques (dérivabilité) ont remplacé la fonction de Heaviside.

Nous utilisons maintenant des fonctions telles que la sigmoïde, la tangente hyperbolique ou encore le ReLU. La ReLU (Rectified Linear Unit) est la fonction qui est aujourd'hui la plus utilisée. Le tableau 1.2 illustre quelques importantes fonctions d'activation.

La fonction d'activation du neurone qui définit son état en fonction de son entrée totale : Elle peut être :

- Une fonction binaire à seuil ; dans ce cas on s'arrange pour que la forme de la fonction soit telle qu'on puisse utiliser la fonction de Heaviside(figure) ou la fonction signe (figure) :
- Une fonction linéaire à seuil (figure) : Soit SATUR la fonction à seuil
- Une fonction sigmoïde (figure).
- La fonction de sortie qui calcule la sortie du réseau en fonction de son état d'activation ; en général cette fonction est considérée comme la fonction identité.

1.4.5.2 Fonction sigmoïde

La fonction sigmoïde (aussi appelée fonction logistique ou fonction en S) est définie par :

$$f_{sig}(x) = \frac{1}{1 + \exp -x}$$

pour tout réel x ,elle est symétrique par rapport au point (0,1/2),c'est le point d'inflexion qui vérifie elle tend vers 0 quand x tend vers $-\infty$ et vers 1 quand x tend vers $+\infty$. Elle possède plusieurs propriétés qui la rendent intéressante comme fonction d'activation. Elle n'est pas polynomiale. Elle est indéfiniment continûment dérivable et une propriété simple permet d'accélérer le calcul de sa dérivée. Ceci réduit le temps calcul nécessaire à l'apprentissage d'un réseau de neurones. On a en effet :

$$\frac{d}{dx} f_{sig}(x) = f_{sig}(x)(1 - f_{sig}(x))$$

On peut donc calculer la dérivée de cette fonction en un point de façon très efficace à partir de sa valeur en ce point.
De plus, la fonction sigmoïde renvoie des valeurs dans l'intervalle $[0; 1]$, ce qui permet d'interpréter la sortie du neurone comme une probabilité.

Elle est aussi liée au modèle de régression logistique et apparaît naturellement quand on considère le problème de la séparation optimale de deux classes de distributions gaussiennes avec la même matrice de covariance.

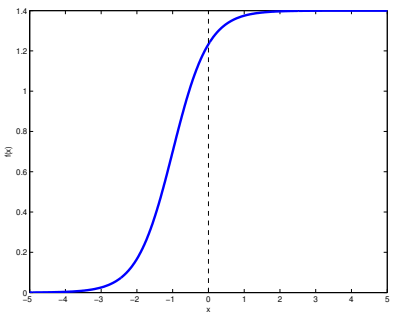


FIGURE 1.7 – Fonction de sigmoïde

1.4.5.3 Fonction tangente hyperbolique :

La fonction tangente hyperbolique, définie par :

$$\tanh(x) = \frac{\exp x - \exp -x}{\exp x + \exp -x}$$

est aussi très utilisée en pratique,car elle partage avec la fonction sigmoïde certaines caractéristiques pratiques :

- non polynomiale indéfiniment
- continûment dérivable
- calcul rapide de la dérivée par la formule : $\frac{d}{dx} \tan h(x) = 1 - (\tan h(x))^2$

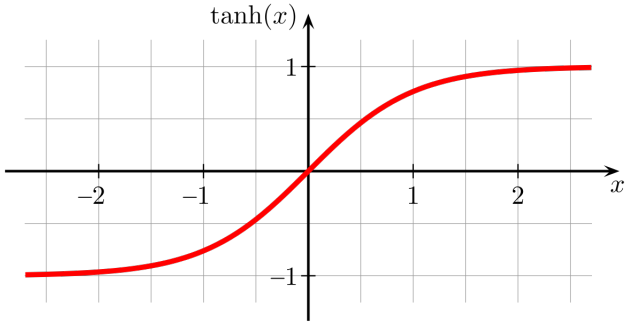


FIGURE 1.8 – La fonction tangente hyperbolique

Ces choix sont motivés par des considérations théoriques et pratiques issues de la combinaison des neurones formels en un réseau de neurones formels. Le développement de ces réseaux conduit à l'introduction de différents modèles. Mais la motivation n'était pas de mieux représenter les neurones réels, mais plutôt d'utiliser les propriétés de certaines constructions mathématiques pour obtenir des réseaux plus efficaces et plus optimisés(par exemple avec un apprentissage plus simple, ou encore utilisant moins de neurones).

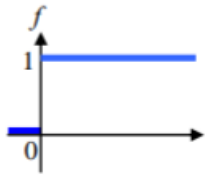
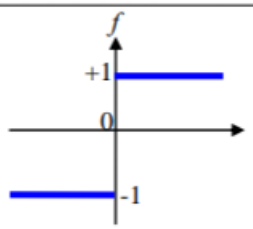
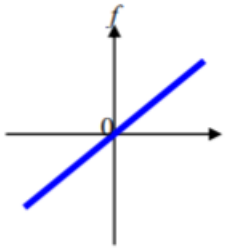
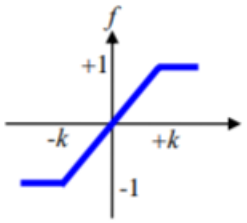
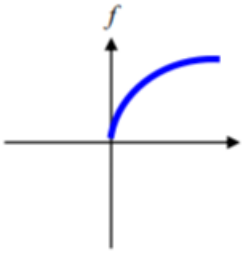
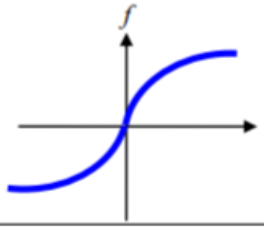
Catégorie	Type	Equation	Allure
Seuil	Binaire (Heaviside)	$f(x)=1$ si $x>0$ $f(x)=0$ si $x<=0$	
	Signe	$f(x)=1$ si $x>0$ $f(x)=-1$ si $x<=0$	
Linéaire	Identité	$f(x)=x$	
	Saturation	$f(k,x)=-1$ si $x<-1/k$ $f(k,x)=1$ si $x>=1/k$ $f(k,x)=kx$ sinon	
Sigmoïde	Positive (logistique)	$f(k,x)=\frac{1}{1+e^{kx}}$	
	Symétrique (type tanch)	$f(k,x)=\frac{2}{1+e^{-kx}}-1$	

TABLE 1.2 – Différentes fonctions d'activations utilisées dans les RNA

1.4.5.4 Propriétés importantes de la fonction d'activation :

— Les propriétés de la fonction d'activation influent en effet sur celle du neurone formel et il est donc important de bien choisir celle-ci pour obtenir un modèle utile en pratique.
— Quand les neurones sont combinés en un réseau de neurones formels, il est important par exemple que la fonction d'activation de certains d'entre eux ne soit pas un polynôme sous réserve de limiter la puissance de calcul du réseau obtenu. Variantes du neurone Propriétés

importantes de la fonction d'activation.
— Un cas caricatural de puissance limitée correspond à l'utilisation d'une fonction d'activation linéaire, comme la fonction identité : dans une telle situation le calcul global réalisé par le réseau est lui aussi linéaire et il est donc parfaitement inutile d'utiliser plusieurs neurones, un seul donnant des résultats strictement équivalents. Cependant, les fonctions de type sigmoïde sont généralement bornées. Dans certaines applications, il est important que les sorties du réseau de neurones ne soient pas limitées a priori,certains neurones du réseau doivent alors utiliser une fonction d'activation non bornée. On choisit généralement la fonction identité.
— Il est aussi utile en pratique que la fonction d'activation présente une certaine forme de régularité.
— Pour calculer le gradient de l'erreur commise par un réseau de neurones, lors de son apprentissage, il faut que la fonction d'activation soit dérivable.
— Pour calculer la matrice hessienne de l'erreur, ce qui est utile pour certaines analyses d'erreur, il faut que la fonction d'activation soit dérivable deux fois.
— Comme elles comportent généralement des points singuliers, les fonctions linéaires par morceaux sont relativement peu utilisées en pratique.

1.5 Réseaux de neurones formels

Les réseaux de neurones sont des réseaux fortement connectés de processeurs éleméntaires en parallèle ,chaque processeur élémentaire (neurone artificielle) calcul une sortie unique sur la base des informations qu'il reçoit.
Généralement ces réseaux sont optimisés par des méthodes d'apprentissage de type probabiliste, en particulier bayésien. Ils sont placés d'une part dans la famille des applications statistiques, qu'ils enrichissent avec un ensemble de paradigmes permettant de créer des classifications rapides (réseaux de Kohonen en particulier), et d'autre part dans la famille des méthodes de l'intelligence artificielle aux quelles ils fournissent un mécanisme perceptif indépendant des idées propres de l'implémenteur, et des informations d'entrée au raisonnement logique formel .

On distingue deux types d'architectures de réseaux de neurones : les réseaux de neurones « non bouclés » et les réseaux de neurones « bouclés ».

1.5.1 Réseaux de neurones non bouclés

Un réseau de neurone non bouclé (dit aussi statique) réalise une (ou plusieurs) fonctions algébriques de ses entrées, par composition des fonctions réalisées par chacun de ses neurones. représenté graphiquement par un ensemble de neurones connectés entre eux, l'information circulant des entrées vers les sorties sans « retour en arrière » ; c'est-à-dire à partir d'un neurone quelconque, en suivant les connexions, on ne peut pas revenir au neurone de départ. Le réseau représenté sur la figure 1.9.
Les réseaux de neurones non bouclés sont utilisés principalement pour effectuer des tâches d'approximation de fonction non linéaire, de classification ou de modélisation de processus.

Les réseaux de neurones non bouclés sont des objets statiques : si les entrées sont indépendantes du temps, les sorties le sont également.Ils ont une structure particulière, très fréquemment utilisée : il comprend des entrées, une couche de neurones "cachés" et des neurones de sortie. Les neurones de la couche cachée ne sont pas connectés entre eux. Cette structure est appelée Perceptron multicouche [12].

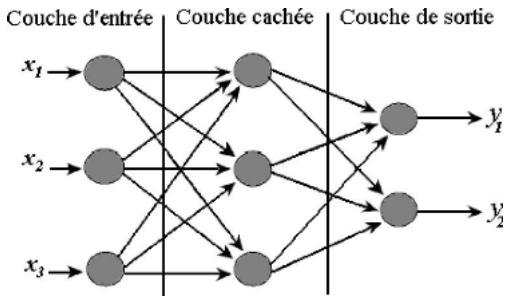


FIGURE 1.9 – Structure de réseau de neurones non bouclé

1.5.1.1 Réseau de neurone à potentiel

Un réseau de neurones potentiel est un réseau de neurones à réaction, qui est largement utilisé dans les problèmes de classification et de reconnaissance de formes.Un réseaux de neurones à potentiel est constitué de neurones interconnectés réalisant une fonction algébrique de ses entrées.Cette fontion scalaire ou vectorielle,dépend des interconnexions entre neurones. L'information circule à partir des entrées vers les sorties sans bouclage (d'ou le nom non bouclé) [40].

Les réseaux de neurones potentiel est souvent utilisé dans les problèmes de classification.
Lorsqu'une entrée est présente, la première couche calcule la distance entre le vecteur d'entrée et les vecteurs d'entrée d'apprentissage. Cela produit un vecteur où ses éléments indiquent à quel point l'entrée est proche de l'entrée d'apprentissage. La deuxième couche additionne la contribution pour chaque classe d'entrées et produit sa sortie nette comme un vecteur de probabilités. Enfin, une fonction de transfert de compétition sur la sortie de la deuxième couche sélectionne le maximum de ces probabilités, et produit un 1 (identification positive) pour cette classe et un 0 (identification négative) pour les classes non ciblées.

1.5.1.2 Réseau de neurone à Base Radiale

Les Réseaux de Fonctions à Base Radiale (Radial Basis Function - **RBF**) ont une architecture semblable à celle des **MLP**, mais ne comporte généralement que troix couches : une couche d'entrée, une couche cachée et une couche de sortie. La particularité de ce réseau réside dans sa couche cachée (deuxième couche) réagit par des fonctions de sortie φi à base radiale $(\|.\|)$. Ce réseau est capable de fournir une représentation locale de l'espace. Il y'a différentes formes des fonctions à base radiales, la plus utilisée est la fonction gaussienne [36].

$$\varphi i = exp\left(\frac{\|X - \mu i\|^2}{2w_i^2}\right) \tag{1.2}$$

L'architecture d'un réseau RBF est représentée sur la Figure 1.10.

Soit un réseau de n entrées et une sortie ; si nc est le nombre de nœuds de la couche cachée :

Soit $\|X - \mu i\|^2 =$ La distance euclidienne.
φ : Fonction de base.
μi : Centre des fonctions $\mu i \in R^{nc}$ et $1 < i < nc$
w_i : Poids des connexions entre la couche cachée et la couche de sortie.
La fonction de sortie du réseau RBF peut être exprimée par [10] :

$$f : R^n \longrightarrow R$$
$$x \longrightarrow F(x) = \sum_{i=1}^{N} w_i \varphi(X - \mu i) \tag{1.3}$$

Pour des fonctions radiales gaussiennes l'expression devien :

$$g(x,w) = \sum_{i=1}^{N} \left[w_{nc+1,i} exp\left(-\frac{\sum_{j=1}^{n}(x_j - wij)^2}{2w_i^2} \right) \right] \tag{1.4}$$

Structure générale d'un réseau de neurones RBF :

Un réseau RBF comporte trois couches de neurones, une couche d'entrée, une couche cachée directement liée à la couche d'entrée par des connexions non pondérées, et une couche de sortie liée à la couche cachée par des connexions pondérées.

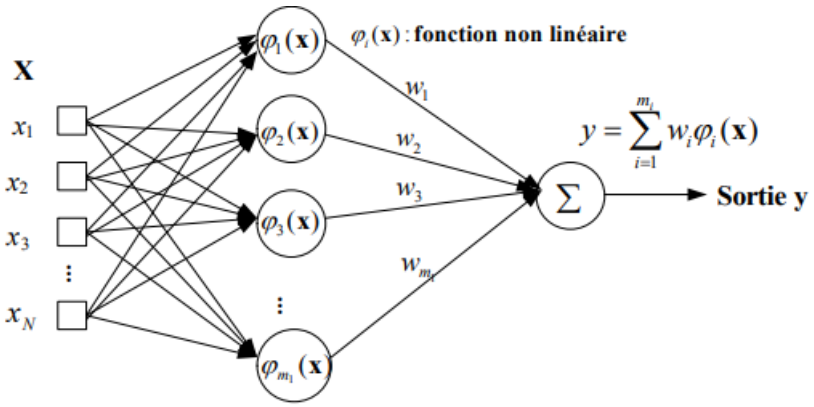


FIGURE 1.10 – Structure générale d'un réseau de neurone RBF

Un RBF est constitué uniquement de 3 couches :
La couche d'entrée : elle retransmet les inputs sans distorsion.
La couche RBF : couche cachée qui contient les neurones **RBF**.
La couche de sortie : simple couche qui contient une fonction linéaire.

Chaque neurone RBF contient une gaussienne qui est centrée sur un points de l'espace d'entrée. Pour une entrée donnée, la sortie du neurone RBF est la hauteur de la gaussienne en ce point.
La fonction gaussienne permet aux neurones de ne répondre qu'à une petite région de l'espace d'entrée, région sur laquelle la gaussienne est centrée.

La sortie du réseau est simplement une combinaison linéaire des sorties des neurones RBF multipliés par le poids de leur connexion respective.

Il y a 4 paramètres principaux à régler dans un réseau RBF :

Le nombre de neurones **RBF** (nombre de neurones dans l'unique couche cachée).
• La position des centres des gaussiennes de chacun des neurones.
• La largeur de ces gaussiennes.
• Le poids des connexions entre les neurones **RBF** et les neurones de sortie.
Tout modification d'un de ces paramètres entraîne directement un changement du comportement du réseau.

1.5.2 Réseaux de neurones bouclés

C'est Contrairement aux réseaux de neurones non bouclés, dont le graphe de connexions est acyclique, ils peuvent avoir une topologie de connexions quelconque,comprenant notamment des boucles

qui ramènent aux entrées la valeur d'une ou plusieurs sorties. Chaque boucle soit assosié un retard pour que le système soit causal ;donc c'est un système dynamique, régi par des équations différentielles ; comme l'immense majorité des applications sont réalisées par des programmes d'ordinateurs, on se place dans le cadre des systèmes à temps discret, où les équations différentielles sont remplacées par des équations aux différences [12].

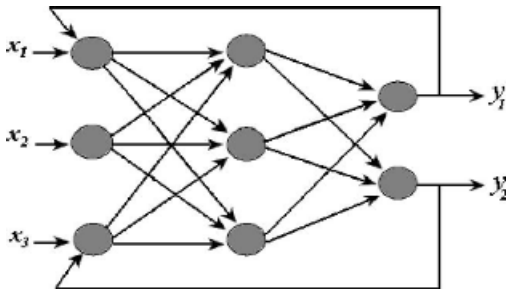


FIGURE 1.11 – Structure de réseau de neurones bouclé

1.5.2.1 Réseau de neurone d'Hopfield

Le réseau de neurone d'Hopfield est un modèle de réseau de neurones récurrents à temps discret dont la matrice des connexions est symétrique et nulle sur la diagonale et où la dynamique est asynchrone (un seul neurone est mis à jour à chaque unité de temps). Il a été popularisé par le physicien John Hopfield en 19821. Sa découverte a permis de relancer l'intérêt dans les réseaux de neurones qui s'était essoufflé durant les années 1970 à la suite d'un article de Marvin Minsky et Seymour Papert.
Un réseau de Hopfield est une mémoire adressable par son contenu : une forme mémorisée est retrouvée par une stabilisation du réseau, s'il a été stimulé par une partie adéquate de cette forme [24].

Le réseau de neurone d'Hopfield représente un réseau sans structure de couches, ni de sens de propagation, Ce réseau se rapproche le plus du fonctionnement du cerveau humain.
Ce modèle de réseau est constitué de N neurones à états binaires (-1, 1) ou (0, 1) suivant les versions) tous interconnectés Figure 1.12. L'entrée totale d'un neurone i est donc :

$$I_i = \sum_i w_{ij} V_j$$

où :

— w_{ij} est le poids de la connexion du neurone i au neurone j.

— V_j est l'état du neurone j.

L'état du réseau peut être caractérisé par un mot de N bits correspondant à l'état de chaque neurone.

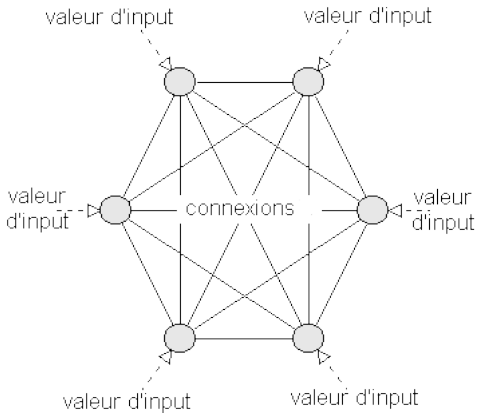


FIGURE 1.12 – Réseau de neorone de Hopfiled

1.5.2.2 Réseau de neurone Kohonen

Un réseau de Kohonen est constitué d'une couche d'entrée de N neurones connectés aux M neurones d'une couche de sortie elle-même interconnectée. Soit $W_j^T = w_{i,j}$ le vecteur des poids des N connexions reliant la couche d'entrée au neurone j de la couche de sortie. Soit X un vecteur d'entrées de N composantes.

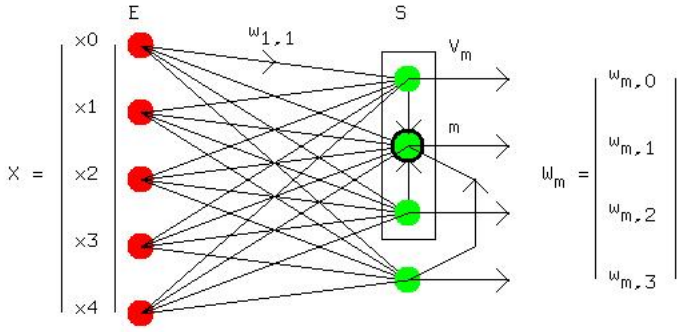


FIGURE 1.13 – Réseau de neurone Kohonen

Le procédé de recherche de regularités se déroule en deux temps : 1) Dans un premier temps, on recherche le neurone de réponse maximale :
1-1) Des poids de faibles valeurs sont affectés aléatoirement aux $N * M$ connexions i, j.
1-2) Appliquer le vecteur X en entrée.
1-3) Calculer les distances $D_j(t)$ de chaque neurone j de sortie au vecteur d´entrée X. Si on choisit la distance euclidienne, l´activation du neurone j est donnée par :

$$D_j(t) = \sum_{i=1}^{N} (xi(t) - wi, j(t))^2$$

1-4) Sélectionner le neurone le plus proche, si m est ce neurone, il vérifie, si W_j est le vecteur poids qui lui est associé.
$\|X - W_m\| = \min_j(\|X - W_j\|)$
L´activation (sortie) d´un neurone de sortie j est alors :
$yj = 1$ si $\|x(t) - w(t)_j\| = \min(\|x(t) - w_k(t)\|)$
$y_j = 0$ sinon.

Le neurone de Kohonen a une fonction d'activation qui ne prend que des valeurs positives.

La carte de Kohonen :

La carte de Kohonen est une carte auto-organisatrice en deux couches, une couche d'entrée et une autre de sortie Figure 1.14. La couche de sortie appelée aussi couche compétitive est en deux dimensions. Chaque neurone d'entrée est connecté à l'ensemble des neurones de sortie par des poids W_{ij}. La carte de Kohonen est une carte à apprentissage non supervisé, à partir d'une base d'apprentissage constitué seulement des éléments d'entrées, elle permet de regrouper ces entrées en classes [36].

Soit un réseau de Kohonen, soit E sa couche d´entrée et S sa couche de sortie. Définissons une distance d (par exemple la distance euclidienne) et une topologie T (par exemple une grille régulière orthogonale) sur S. Un voisinage Vm d´un neurone m de S est l´ensemble des neurones k de S dont la distance à m est bornée :
$V_m = k$ appartenant a S tels que $d(m,k) < d0$
(S,d,T) s´appelle **carte topologique de Kohonen**.

Le réseau fonctionne en mode dynamique lorsque des neurones sont créés ou détruits selon leur éloignement ou leur densité dans la carte. L´apprentissage consiste en une compétition des neurones de la couche de sortie qui élisent un neurone vainqueur, puis un processus de concurrence entre neurones élus affine la recherche.

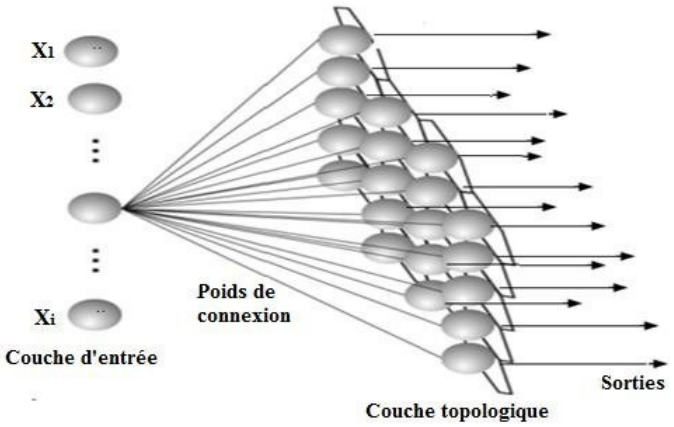


FIGURE 1.14 – L'architecture d'une carte de Kohonen

1.5.3 Autres types de réseaux de neurones

Les réseaux de neurones sont divisés en types en fonction du nombre de couches cachées qu'ils contiennent ou de la profondeur du réseau. Chaque type a ses propres niveaux de complexité et cas d'utilisation. Peu de types de réseaux neuronaux sont les réseaux de neurones de type Feed-forward (Perceptron simple, Perceptron multicouche), le réseau neuronal récurrent et le réseau neuronal convolutif.

1.5.3.1 Perceptrons simple (monocouche)

Parmi les plus simples des réseaux de neurones, on cite le réseau monocouche appelé perceptron simple. Historiquement, le perceptron est parmi les premiers réseaux efficaces qui a été proposé et étudié en détail. Il est composé uniquement d'une couche d'entrée et d'une couche de sortie. La couche d'entrée comporte autant de neurones que le vecteur de caractéristiques (primitives) a de

composantes. Les connexions entre ces deux couches sont modifiables et bidirectionnelles [3]. C'est la couche de sortie qui remplit la tâche de classification et les unités de la couche de sortie réalisent une fonction à seuil. Nous aborderons en détails ce type de réseau au chapitre 2

1.5.3.2 Perceptrons Multi-Couches

Les réseaux de neurones de type Perceptrons Multi-Couches (Multi Layer Perceptron - MLP) sont des réseaux à propagation avant, composés d'une ou plusieurs couches cachées et d'une couche de sortie. Chaque couche du réseau est composée de neurones artificiels. La première couche cachée reçoit l'information provenant des entrées. L'information est traitée et transmise vers les couches suivantes jusqu'à la dernière. Les MLP sont connus comme étant des approximateurs universels et sont très utilisés dans des probl'emes de régression non linéaire [48]. Ils font l'objets d'une étude détaillée dans le chapitre 3.

1.5.3.3 Réseaux de neurones récurrents-Recurrent Neural Network (RNN)

Les réseaux récurrents sont appelés également réseaux bouclés ou dynamiques. Ce sont des réseaux dans lesquels beaucoup plus complexes et les plus largement utilisés. Les données circulent dans plusieurs directions dans ce réseau. Ils stockent les données de sortie des nœuds de traitement et apprennent à améliorer leur fonctionnement.
Les réseaux de neurones récurents trouvent des applications dans des domaines qui nécessitent un apprentissage supervisé, comme la vision par ordinateur.
Les réseaux de neurones à réaction directe sont les plus couramment utilisés dans les systèmes de reconnaissance d'objets et de reconnaissance vocale et sont souvent utilisés pour identifier des systèmes dynamiques non linéaires. Le schéma suivant montre le détail d'une couche récurrente figure 1.15. Les x_i^t et les y_j^t désignent respectivement les entrées et les sorties de la couche à l'instant t.

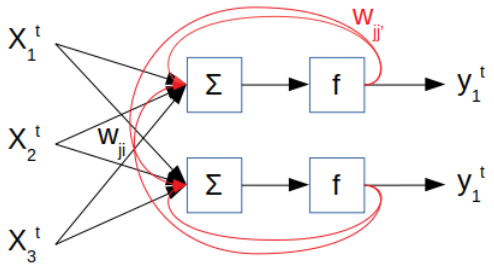


FIGURE 1.15 – Exemple d'une couche RNN simple à trois entrée et deux sorties. Les connexions récurrentes sont notées en rouge.

Depuis le début des années 1980, les réseaux de neurones récurrents ont montré des potentialités intéressantes pour la résolution des problèmes d'optimisation. Ils présentent deux atouts majeurs dont le premier réside dans le fait que les réseaux de neurones artificiels solutionnent souvent très bien des problèmes d'optimisation et le second atout vient de ce que ces réseaux peuvent donner lieu à des réalisations bénéficiant des avantages du parallélisme. Les réseaux de neurones récurrents sont particulièrement adaptés aux problèmes qui requièrent des temps de réponse extrêmement brefs, et qui justifient éventuellement leur utilisation [1].

1.5.3.4 Réseaux de neurones convolutifs-Convolution Neural Network (CNN)

Les réseaux de neurones convolutifs appelé aussi CNN «Convolutional Neural Network» sont ceux qui sont populaires aujourd'hui en raison de leur spécialité dans la capacité à effectuer la recon-

naissance faciale. Ils permettent d'encoder des attributs dans l'entrée, en supposant qu'il s'agit d'une image. Se représente en général sous la forme suivante figure 1.16 :

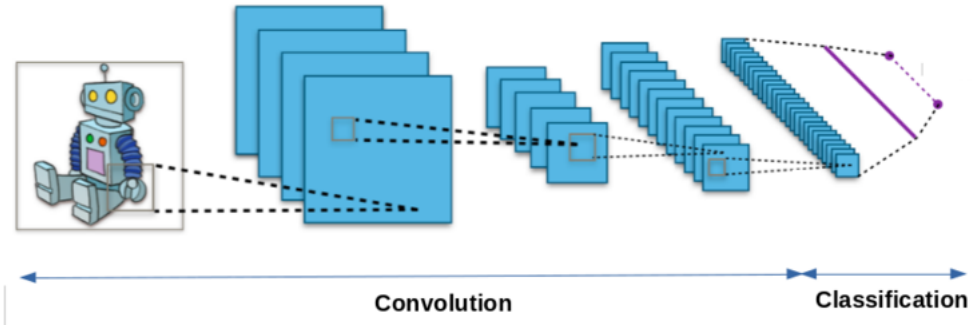


FIGURE 1.16 – Architecture d'un réseau de neurones convolutif.

On distingue deux parties, une première partie que l'on appelle la partie convolutive du modèle et la seconde partie, que l'on va appeler la partie classification du modèle qui correspond à un modèle MLP (Multi Layers Perceptron) figure 1.17.

Une architecture de réseau de neurones convolutifs est formée par un empilement de couches de traitement [43] :

1. la couche de convolution (CONV) qui traite les données d'un champ récepteur.
2. la couche de pooling (POOL), qui permet de compresser l'information en réduisant la taille de l'image intermédiaire (souvent par sous-échantillonnage) .
3. la couche de correction (ReLU), souvent appelée par abus « ReLU » en référence à la fonction d'activation (Unité de rectification linéaire)
4. la couche « entièrement connectée » (FC), qui est une couche de type perceptron
5. la couche de perte (LOSS).
   Dans un réseau de neurones convolutionnels, une couche est composée de trois étapes : la convolution, l'application d'une fonction d'activation et enfin le pooling. Le résultat de ces trois étapes est appelé une feature map [42].

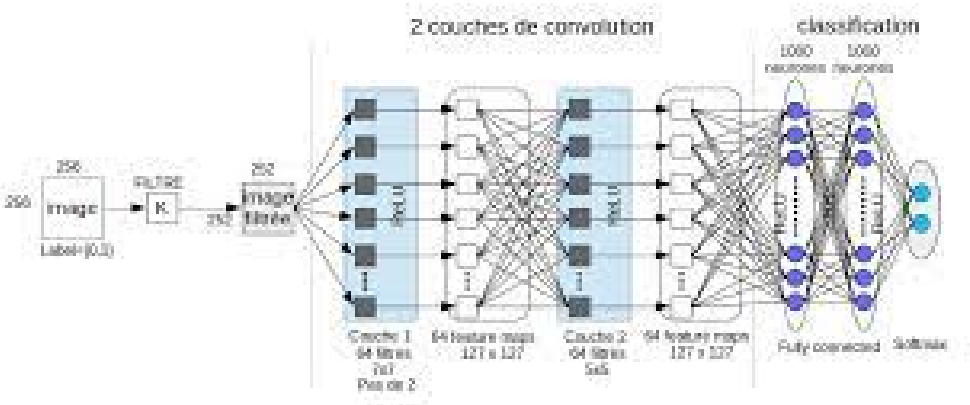


FIGURE 1.17 – Schéma du réseau de neurones convolutionnels

Plusieurs travaux ont été élaboré par des auteurs qui ont pour objectif de comparer les différents types des réseaux de neurones[13, 39]. En se basant sur ces travaux, le tableau 1.3 résume les avantages est les inconvénients de chaque architecture des réseaux de neurones les plus utilisés.

RN	Description	Apprentissage	Avantages	Inconvénients
Perceptron simple	• Un seul neurone	**Supervisé**	- Architecture simple	N'accepte pas les données bruitées et la classification non linéaire
PMC	• Plusieurs neurones organisés en couches • Sorties des neurones : sigmoïde	**Supervisé**	- Accepte les données bruitées et la classification non-linéaire - Représentation globale de l'espace - Architecture simple	- Classe les éléments qui n'appartiennent à aucune classe à la classe la plus proche - Le nombre de couches caché et des neurones dans les couches caches est indéfinit
RVFLNN	• Réseau en trois couches • Couche d'entrée reliée à la couche de sortie	**Supervisé** (seulement les poids reliant la couche d'entrée avec celle de la sortie)	- Accepte les données bruitées et la classification non-linéaire - Représentation globale de l'espace - Des entrées optimisées	-Architecture plus complexe que le PMC Moins de précision que le PMC
RBF	• Plusieurs neurones organisés en 3 couches • Couche cachée : gaussienne	**Non supervisé** (μ, σ) + **Supervisé**	- Accepte les données bruitées et la classification non-linéaire - Capable de dire « je ne sais pas » - Représentation locale de l'espace - Une grande précision. - Apprentissage rapide.	- Apprentissage complexe - Nécessite une grande capacité de calcul
Carte de Kohonen	• Couche de sortie sous forme d'une carte (2D)	**Non supervisé**	- Classifier les données sans connaissance préalable de procédé. - regroupement automatique des données	- Parfois les résultats ne correspondent pas à celle souhaitée

TABLE 1.3 – Les avantages et les inconvenients de quelques architectures des réseaux de neurones

1.6 Apprentissage dans les réseaux de neurones

L'apprentissage est une phase du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré.
c'est à dire l'apprentissage consiste à modifier le poids des connections entre les neurones par des règles de modification.

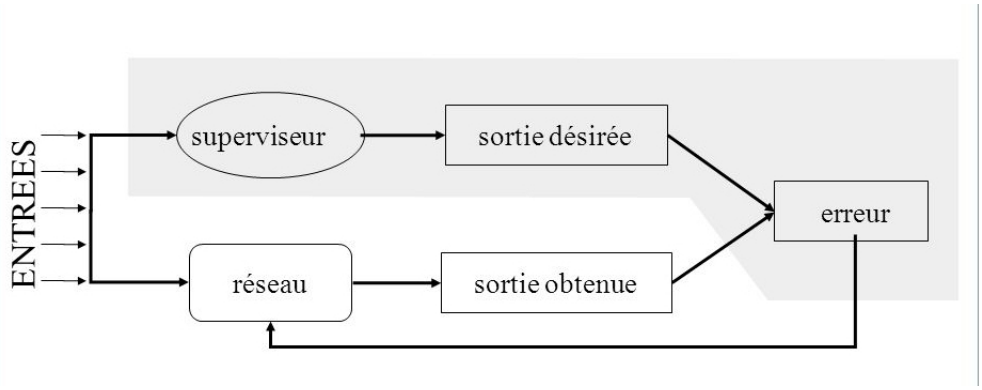
1.6.1 Types d'apprentissage :

On distinque trois grandes classes d'algorithmes d'apprentissage

1.6.1.1 Apprentissage supervisé

Ce mode est le plus courant,car la majorité des apprentissages automatiques utilisent un apprentissage supervisé .

Dans ce type d'apprentissage,le réseau s'adapte par comparaison entre le résultat qu'il a calculé en fonction des entrée fournies,et la sortie désiré.le réseau va se modifier jusqu'à ce qu'il trouve la bonne sortie.
Schématiquement, il consiste à évaluer les poids synaptiques minimisant l'erreur sur une base d'apprentissage, cette base comprend un ensemble d'observations pour lesquelles on connaît à la fois les entrées et les sorties souhaitées.
L'apprentissage est dit supervisé, car il est nécessaire de connaître la sortie attendue pour chaque jeu d'entrées, le réseau de neurones va ajuster ses paramètres afin de minimiser l'erreur entre la sortie désirée et sa sortie réelle [33].



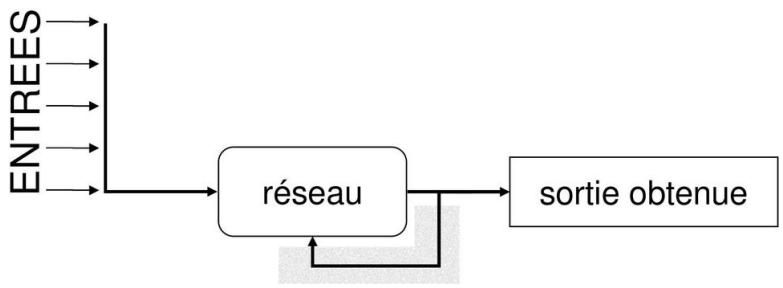
1.6.1.2 Apprentissage semi- supervisé

L'apprentissage semi- supervisé est une classe de techniques d'apprentissage automatique qui utilise un ensemble de données étiquetées et non étiquetées. Il se situe ainsi entre l'apprentissage supervisé qui n'utilise que des données étiquetées et l'apprentissage non supervisé qui n'utilise que des données non étiquetées. Il a été démontré que l'utilisation de données non étiquetées, en combinaison avec des données étiquetées, permet d'améliorer significativement la qualité de l'apprentissage.
Cet apprentissage est identique au précédent dans la mesure où il se base sur la présence d'un concepteur, cependant la valeur exacte de la sortie n'est pas disponible, en général la seule information disponible est un signal d'échec ou de succès [33].

Un exemple d'apprentissage semi-supervisé est le coapprentissage, dans lequel deux classifieurs apprennent un ensemble de données, mais en utilisant chacun un ensemble de caractéristiques différentes, idéalement indépendantes. Si les données sont des individus à classer en hommes et femmes, l'un pourra utiliser la taille et l'autre la pilosité par exemple.

1.6.1.3 Apprentissage non supervisé

Dans ce type d'apprentissage,l'apprentissage est basé sur des probabilités.Le réseau va se modifier en fonction des régularités statistiques de l'entrée en optimisant une valeur de qualités aux catégories reconnues.
L'apprentissage non supervisé consiste à ne disposer que de données d'entrée (X) et pas de variables de sortie correspondantes. L'objectif de cet apprentissage est de modéliser la structure ou la distribution sous-jacente dans les données afin d'en apprendre davantage sur les données.
On l'appelle apprentissage non supervisé car, contrairement à l'apprentissage supervisé ci-dessus, il n'y a pas de réponse correcte ni d'enseignant. Les algorithmes sont laissés à leurs propres mécanismes pour découvrir et présenter la structure intéressante des données.



La principale différence entre les deux types réside dans le fait que l'apprentissage supervisé se fait sur la base d'une vérité. En d'autres termes, nous avons une connaissance préalable de ce que devraient être les valeurs de sortie de nos échantillons. Par conséquent, l'objectif de l'apprentissage supervisé est d'apprendre une fonction qui, à partir d'un échantillon de données et des résultats souhaités, se rapproche le mieux de la relation entre entrée et sortie observable dans les données. En revanche, l'apprentissage non supervisé n'a pas de résultats étiquetés. Son objectif est donc de déduire la structure naturelle présente dans un ensemble de points de données. Dans cet article, nous allons voir toutes les différences entre l'apprentissage supervisé et non supervisé.

1.7 L'utilité des réseaux de neurones artificiels

— Les réseaux de neurones, en tant que systèmes capables d'apprendre, mettent en oeuvre le principe de l'induction, c'est-à-dire l'apprentissage par l'expérience.
— Par confrontation avec des situations ponctuelles, ils infèrent un système de décision intégré dont le caractère générique est fonction du nombre de cas d'apprentissages rencontrés et de leur complexité par rapport à la complexité du problème à résoudre.
— Par opposition, les systèmes symboliques capables d'apprentissage, s'ils implémentent également l'induction, le font sur base de la logique algorithmique, par complexification d'un ensemble de règles déductives (Prolog par exemple).
— Grâce à leur capacité de classification et de généralisation, les réseaux de neurones sont généralement utilisés dans des problèmes de nature statistique, tels que la classification automatique de codes postaux ou la prise de décision concernant un achat boursier en fonction de l'évolution des cours.
— Autre exemple, une banque peut créer un jeu de données sur les clients qui ont effectué un emprunt constitué : de leur revenu, de leur âge, du nombre d'enfants à charge... et s'il s'agit d'un bon client. Si ce jeu de données est suffisamment grand, il peut être utilisé pour l'entraînement d'un réseau de neurones.
— La banque pourra alors présenter les caractéristiques d'un potentiel nouveau client, et le réseau répondra s'il sera bon client ou non, en généralisant à partir des cas qu'il connaît.
— Si le réseau de neurones fonctionne avec des nombres réels, la réponse traduit une probabilité de certitude. Par exemple : 1 pour « sûr qu'il sera un bon client », -1 pour « sûr qu'il sera mauvais client », 0 pour « aucune idée », 0,9 pour « presque sûr qu'il sera bon client ».
— Le réseau de neurones ne fournit pas toujours de règle exploitable par un humain. Le réseau reste souvent une boîte noire qui fournit une réponse quand on lui présente une donnée, mais le réseau ne fournit pas de justification facile à interpréter.

1.7.1 Quelques exemples réelles d'utilisation des reseaux de neurones

les réseaux de neurones sont réellement utilisés,par exemple :

— Pour la classification d'espèces animales par espèce étant donnée une analyse ADN. reconnaissance de motif ; par exemple pour la reconnaissance optique de caractères (OCR), et notamment par les banques pour vérifier le montant des chèques, par La Poste pour trier le courrier en fonction du code postal, etc. ; ou bien encore pour le déplacement automatisé de robots mobiles autonomes.
— Approximation d'une fonction inconnue.
— Modélisation accélérée d'une fonction connue mais très complexe à calculer avec exactitude ; par exemple certaines fonctions d'inversions utilisées pour décoder les signaux de télédétection émis par les satellites et les transformer en données sur la surface de la mer.
— Estimations boursières :
— Apprentissage de la valeur d'une entreprise en fonction des indices disponibles : bénéfices, endettements à long et court terme, chiffre d'affaires, carnet de commandes, indications techniques de conjoncture. Ce type d'application ne pose pas en général de problème.
— Tentatives de prédiction sur la périodicité des cours boursiers. Ce type de prédiction est très contesté pour deux raisons, l'une étant qu'il n'est pas évident que le cours d'une action ait de façon tout à fait convaincante un caractère périodique (le marché anticipe en effet largement les hausses comme les baisses prévisibles, ce qui applique à toute périodicité éventuelle une variation de période tendant à la rendre difficilement fiable), et l'autre que l'avenir prévisible d'une entreprise détermine au moins aussi fortement le cours de son action, si ce n'est plus encore que peut le faire son passé ; les cas de Pan Am, Manufrance ou IBM permettent de s'en convaincre.
— En météorologie, pour la classification de conditions atmosphériques et la prévision statistique du temps.

1.8 Domaines d'application des réseaux de neurones

Le réseau de neurones artificiels existe depuis 1943, lors de sa conception initiale, mais n'est apparu que récemment sous l'intelligence artificielle en raison des applications qui le rendent plus préférable. Ceux-ci inclus :

Traitement d'image,Traitement de la langue, traduction et Détection d'itinéraire Reconnaissance de la parole Prévision ...

• **Traitement d'images :** reconnaissance de caractères et de signatures, compression d'images, reconnaissance de forme, cryptage, classification, etc.

• **Traitement du signal :** filtrage, classification, identification de source, traitement de la parole. . . etc. Le traitement du signal est la discipline qui développe et étudie les techniques de traitement, d'analyse et d'interprétation des signaux. Parmi les types d'opérations possibles sur ces signaux, on peut dénoter le contrôle, le filtrage, la compression et la transmission de données, la réduction du bruit, la déconvolution, la prédiction, l'identification, la classification, etc.
Bien que cette discipline trouve son origine dans les sciences de l'ingénieur (particulièrement l'électronique et l'automatique), elle fait aujourd'hui largement appel à de nombreux domaines des mathématiques, comme la théorie du signal, les processus stochastiques, les espaces vectoriels et l'algèbre linéaire et des mathématiques appliquées, notamment la théorie de l'information, l'optimisation ou encore l'analyse numérique.

• **Contrôle :** commande de processus, diagnostic, contrôle qualité, asservissement des robots, systèmes de guidage automatique des automobiles et des avions. . . etc.

• **Défense :** guidage des missiles, suivi de cible, reconnaissance du visage, radar, sonar, lidar, compression de données, suppression du bruit. . . etc.

• **Optimisation :** planification, allocation de ressource, gestion et finances, etc.

• **Simulation :** simulation du vol, simulation de boîte noire, prévision météorologique, recopie de modèle. . . etc.
•**Eléctronique :** prédiction de la séquence d'un code, vision machine, synthétiseur vocal,. . .
•**Finance :** Prévision du coût de la vie.
•**Secteur médical :** Analyse EEG et ECG.
•**Télécommunications :** Compression de données . . .

Les réseaux de neurones artificiels sont actuellement utilisés pour résoudre de nombreux problèmes complexes et la demande augmente avec le temps. Le grand nombre d'applications, de la reconnaissance faciale à la prise de décision, est géré par des réseaux de neurones.

Le tableau 2.2, représente la Correspondance entre chaque domaine d'application et le type de réseaux de neurones artificiels les plus appropriés.

Domaine d'application	Type de RNA
Optimisation	Hopefield
Approximation de fonctions	MLP, RBF
Trraitement d'images	Hopefield
Classification	MLP, Kohonen, RBF

TABLE 1.4 – Correspondance type de RNA-Domaine d'application.

1.9 Conclusion

Dans ce chapitre, on a présenté une étude sommaire sur les réseaux de neurones artificiels l'un des applications d'intelligence artificielle, qui ont reçus une grande attention des chercheurs en raison de leur flexibilité athlétique. Un aperçu général est donné sur la notion des réseaux de neurones artificiels, en passant par la définition, la structure, le fonctionnement, les types d'apprentissage et les différents types des réseaux de neurones, on a vu aussi le domaine d'applications de ces réseaux et leurs grande utilité.

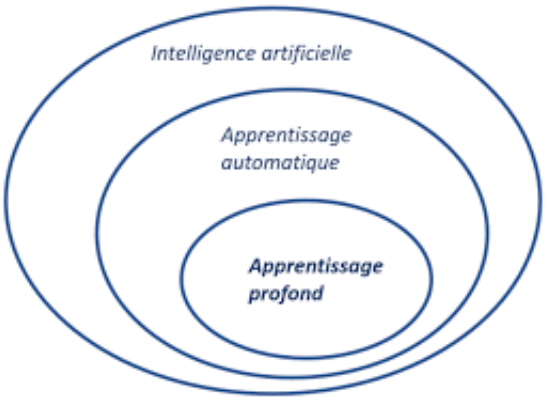
Chapitre

2 Apprentissage supervisé

Sommaire

2.1 Introduction

L'apprentissage profond ou apprentissage en profondeur est un ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires,il fait partie d'une famille de méthodes d'apprentissage automatique fondés sur l'apprentissage de modèles de données c'est à dire développer un modèle en se servant d'un algorithme pour minimiser les erreurs entre le modèle et nos données,parmi ces models le plus connus et développé on a le Percepton .
Le Percepton est Le premier réseau de neurones artificielle inventé en 1958 par Rosenblatt ,il est l'un des algorithmes d'inteligence artificielle les plus sophistiqué au monde , il est inspiré de fonctionnement des neurones biologique,C'est un réseau simple, puisque il ne se compose que d'une couche d'entrée et d'une couche de sortie, il est calqué, à la base, sur le système visuel et de ce fait a été conçu dans un but premier de reconnaissance des formes ,cet algorithme a la capacité de réaliser n'importe tâche,conduire une voiture ,jouer des échecs ,entretien au conversation,reconnaitre et classer des images,résoudre des opérations logiques simples (telle "ET" ou "OU") ...
Le percepton dispose d'un algorithme d'apprentissage qui permet de trouver les valeurs de poids synaptiques W afin d'obtenir les sorties Y qui nous convient,Il suit généralement un apprentissage supervisé selon la règle de correction de l'erreur (ou selon la règle de Hebb). pour développer cet algorithme Rosenblatt a s'inspiré de théorie de Hebb,aprés ce développement tout le monde s'est mis à imaginer et à rêver qu'il était possible de construire des machines capable de lire et de parler et meme d'avoir une conscience le truc de foux,mais ces attentes n'ont pas duré longtemps car aprés des années ils ont découvert que ces promesses ne pourait etre tenue en partie car le perceptron est un modèle simple et linéare donc sa principale limite est qu'il ne peut résoudre que des problèmes linéairement séparables,et cela est considéré comme un énorme défaut dans le perceptron.



Dans ce chapitre,en savoir plus sur l'algorithme de perceptron la mère des réseaux de neurones et de l'apprentissage profond et les règle d'apprentissage qu'il suit.

2.2 Perceptron simple

Le perceptron simple est un modèle de prédiction (supervisé) linéaire ,et c'est le premier modèle informatique opérationnel de réseau de neurones.

Le perceptron est un algorithme d'apprentissage supervisé de classifieurs binaires (c'est-à-dire séparant deux classes). Ce type de réseau neuronal ne contient aucun cycle (il s'agit d'un réseau de neurones à propagation avant).

Il s'agit d'un neurone formel muni d'une règle d'apprentissage qui permet de déterminer automatiquement les poids synaptiques de manière à séparer un problème d'apprentissage supervisé. Si le problème est linéairement séparable, un théorème assure que la règle du perceptron permet de trouver une séparatrice entre les deux classes.
Dans sa version simplifiée, le perceptron est mono-couche et n'a qu'une seule sortie (booléenne) à laquelle toutes les entrées (booléennes) sont connectées.

2.2.1 Architecture d'un perceptron simple

Le perceptron est formée d'une première couche d'unités (ou neurones) qui permettent de « lire » les données : chaque unité correspond à une des variables d'entrée. On peut rajouter une unité de biais qui est toujours activée (elle transmet 1 quelles que soient les données). Ces unités sont reliées à une seule et unique unité de sortie, qui reçoit la somme des unités qui lui sont reliées, pondérée par des poids de connexion. Pour p variables $x_1, x_2, \ldots, x_p$, la sortie reçoit donc :

$$w_0 + \sum_{j=1}^{p} w_j x_j$$

L'unité de sortie applique alors une fonction d'activation a à cette sortie.
Un perceptron prédit donc grâce à une fonction de décision f définie par :

$$f(x) = a(\sum_{j=1}^{p} w_j x_j + w_0)$$

Cette fonction a une forme explicite, il s'agit bien d'un modèle paramétrique.

Un réseau de neurones monocouche est caractérisé par :

— p informations en entrée
— q neurones
— chacun des q neurones est connecté aux p informations d'entrée

A noter :

— $X = (x_i)_{1 \leq i \leq p}$: les p informations d'entrée.

— w_{ji}, $1 \leq i \leq p$ et $1 \leq j \leq q$: les poids de connexion

— y_j : la sortie du j-ème neurone.

— a_j : la donnée d'entrée (somme pondérée) du j-ème neurone.

— seuil : seuil d'activation du neurone
On parle aussi du coefficient de biais w_0, pour ajuster la sensibilité du neurone
W_0=seuil ils jouent le même rôle, le seuil est fixe le biais se reactualise avec les poids de connexion
La sortie o est calculée par la formule :

$$o = f(x) = \begin{cases} 1 \quad si \quad \sum_{i=1}^{n} w_i x_i > \theta \\ 0, \quad sinon \end{cases}$$

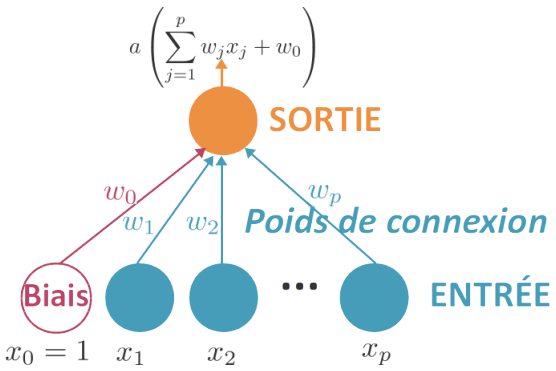


FIGURE 2.1 – Schéma d'un réseau de neurone Perceptron

Perceptron linéaire à seuil

- n entrées $x_1, \ldots, x_n$.
- n coefficients synaptiques $w_1, \ldots, w_n$.
- une sortie o.
- un seuil θ.

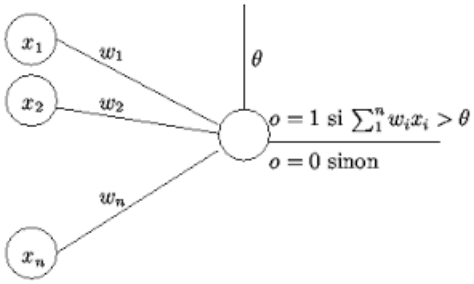


FIGURE 2.2 – Perceptron linéaire à seuil

Le perceptron avec entrée supplémentaire

Pour simplifier les notations et certaines preuves, nous allons remplacer le seuil par une entrée supplémentaire qui prend toujours comme valeur d'entrée la valeur =1. À cette entrée est associée un coefficient synaptique [41]. Le modèle correspondant est décrit dans la figure 2.3.
On peut décomposer le calcul de la sortie en un premier calcul de la quantité appelée ou l' suivi d'une application d'une sur cette entrée totale. La fonction d'activation est la fonction de Heaviside définie par :

$$f(x) = \begin{cases} 1 \quad si \quad \text{x>0} \\ 0 \quad , \quad sinon \end{cases}$$

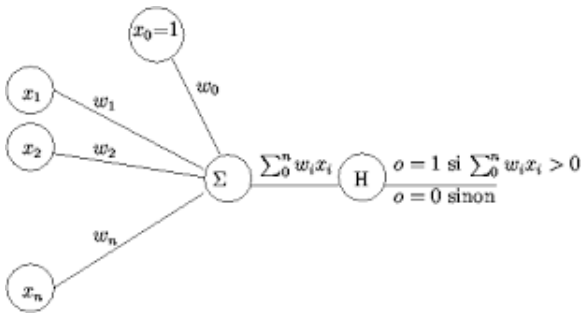


FIGURE 2.3 – Perceptron avec entrée supplémentaire

Le biais :
On a donc l'équation suivante :

$$\forall 1 \leq j \leq q, y_j = g(a_j) = g(\sum_{i=0}^{p}(w_{ji}) \times x_i)$$

Généralement, le neurone est activé si l'activation $a_j \geq 0$ on atteint le seuil de la fonction d'activation lorsque la somme pondérée des informations d'entrée vaut le coefficient de biais.

$$a_j = \sum_{i=1}^{p}(w_{ji}) \times x_i - w_{j0} \geq 0 \longrightarrow \sum_{i=1}^{p}(w_{ji}) \times x_i \geq w_{j0}$$

2.2.2 Interprétation géométrique

Theorem 1

Soit S un ensemble d'exemples dans $R^n \times \{0, 1\}$.
- Soit $S_0 = \{s \in R^n | (s, 0) \in S\}$.
- Soit $S_1 = \{s \in R^n | (s, 1) \in S\}$.
S est dit linéairement séparable s'il existe un hyperplan H de R^n tel que les ensemble S_0 et S_1 soient situés de part et d'autre de cet hyperplan.

Données $(x_1, ..., x_n)$
$\rightarrow$ E un espace de dimension d
Points vérifiants $b + \langle \overrightarrow{w}, \overrightarrow{x} \rangle = 0$
$\rightarrow$ hyperplan défini par b et $\overrightarrow{w}$

points vérifiants $b + \langle \overrightarrow{w}, \overrightarrow{x} \rangle > 0$
$\rightarrow$ points d'un coté de l'hyperplan
points vérifiants $b + \langle \overrightarrow{w}, \overrightarrow{x} \rangle < 0$
$\rightarrow$ points de l'autre coté de l'hyperplan

un perceptron divise l'espace des données en deux demi-espaces $\rightarrow$ situés de part et d'autre de l'hyperplan

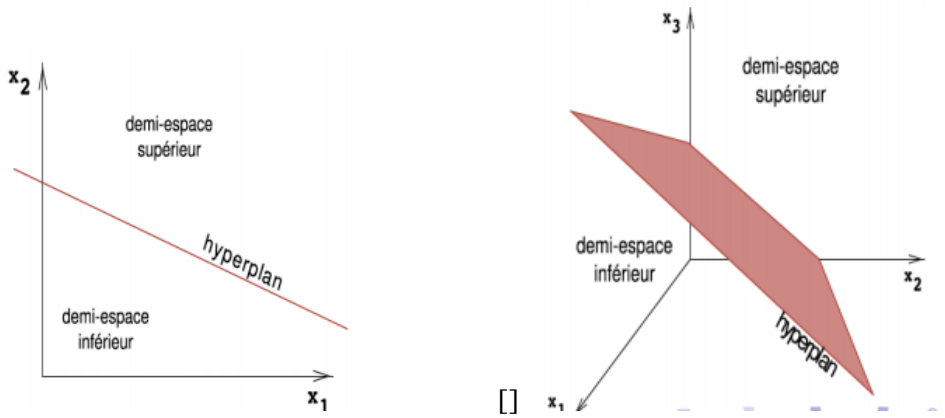
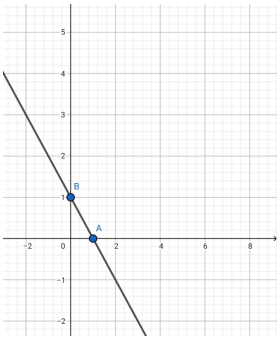


FIGURE 2.4 – Interprétation géométrique d'un séparation linéaire

Exemple :
$X = R^2$ classifieur linéare définit par $W = (1, 2)$, et $b = -1$

$$f(x_1, x_2) = \begin{cases} 1 & si \quad x_1 + 2x_2 + 3 >= 0 \\ -1 & sinon \end{cases}$$

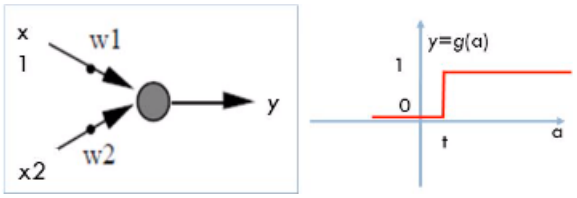
par exemple $f(0, 0) = -1$ et $f(1, 1) = 1$
hyprplan d'equation $x_1 + 2x_2 - 1 = 0$



Quel type de problème peut résoudre le Perceptron ?

En mathématique l'équation d'un plan dans un espace de dimension N, ce que l'on appelle hyper-plan.
Un plan sépare l'espace géométrique en deux parties, nous dirons que l'on a un classificateur linéaire. Le perceptron va donc être capable de classer des entrées et décider si elles corresponde à un côté du plan ou bien à un autre.
C'est la fonction de transfert qui distribue la réponse S d'un côté ou l'autre du plan.

Le neurone réalise une simple somme pondérée de ses entrées, compare une valeur de seuil t, et fourni une réponse binaire en sortie y.
On peut interpréter sa décision comme classe 1 si la valeur de y est $+1$ et classe 2 si la valeur de y est 0.

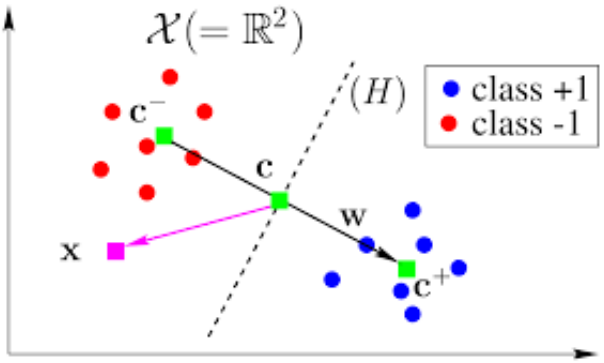


2.2.3 Problème de classification linéaire

Theorem 2

Soit deux sous ensembles S+ et S- de $\mathbf{R^d}$. Ces deux ensembles sont séparables si et seulement si il existe $W = (W_0; ...; W_d) \in \mathbf{R^{d+1}}$ tel que :
$\forall$ x $\in$ S+ :$w_0 + \sum_{i=1}^{d} w_i x_i > 0$
$\forall$ x $\in$ S- :$w_0 + \sum_{i=1}^{d} w_i x_i < 0$
Un perceptron F_w avec la fonction signe pour fonction d'activation peut donc séparer ces deux ensembles :
$F_w(x)$ = 1 si x $\in$ S+
$F_w(x)$ = -1 si x $\in$ S-

Si les classes sont linéarement séparable ,le perceptron est garanti de converger à une solution avec un erreur nulle sur l'ensemble d'entrâinement pour tout α.
Donc le perceptron cherche un séparateur linéare entre les deux classes.
La frontière de décision(H) d'un classifieur est la surface qui sépare les deux régions classifiées dans les deux classes différentes.



2.2.4 Algorithme Perceptron (Rosenblatt)

C'est le plus ancien des algorithmes d'apprentissage proposé par Rosenblatt en 1958,il peut etre utilisé comme une méthode de gradient exact(algorithme du cout total :les modifications des paramètres du réseau sont opérées aprés présentation de tout les exemples de la base d'apprentissage)ou bien comme méthode de gradient stochastique(algorithme du cout partiel :dans ce cas ,on modifie les paramètres du réseau aprés présentation de chaque exemple).
L'algorithme d'apprentissage du Perceptron doit adapter la valeur des paramètres de façon que $H_W(x)$ soit la bonne réponse sur les données d'entrâinement.
Cet algoritheme est semblable à celui utilisé pour la loi de Hebb. Les différences se situent au niveau de la modification des poids.

Soit le perceptron, avec le signal d'apprentissage D : Algorithme :

1. choisir des valeurs aléatoires pour les poids et le biais.
2. pour chaque paire $(x_t, y_t) \in D$
   a-calculer $H_W(x_t) = seuil(x, x_t)$
   b-si $y_t \neq H_W(x_t)$
   •$W_i \leftarrow W_i + \alpha(y_t - H_W(x_t), \forall i$

3. retourner à 1 jusqu'à l'atteinte d'un critère d'aret.

la mise à jour des poids est appelée la règle d'apprentissage.
le multiplicateur α : les taux d'apprentissage.

La règle d'apprentissage du perceptron (Rosenblatt 1958) est caractérisé par :
- la règle est applicable aux neurones de type signe.
- le signal d'apprentissage est la différence entre l'actuelle et la réponse désirée.
- la nécessité d'un signal désiré implique un apprentissage supervisé.

• **exemple de l'algorithme perceptron :**

simulation avec biais $\alpha = 0.1$

x_t	y_t
[2,0]	1
[0,3]	0
[3,0]	0
[1,1]	0

•Initialisation : W $\leftarrow [0,0], b \leftarrow 0.5$

paire (x_1, y_1) :
•$H(x_1) = Heavside(Wx_1 + b = Heavside(0.5) = 1$
•puisque $H(x_1) = y_1$,on fait pas de met à jour W et b.

paire (x_2, y_2) :w$\leftarrow [0,0], b \leftarrow 0.5$.
H(x_2) $= Heavside(Wx_2 + b = Heavside(0.5) = 1$
•puisque $H(x_1) \neq y_1$,on met à jour W et b.
$\gg$W $\leftarrow W + \alpha(y_2 - H(x_2)x_2 = [0,0] + 0.1(0-1)[0,3] = [0,-0.3]$
$\gg$b $\leftarrow b + \alpha(y_2 - H(x_2)) = 0.5 + 0.1(0-1) = 0.4$

paire (x_3, y_3) :w$\leftarrow [0,-0.3], b \leftarrow 0.4$.
H(x_3) $= Heavside(Wx_3 + b = Heavside(0.4) = 1$
•puisque $H(x_3) \neq y_3$,on met à jour W et b.
$\gg$W $\leftarrow W + \alpha(y_3 - H(x_3)x_3 = [0,-0.3] + 0.1(0-1)[3,0] = [-0.3,-0.3]$
$\gg$b $\leftarrow b + \alpha(y_3 - H(x_3)) = 0.4 + 0.1(0-1) = 0.3$

paire (x_4, y_4) :w$\leftarrow [-0.3,-0.3], b \leftarrow 0.3$.
H(x_4) $= Heavside(Wx_4 + b = Heavside(-0.3) = 0$
•puisque $H(x_4) \neq y_4$,on met à jour W et b.
$\gg$W $\leftarrow W + \alpha(y_4 - H(x_4)x_4 = [-0.3,-0.3] + 0.1(0-1)[1,1] = [-0.2,-0.2]$
$\gg$b $\leftarrow b + \alpha(y_4 - H(x_4)) = 0.3 + 0.1(0-1) = 0.4$

2.3 Règles d'apprentissages

L'apprentissage consiste à modifier le poid des connexions entre les neurones,il existe plusieurs règles de modification :
•Loi de Hebb
•Règle de Delta

2.3.1 Règle de Hebb

Soit un Réseau de neurones à :

— n entrées $e_1, \ldots, e_n$.

— m neurones $N_1, \ldots, N_m$.

— $w_i j$ le coefficient synaptique de la liaison entre les neurones N_i et N_j.

— une sortie o.

— un seuil S.

— Fonction de transfert ; fonction Signe :

$$\begin{cases} Signe(x) = +1 & \text{Si : x>0} \\ Signe(x) = \breve{}1 & \text{Si : x<0} \end{cases}$$

Si deux unités connectés (2 neurones) sont activés simultanément (en même temp) ,le poid de leur connexion est augmenté ou diminué,on peut reformuler l'énoncé de Hebb sous la forme d'une règle d'apprentissage en deux parties :

1. Si deux neurones sont activés simultanément (d'une manière synchrone), alors la force de connexion doit être augmentée .
2. Si les mêmes deux neurones sont activés d'une manière asynchrone, alors la force de connexion correspondante doit être affaibli ou carrément éliminé.

α est une constante positive qui représente la force d'apprentissage :

$$\Delta c_i j = \alpha . x_i . yj$$

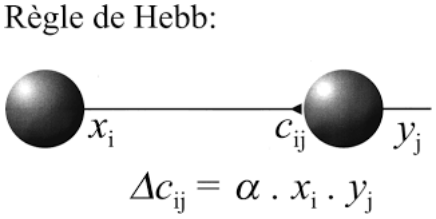


FIGURE 2.5 – Loi de Hebb

Base d 'apprentissage : On note S la base d'apprentissage. S est composée de couples (e, c) où : e est le vecteur associé l'entrée $(e_1, \ldots, e_n)$ et c la sortie correspondante souhaitée.

L'algorithme de Hebb :

La loi de Hebb peut être modélisée par les équations suivantes ($w_{(t+1)}$ est le nouveau poids :

$$w_ij(t)(l'ancien) : w_ij(t+1) = w_ij(t) + \partial w_ij(t)$$

$\partial w_ij(t) = x_i.x_j$(la coactivité est modélisée comme le produit des deux valeurs d'activation)
L'algorithme d'apprentisage modifie de façon itérative (petit à petit) les poids pour adapter la réponse obtenue à la réponse désirée. Il s'agit en fait de modifier les poids lorsqu'il y a un erreur seulement.

1. Initialisation des poids et du seuil S à des valeurs (petites) choisies au hasard.
2. Présentation d'une entrée $E_1 = (e_1, ...e_n)$ de la base d'apprentissage.
3. Calcul de la sortie obtenue x pour cette entrée :
   $a = \sum(w_i.e_i) - S$ (la valeur de seuil est introduite ici dans le calcul de la somme pondérée)
   x = signe (a) (si a > 0 alors $x = +1$ sinon $x = -1$)
4. Si la sortie x est différente de la sortie désirée dl pour cet exemple d'entrée El alors : Modification des poids (est une constante positive, qui spécifie le pas de modification des poids) :

   $$w_ij(t+1) = w_ij(t) + \mu.(x_i.x_j)$$

5. Tant que tous les exemples de la base d'apprentissage ne sont pas traités correctement : modification des poids et retour à l'étape 2.

Exemple d'application de l'algorithme d'apprentissage de Hebb :
nous allons réaliser l'apprentissage sur un problème très simple. La base d'apprentissage est décrite par la table suivante :

e1	e2	x
1	1	1
1	-1	1
-1	1	-1
-1	-1	-1

1/ Conditions initiales : $\mu = +1$, les poids et le seuil sont nuls.
2/ Calculons la valeur de x pour l'exemple (1)
3/$a = w_1.e_1 + w_2.e_2 = 0.1 + 0.1 = 0eta \leq 0 => x = -1$
4/ La sortie est fausse, il faut donc modifier les poids en appliquant :
$w_1 = w_1 + e_1.x = 0.0 + 1.1 = 1$
$w_2 = w_2 + e_2.x = 0.0 + 1.1 = 1$
2/ On passe à l'exemple suivant (2) :
3/ $a = 1.1 + 1. - 1 = 0eta \leq 0 => x = -1$
4/ La sortie est fausse, il faut donc modifier les poids en appliquant :
$w_1 = 1 + 1.1 = 2$
$w_2 = 1 + 1. - 1 = 0$
. . . / L'exemple suivant (3) est correctement traité : a = -2 et x = -1 (la sortie est bonne).
On passe directement, sans modification des poids à l'exemple (4).
Celui-ci aussi est correctement traité. On revient alors au début de la base d'apprentissage :
L'exemple (1) : il est correctement traité, ainsi que le second (2).
L'algorithme d'apprentissage est alors terminé : toute la base d'apprentissage a été passée en revue sans modification des poids. [int aux res et connexi]

2.3.2 Règle de Delta (Widrow Hoff)

Une généralisation de cet algorithme d'entrainement est donnée par la règle d'apprentissage du "moindre carré" ou (LMS) pour 'least mean square', crée par Widrow et Hoff. Cette règle utilise la valeur brute de la sortie d'un neurone plutôt que celle donnée par la fonction d'activation. Cette méthode vise à minimiser l'erreur produite sur les valeurs des sorties par rapport à celles désirées en modifiant les poids avec une fonction d'erreur définie par :

$$E = \sum_p E^p = \frac{1}{2}\sum_p (d^p - y^p)^2$$

p est l'indice du neurone produisant la sortie y et ayant comme sortie désirée d. La correction produite sur un poids est proportionnelle à la dérivée de la fonction d'erreur par rapport à ce poids :

$$\Delta_p w_j = -\gamma \frac{\partial E^p}{\partial w_j}$$

Où est une constante de proportionnalité.
Cette dérivée s'exprime par :

$$\frac{\partial E^p}{\partial w_j} = \frac{\partial E^p}{\partial y^p}\frac{\partial y^p}{\partial w_j}$$

La sortie y d'un neurone lié à un biais par un poids , et par des poids w_j aux neurones j ayant des entrées x_j, s'exprime par :

$$y = \sum_j w_j x_j + \theta$$

D'où :

$$\frac{\partial y^p}{\partial w_j} = x_j$$

Et :

$$\frac{\delta E^p}{\delta y^p} = -(d^p - y^p)$$

Alors, on déduit l'expression de $\Delta_p w_j$:

$$\Delta_p w_j = \gamma (d^p - y^p) x_j$$

$$\Delta_p w_j = \gamma \delta^p x_j$$

Algorithm 2.1 *Algorithme Widrow Hoff*

1. **Entrée** :
   — n poids reliant les n informations à notre neurones ayant des valeurs quelconques
   — N exemples (X_k, y_k) ou X_k est un vecteur à n composantes x_i, chacune représentant une information de cet exemple
   — Le taux d'apprentissage α
2. **Sortie :** les n poids modifiés
3. **Pour** Tout exemple $= (X_k, y_k)$ **faire**
   Calculer la sortie s_k du neurone
4. **Pour** $1 \leq i \leq n$ **faire**
   $w_i = w_i + \alpha(y_k - s_k)x_i$
5. **Fin pour**
6. **Fin pour**

2.4 Algorithmes d'apprentissage supervisé

C'est le plus ancien des algorithmes d'apprentissage proposé par Rosenblatt en 1958,il peut etre utilisé comme une méthode de gradient exact(algorithme du cout total :les modifications des paramètres du réseau sont opérées aprés présentation de tout les exemples de la base d'apprentissage)ou bien comme méthode de gradient stochastique(algorithme du cout partiel :dans ce cas ,on modifie les paramètres du réseau aprés présentation de chaque exemple).

2.4.1 Algorithme de coût total

Dans cet algorithme, on modifié les poids en fonction du gradient de l'erreur quadratique totale sur l'ensemble d'apprentissage selon la relation :

$$\begin{cases} j(W) = \sum_{k=1}^{N} j^k(W) \\ j^k(W) = \begin{cases} y_p^k . W(X^k)^T & \text{Si X est mal classé} \\ 0 & sinon \end{cases} \end{cases} \tag{2.1}$$

Cet algorithme est itératif,à l'itération i, on aura les paramètres W_i qui seront modifiés dans le sens opposé au gradient calculé en W(i-1).Le gradient est obtenu par la formule :

$$\frac{j}{W} = \sum_{k=1}^{N} \frac{\partial j^k}{\partial W} avec \frac{\partial j^k}{\partial W} = \begin{cases} -y_p^k . X^k & \text{Si X est mal classé} \\ 0 & sinon \end{cases} \tag{2.2}$$

2.4.2 Algorithme de coût partiel

C'est un algorithme permet de modifier les paramètres du neurone en fonction du gradient de l'erreur partielle sur chaque exemple. Le coût est donné par la relation :

$$f(W) = \left\{ -Y_p^k.W.(X^k)^T \right. \tag{2.3}$$

son gradient vaut :

$$\frac{\partial f^k}{\partial W} = \begin{cases} -Y_p^k.X^k & \text{Si X est mal classé} \\ 0 & sinon \end{cases}$$

à la présentation du K^ieme exemple, si la réponse souhaité et la réponse réelle sont identiques(exemple bien classé),ce qui se traduit par :$Y_p^k.W.(X^k)^T > 0$. On applique :
$W(k) = W(k-1)$ c'est à dire les poids sont inchangés,tandis que,si la réponse calculéé est différente de la réponse désirée(exemple mal classé), on aura alors : $Y_p^k.W.(X^k)^T < 0$.On applique :$W(k) = W(k-1) + Y_p^k.X^k$.
Les paramètres du neurone sont initialisés aléatoirement.on calcule les potentiels des N exemples avec les paramètres obtenus afin de savoir s'il existe des exemples mal classés.
Dans le cas positif, on recommence avec un autre exemple. Sinon on s'arrête et la séparation des exemples en deux classes est réussie.le fonctionnement de l'algorithme de perceptron est décrit par les six étapes suivants :

1. choisir des valeurs aléatoires pour les poids et le biais.
2. présenter un vecteur d'entré X^k au réseau.
3. calculer la valeur de la sortie S du perceptron :

$$S = \sum_{i=1}^{n} W_i.x_i^{k+b}.$$

4. si la valeur de sortie est égal à la valeur cible ,retourner à l'étape 2 sinon continuer à l'étape 5.
5. la variation du biais est égale à la valeur de la cible.La variation du poids W est égale à la valeur de l'entrée multipliée par la valeur de la cible :

$$\begin{cases} \Delta b = Y_p^k \\ \Delta W = X^k.Y_P^K \end{cases}$$

6. ajusterles poids et le biais avec les nouvelles valeurs obtenus :

$$\begin{cases} b = b + \Delta b \\ W = W + \Delta W \end{cases}$$

et puis retourner à l'étape 2.

2.4.3 Algorithme du perceptron

On note S la base d'apprentissage. S est composée de couples (x, c) où : x est le vecteur associé à l'entrée $(x_0, x_1, \ldots, x_n)$ c la sortie correspondante souhaitée On cherche à déterminer les coefficients $(w_0, w_1, \ldots, w_n)$.

Algorithm 2.2 *Algorithme du perceptron*

1. **Initialiser** : aléatoirement les coefficients wi.
2. **Répéter** :
   a) Prendre un exemple (x, c) dans S
   b) Calculer la sortie o du réseau pour l'entrée x
   c) Mettre à jour les poids
3. **Pour** i de 0 à n
   $w_i = w_i + (c - o)x_i$
4. **Fin pour**
5. **Fin Répéter**

Le problème XOR :

Le problème du XOR consiste à classer les quatre sommets du cube binaire $B^2 = \{0, 1\}^2$ en deux classes selon leur valeur par la fonction logique "ou exclusif", ce qui correspond à l'ensemble d'apprentissage :

$$E = \{((0,0)^t, -1), ((1,1)^t, -1), ((0,1)^t, 1), ((1,0)^t, 1)\}$$

Les deux classes ainsi définies ne sont évidemment pas linéairement séparable et ne peuvent être apprises ni par l'algorithme du perceptron, ni par celui l'Adaline. Néanmoins, ces deux méthodes proposent des solutions pour contourner cette écueil [32].
Le perceptron peut résoudre le problème du XOR moyennant une couche associative judicieusement choisie. En effet, on vérifie facilement que :

$$y = XOR(x_1, x_2) = 1(x_1 + x_2 - 2x_1.x_2 - 1)$$

On utilisant trois cellules associatives w_1 w_2 w_3 telles que $w_1 = (x_1, x_2) = x_1$, $w_2 = (x_1, x_2) = x_2$ et $w_3 = (x_1, x_2) = x_1.x_2$, on construit un espace dans lequel le problème du XOR est résoluble par l'algorithme du perceptron Figure 2.7.

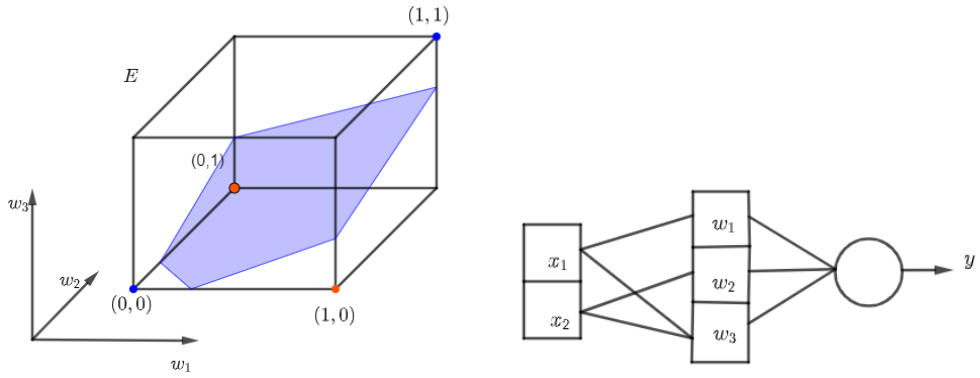
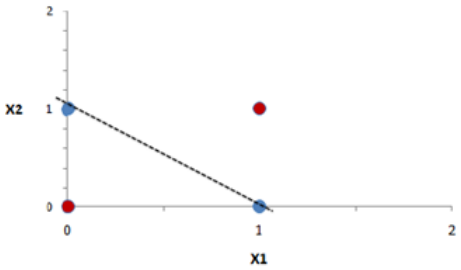


FIGURE 2.6 – Le perceptron construit une séparation linéaire dans un espace transformé, celui de la couche cachée des celulles associatives qui est construite manuellement pour rendre les formes séparables.

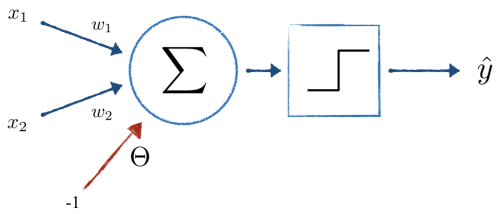
Parce que le perceptron monocouche est un classificateur linéaire et si les cas ne sont pas linéairement séparables, le processus d'apprentissage n'atteindra jamais un point où tous les cas sont classés correctement.
L'exemple le plus célèbre de l'incapacité du perceptron à résoudre des problèmes avec des cas linéairement non séparables est le problème XOR.



Theorem 3

Le XOR ne peut pas être calculé par un perceptron linéaire à seuil.

Exemple : ET logique
Il est possible d'apprendre à un perceptron la porte logique ET dont voici les tables de vérité :



X	Y	ET
0	0	0
0	1	0
1	0	0
1	1	1

FIGURE 2.7 – Neurone de la porte ET logique

Prenons la fonction de transfert "Heaviside" :

$$f(x) = \begin{cases} 1 \quad si \quad \text{x>0} \\ 0 \quad sinon \end{cases}$$

Notre perceptron schématisé ci-dessous figure 2.7 présente deux entrées (x_1 et x_2) pondérées par des poids (w_1 et w_2), ainsi qu'un biais . Il pourrait être vu comme une troisième entrée toujours égale à -1 dont on pourrait également modifier le poids. Ces informations sont sommées et passées dans une fonction de transfert pour calculer une sortie.

A chaque itération nous allons donc effectuer l'opération suivante :

$$\hat{y} = f(x_1 \times w_1 + x_2 \times w_2 - \theta)$$

Ici $\hat{y}$ est la prédiction du modèle. Nous comparerons ensuite le résultat obtenu avec le résultat attendu pour mettre à jour les poids et le biais :

$$w_i = w_i + \Delta w_i = w_i + (y - y)x_i.\eta$$

2.4.4 Algorithme des moindres carrés

On dispose d'un ensemble d'apprentissage $\{X^k, Y_p^k\}_{k=1.N}$. Le cout quadratique est défini par :

$$j(W) = \frac{1}{2}\sum (Y_p^k - W(X^k)^T)^2 \tag{2.4}$$

L'objectif étant d'optimiser le vecteur W en minimisant le coût $j(W)$ de la formule (2.4), qui s'écrit sous forme matricielle comme suit [40] :

$$j(W) = \frac{1}{2}\|y_p - m.W\|^2 \tag{2.5}$$

OU : m est la matrice dite de l'expérience de dimensions (N, n) définie par :

$$m = \begin{bmatrix} x_1^1 & \dots & x_n^1 \\ \vdots & \ddots & \\ \vdots & \ddots & \\ x_1^N & \ddots & x_n^N \end{bmatrix} = \begin{bmatrix} (x^1)^T \\ \vdots \\ (x^N)^T \end{bmatrix} = (x_1, x_2, ..., x_n)$$

Avec :

$X^t = [x_1^k, ..., x_n^k]$: Vecteur constitué des n valeurs des entrées de l'exemple K.

$X^t = [x_t^1, ..., x_t^N]^T$: Vecteur des N valeurs prises par l'entrés i.

$Y^t = [y_1^t, ..., y_p^N]^T$: Vecteur de sortie désirée.

$W^t = [w_1, ..., w_n]^T$: Vecteur poids.

La formule (2.5) s'écrit alors comme suit [40] :

$$\begin{aligned} j(W) &= \frac{1}{2}\|y_p - m.W\|^2 \\ &= \frac{1}{2}(y_p - m.W)^T(y_p - m.W) \\ &= \frac{1}{2}(y_p^T y_p - 2.W^T y_p + W^T m^T m.W) \end{aligned}$$

Le gradient est obtenu par dérivation du j par rapport à w :

$$\frac{\partial j}{\partial W} = -m^T y_p + m^T.m.W \tag{2.6}$$

La solution des moindres carrés s'obtient on posant : $\frac{\partial j}{\partial W} = 0$
c'est-à-dire :

$$-m^T y_p + m^T.m.W = 0 \Longrightarrow (m^T.m).W_{MC} = m^T.y_p \tag{2.7}$$

W_{MC} est le vecteur poids des moindres carrés. Généralement, le nombre d'entrées du neurone est nettement moins important que le nombre d'exemples d'apprentissage, et dans ce cas le rang de la matrive m est n[5]. Par conséquent, le rang de la matrice $(m^T.m)$ est également n, on multipliant les deux termes de l'équation définie par la formule (2.7) par $(m^T.m)^{-1}$ on obtient :

$$\begin{gathered} (m^T.m)^{-1}.(m^T.m)W_{MC} = (m^T.m)^{-1}.m^T y_p \\ \Longleftrightarrow W_{MC} = (m^T.m)^{-1}.m^T y_p \end{gathered}$$

Cette formule définit la solution des moindres carrés. L'hypothèsesolution est :

$$v = m^T.W_{MC} = 0$$

IL est appelé hyperplan MC-potentiel [40].

Exemple :
Soit à calculer les paramètre poids d'un neurone à deux entrées X_1 et X_2. L'ensemble d'apprentissage est constitué de 3 exembles : Pour $k = 1, 2, 3$
Et $X_1^k = 1$
$X_2^1 = -1$
$X_2^2 = 0$
$X_2^3 = 1$
Le vecteur de sortie désirée est $Y_p = [-0.5, 1.7, 1.3]^T$

La matrice de l'expérience m est donc :

$$m = \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

La solution des moindres carrés est :

$$W_{MC} = (m^T.m)^{-1}.m^T.y_p = [0.833, 0.9]^T$$

On démontre que l'algorithme des moindres carrés présente deux inconvénients [40] :

— D'une part, l'algorithme converge même s'il n'existe pas aucun hyperplan séparant les exemples linéairement et dans ce cas, certains exemples sont mal classés Figure 2.8.(5 exemples de la classes "boules en rouge" sont mal classés, et 3 exemples de la classe "boules en bleu" sont mal classés)
— D'autre part, si les exemples sont séparables linéairement, l'algorithme ne les séparent pas nécessairement Figure 2.9. (Un exemple de la classe "boules en bleu" sont mal classés)

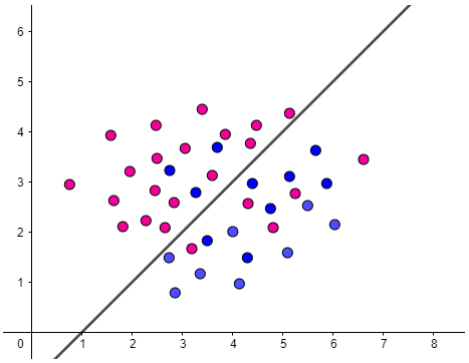


FIGURE 2.8 – Exemples non séparables linéairement.

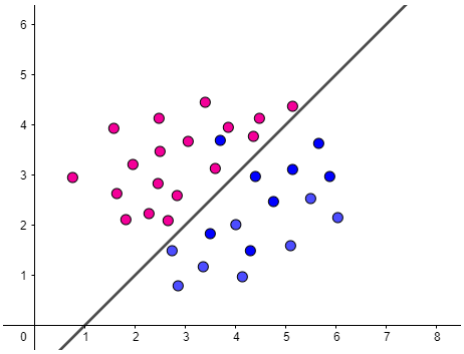


FIGURE 2.9 – Exemples séparables linéairement, mais l'algorithme des moindres carrés ne les séparent pas.

2.4.5 Algorithme d'apprentissage par correction d'erreur

Soit un ensemble d'apprentissage S de $\{0,1\}^n \times \{0,1\}$ ou $IR^n \times \{0,1\}$, c'est-à-dire un ensemble d'exemples dont les descriptions sont sur n attributs réels (respectivement binaires) et la cible réelle (respectivement binaire). Il s'agit de trouver un algorithme qui infère à partir de S un perceptron qui prédit correctement les éléments de A au vu de leurs descriptions, et au mieux sinon.

L'algorithme d'apprentissage peut être décrit de la manière suivante :

1. On initialise le poids du perceptron à des valeurs quelconques. A chaque fois que l'on présente un exemple, on ajuste les poids selon que le perceptron l'ait correctement classé ou non. L'algorithme s'arrête lorsque tous les exemples lui ont été présentés sans aucune modification des poids.
2. nous désignerons par $\vec{x}$. La i^{me} posante de $\vec{x}$ sera notée x_i, un échantillon S est un ensemble de couples $(\vec{x}, c)$ ou c est la valeur à prédire de Lorsqu'il sera utile de désigner un élément de S, nous noterons $(\vec{x}^j, c^j)$ le j^{me} élément de S. x_i^j désigne donc la i^{me} composante du vecteur $\vec{x}^j$, Si $\vec{x}^j$ est présenté en entrée au perceptron, nous noterons o^j la sortie calculée par le perceptron.

3. Le processus d'apprentissage du perceptron, est un processus de correction d'erreur, puisque les poids ne sont pas modifiés lorsque la sortie attendue st égale à la sortie calculée par le perceptron courant. Etudions les modifications apportées au poids pour un perceptron linéaire à sortie binaire.

   a) $o = 0$ et $c = 1$, cela signifie que le perceptron n'a pas assez pris en compte les neurones actifs de l'entrée (c'est-à-dire les neurones ayant une entrée à 1) ; dans ce cas $w_i \longleftarrow w_i + x_i$ gorithme ajoute de la valeur de l'entrée au poids synaptique (renforcement).

   b) $o = 1$ et $c = 0$ alors $w_i \longleftarrow w_i - x_i$, l'algorithme retranche la valeur de l'entrée au poids synaptique (inhibition).

Algorithm 2.3 *Algorithme par correction d'erreurs*

1. **Entrée** : un échantillon S
2. **pour** $i = 0, ..., n$
   initialiser aléatoirement les poids wi
3. **Répéter**
   a) Prendre un exemple $(\vec{x}, c)$ dans S
   b) Calculer la sortie o pour l'entrée $\vec{x}$
4. **Pour** $i = 0...n$
   $w_i \longleftarrow w_i + (c - o)x_i$
5. **Fin pour**
6. **Fin Répéter**
7. **Sortie** : un perceptron P défini par $(w_0, w_1, ..., w_n)$

Exemple : Apprentissage du OU logique :
Les descriptions appartiennent à $\{0, 1\}^2$, les entrées du perceptron appartiennent à $\{0, 1\}^3$, la première composante correspond à l'entrée x_0 vaut toujours 1, les deux composantes suivantes correspondent aux variables $x1$ et $x2$. Ce perceptron calcule le OU logique pour tout couple $(x1; x2)$

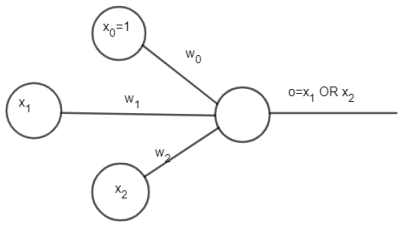


FIGURE 2.10 – Le perceptron qui correspondant à l'apprentisage du OU logique.

a	b	OU
0	0	1
0	1	1
1	0	1
1	1	1

$\epsilon = 1$
On suppose qu'à l'initialisation les poids suivant ont été choisis : $w_0 = 0$, $w_0 = 1$, $w_0 = 1$

Étape	$w0$	$w1$	$w2$	Entrée	$\sum w_i x_i$	o	c	$w0$	$w1$	$w2$
init					0	0	0	0	1	-1
1	0	1	-1	100	-1	0	1	0+1x1	1+1x0	-1+0x0
2	0	1	-1	101	2	1	1	0+0x1	1+0x0	-1+0x0
3	1	1	0	110	2	1	1	1	1	0
4	1	1	0	111	1	1	0	1	1	0
5	1	1	0	100	0	0	1	1+(-1)x1	1+(-1)x0	0+(-1)x0
6	0	1	0	101	0	0	1	0+1x1	1+1x0	0+1x1
7	1	1	1	110	2	1	1	1	1	1
8	1	1	1	111	3	1	1	1	1	1
9	1	1	1	100	1	1	0	1+(-1)x1	1+(-1)x0	1+(-1)x0
10	0	1	1	101	1	1	1	0	1	1

TABLE 2.1 – Valeurs des poids à chaque itération

Donc : $w0 = 0$; $w1 = 1$; $w2 = 1$

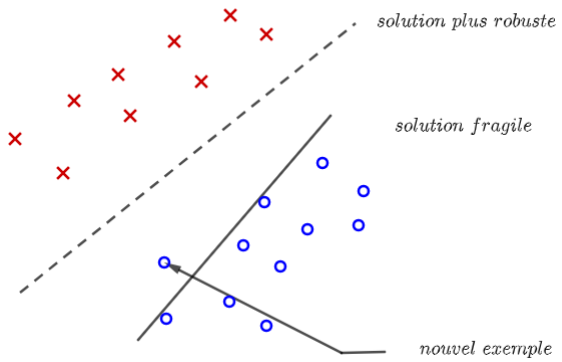


FIGURE 2.11 – L'algorithme peut converger vers plusieurs solutions (selon les valeurs initiales des coefficients, la valeur de ϵ, l'ordre de présentation des exemples)

Critiques :
L'algorithme par correction d'erreur converge à condition que l'échantillon d'apprentissage soit linéairement séparable. De plus, il n'existe aucun moyen de savoir que celui-ci n'est pas convergent.
Le but des sections suivantes est de présenter des algorithmes qui conduisent à des solutions plus ou moins robustes selon que l'échantillon d'apprentissage soit linéairement séparable ou non.
Remarques :
•bien choisi, suffisamment petit.
•Si trop grand : risque d'oscillation autour du minimum.
•Si trop petit : nombre élevéd'itérations.
•En pratique : on diminue graduellement au fur et àmesure des itérations.

Theorem 4

Si l'échantillon S est linéairement séparable et si les exemples sont présentés équitablement (c'est-à-dire que la procédure de choix des exemples n'en exclut aucun), la procédure d'apprentissage par correction d'erreur converge vers un perceptron linéaire à seuil qui calcule S.

2.4.6 Algorithme par descente du gradient

Méthode du gradient

Soient $\mathbb{E}$ un espace hilbertien (produit scalaire noté $\langle .,. \rangle$ et norme associée notée $\|.\|$) et $x \in \mathbb{E} \longmapsto f(x) \in \mathbb{R}$ une fonction différentiable. On note $df(x)$ la différentielle de f en x et $\nabla f(x)$ le gradient de f en x, si bien que pour tout direction $d \in \mathbb{E}$, on a $df(x)(d) = \langle \nabla f(x), d \rangle$

On se donne un point/itéré initial $x_0 \in \mathbb{E} x_0 \in \mathbb{E}$ et un seuil de tolérance $\varepsilon \geqslant 0 \varepsilon \geqslant 0$.
L'algorithme du gradient définit une suite d'itérés $x_1, x_2, \ldots \in \mathbb{E}$, jusqu'à ce qu'un test d'arrêt soit satisfait. Il passe de x_k à x_{k+1} par les étapes suivantes.

1. Simulation : calcul de $\nabla f(x_k)$.
2. Test d'arrêt : si $\|\nabla f(x_k)\| \leqslant \varepsilon$, arrêt.
3. Calcul du pas $\alpha_k > 0$ par une règle de recherche linéaire sur f en x_k le long de la direction $-\nabla f(x_k)$.
4. Nouvel itéré : $x_{k+1} = x_k - \alpha_k \nabla f(x_k)$.

— x_{k+1} est le point suivant
— x_k est le point courant
— α est le pas de l'algorithme (step size multiplier)

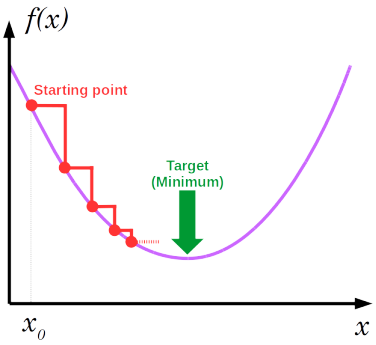
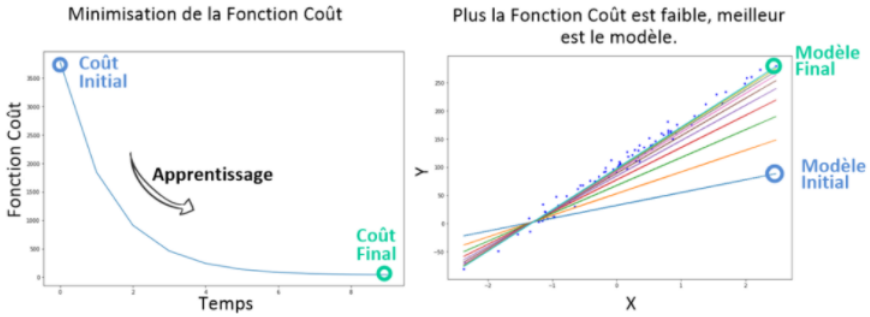


FIGURE 2.12 – La descente de gradient

La Descente de Gradient, est un algorithme d'optimisation qui permet de trouver le minimum de n'importe quelle fonction convexe en convergeant progressivement vers celui-ci. Il permet d'entraîner les modèles de régression linéaire, régression logistiques ou encore les réseaux de neurones.
En Machine Learning, on va utiliser l'algorithme de la Descente de Gradient dans les problèmes d'apprentissage supervisé pour minimiser la fonction coût, qui justement est une fonction convexe.



Soit un perceptron linéaire P ayant pour entrer un vecteur de n valeurs $(x_1, x_2, ..., x_n)$ calcule une sortie o.Un perceptron est défini par la donnée d'un vecteur w de n constantes : les coefficients synaptiques $(w_1, w_2, ..., w_n)$. La sortie o est définie par :

$$o = \vec{x}.\vec{w} = \sum_{i=1}^{n} x_i w_i \tag{2.8}$$

L'erreur du perceptron P, défini par $\vec{w}$ sur l'échantillon d'apprentissage A, est définie en utilisant la fonction d'erreur quadratique par :

$$E(\vec{w}) = \frac{1}{2} \sum_{(\vec{x},c)\in A} (c - o)^2 \tag{2.9}$$

Algorithm 2.4 *Algorithme par descente du gradient*

1. **Entrée** : un échantillon S de $E \times S$
   initialiser aléatoirement les poids wi pour $i = 0, ..., n$
2. **Répéter**
3. **Pour** $i = 0...n$ $Dw_i \longleftarrow 0$
4. **Fin Pour**
5. **Pour** tout exemple $(\vec{x}^j, c^j)$ de S calculer la sortie o^j
6. **Pour** tout i
   $Dw_i \longleftarrow Dw_i + e(c^j - o^j)x_i^j$
7. **Fin pour**
8. **Pour** tout i
   $w_i \longleftarrow w_i + Dw_i$
9. **Fin pour**
10. **Fin Répéter**
11. **Sortie** : un perceptron P défini par $(w_1, w_2, ..., w_n)$

Descente de gradient stochastique :

Gradient stochastique est une approximation de la descente de gradient, applicable lorsque la fonction objectif s'écrit comme une somme de fonctions dérivables : c'est très souvent le cas en apprentissage supervisé.

Exemple de la régression linéaire multiple via les moindres carrés :

$$S = \sum_{i=1}^{n} (y_i - \langle a, x_i \rangle)^2$$

Est dérivable par rapport auparamètres (a_j)

$$(y_i - \langle a, x_i \rangle)^2$$

Il est possible de corriger les paramètres estimés pour le passage de chaque observation.

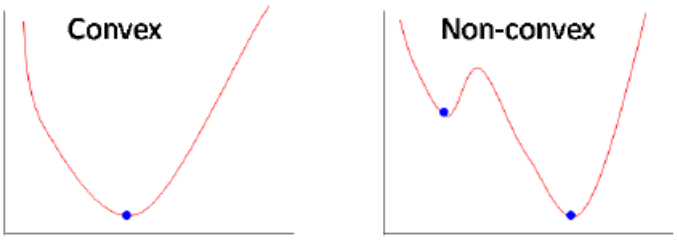
$$a := a - \eta \times \nabla S_i$$

Où :

$$\frac{\partial S_i}{\partial a_j} = (-x_{ij}) - (y_i - \langle a, x_i \rangle)$$

La fonction Coût :

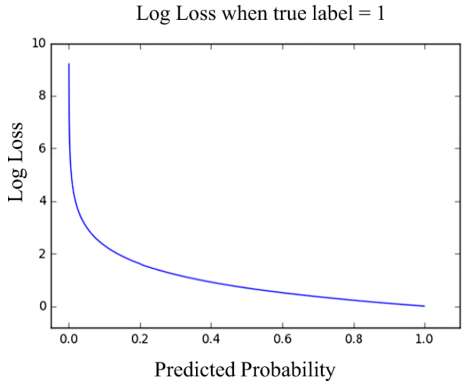
En machine learning une fonction Coût est une fonction qui permet de quantifier les erreurs effectués par un model, c'est la fonction qui permettre de mesurer les distances entre les sorties et les données y que nous disposons.
Le problème de l'apprentissage peut etre formulé comme un problème d'optimisation c'est à dire pour chaque exemple d'entrâinement ,on souhaite minimiser une certaine distance entre la cible Y_t et la prédiction $H_W(t)$,on appelle cette distance une perte. Dans le cas de perceptron la fonction qui permettre de mesurer ces distances est la fonction Log Loss ,c'est une fonction convexe c'est à dire elle ne contient qu'un seul minimum.



sa formule mathématique est donnée par :

$$Loss(y_t, H_W(t)) = -\frac{1}{m} \sum_{i=1}^{m} y^i \log(H^i) + (1 - y^i) \log(1 - H^i)$$

Si la prédiction est bonne,le coût est nulle.
Si la prédiction est mauvaise ,la perte est la distance entre $W.X_t$ et le seuil à franchir pour que la prédiction soit bonne.



Pour minimiser les erreurs de notre model on utilise l'algorithme de descente de gradient qui consiste à ajuster les paramètres (W et b)

Pour cela on calcul le gradient(la dérivée) de la fonction cout(Log Loss).

En mathématique : la dérivée d'une fonction indique comment une fonctin varie
$\frac{\partial L}{\partial W} < 0$: indique que la fonction diminue quand W augmente ,donc il faut augmenter W si on veut reduire nos erreurs.
$\frac{\partial L}{\partial W} > 0$: cela indique que la fonction cout augmente quand W augmente ,donc il faut diminuer W si on veut reduire nos erreurs.
pour faire ça on utilise la formule suivant :

$$W_{t+1} = W_t - \alpha.\frac{\partial L}{\partial W} b = b - \alpha.\frac{\partial L}{\partial W}$$

W_{t+1} :paramètre W à l'instant t
W_t :paramètre W à l'instant t
α :pas d'apprentissage positif
$\frac{\partial L}{\partial W}$:gradient à l'instant t
Répétons cette formule jusqu'a atteindre au minimum de fonction cout.

2.5 Perceptron pour la classification binaire

L'algorithme de perceptron, dans sa version standard, est dédié à la classification binaire. Les données d'apprentissage utilisées dans cette section s'écrivent donc sous la forme $S = \{(x_i, y_i)\}_{i=1}^n$ avec $x_i \in \mathbf{R}^d$ (d est le nombre d'attributs des données d'entrée) et $y_i \in \{-1, 1\}$. Les étiquettes y_i ne peuvent prendre que deux valeurs (1 ou -1), d'où l'appellation classification binaire.
Un perceptron binaire est un classifieur linéaire. A partir des données $\{(x_i, y_i)\}_{i=1}^n$, l'objectif est d'apprendre un vecteur $w \in \mathbf{R}^d$ tel que $\langle w, x_+\rangle > 0$ pour tout x_+ appartenant à la classe 1 et $\langle w, x_+\rangle \leq 0$ pour tout x_- appartenant à la -1. $\langle w, x\rangle := \sum_{j=1}^d w^{(j)} x^{(j)}$ est le produit scalaire entre les deux vecteurs w et x, avec $w(j)$ et $x(j)$ les j-èmes composantes des vecteurs w et x.
L'idée de l'algorithme du perceptron est d'initialiser w au vecteur nul, itérer un nombre de fois (fixé a priori ou jusqu'à convergence) sur les donn´ees d'apprentissage, et ajuster le vecteur de pondération w à chaque fois qu'une donnée est mal classée.

Algorithm 2.5 *Algorithme Perceptron binaire*

— **Entrée** : une liste S de données d'apprentissage, $(x_1, y_1), ..., (x_n, y_n)$ où $x_i \in \mathbf{R}^d$ et $y_i \in \{-1, 1\}$, le nombre d'itérations N.
— **Sortie** : les vecteurs de pondération w.

1. Initialiser les vecteurs $w \longleftarrow 0$
2. **Pour** iteration $= 1$ à N **faire**
3. **Pour** chaque exemple $(x_i, y_i) \in S$ **faire**
   Calculer la prédiction $\hat{y}_i = signe(\langle w, x_i\rangle)$
4. **Si** $\hat{y}_i = y_i$ **Alors**
   Ajuster w :$w \longleftarrow w + x_i$ si y_i est positive,
   ou $w \longleftarrow w - x_i$ si y_i est négative .
5. **Fin si**
6. **Fin pour**
7. **Fin pour**

2.6 Perceptron multi-classe

Un Perceptron multi-classe généralise le principe de classification linéaire du Perceptron binaire au cas où le nombre de classes peut être supérieur à deux. A partir d'un jeu de données $\{(x_i, y_i)\}_{i=1}^n$, où maintenant $y_i \in \{l_1, ..., l_c\}$ et C est le nombre de classes figure 2.13, l'objectif est d'apprendre un ensemble de vecteurs de pondération $w_{l1}, ..., w_{lc}$ tel que la classe prédite par $\arg\max_{lk}\langle w_{lk}, x\rangle$, soit le plus souvent en accord avec la vraie classe y d'un exemple x. L'algorithme d'apprentissage pour ce problème est similaire à celui pour le cas binaire. Tous les vecteurs de pond´eration sont d'abord initialisés à zéro, puis plusieurs itérations sont effectuées sur les données d'apprentissage, avec ajustement des vecteurs de pond´eration chaque fois qu'une paire de données d'apprentissage est incorrectement étiquetée.

1. Implémenter l'algorithme du perceptron multi-classe.
2. Ecrire une fonction permettant de prédire, à partir d'un perceptron multi-classe, la classe associée à une donnée d'entrée .
3. Tester l'algorithme sur le jeu de données.

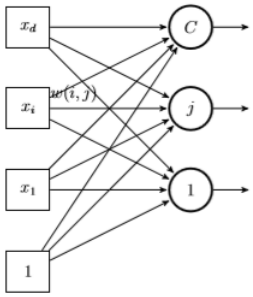


FIGURE 2.13 – Le perceptron pour C classes.

Algorithm 2.6 *Algorithme Perceptron multi-classe*

— **Entrée** : une liste S de données d'apprentissage, $(x_1, y_1), ..., (x_n, y_n)$ où $x_i \in \mathbf{R}^d$ et $y_i \in \{l1, ..., lc\}$, le nombre d'itérations N.
— **Sortie** : les vecteurs de pondération $w_{l1}, ..., w_{lc}$.

1. Initialiser les vecteurs $w_{lk} \longleftarrow 0$ pour $k = 1, ..., c$
2. **Pour** iteration $= 1$ à N **faire**
3. **Pour** chaque exemple $(x_i, y_i) \in S$ **faire**
   Calculer la prédiction $\hat{y}_i = \arg\max_{lk}\langle w_{lk}, x_i\rangle$
4. **Si** $\hat{y}_i = y_i$ **Alors**
   Ajuster le score pour la "vraie" classe : $w_{y_i} \longleftarrow w_{y_i} + x_i$
   Ajuster le score pour la classe prédite : $w_{\hat{y}_i} \longleftarrow w_{\hat{y}_i} - x_i$
5. **Fin si**
6. **Fin pour**
7. **Fin pour**

2.7 Exemple général de perceptron

Considérons un ensemble de données contenant des informations sur des fruits ayant certaines caractérisques et dont on veut prédire si c'est un fruit délicieux ou pas :

Fruit	Goût	Graine	Chair	Délicieux ?
1	1	1	0	1
2	1	0	1	1
3	0	0	0	0
4	1	1	1	1
5	0	0	1	0

TABLE 2.2 – Exemple d'un dataset pour étude de cas d'un perceptron

Dans l'exemple ci-haut , les caractéristiques sont : goût, graine et chair. C'est en fonction de ces trois caractéristiques que l'on peut savoir si le fruit est délicieux ou pas [23]. Et, 1 dans la colonne ayant la cellule jaune représente délicieux et 0 représente pas délicieux. Cette dernière colonne est appelée $Target(y)$. Tandis que les trois colonnes qui la précédent (caractéristiques) sont appelées Features.

De ces données, il faut maintenant créer un modèle de neurone pour arriver à prédire, à partir des nouvelles données des fruits si, partant de leurs caractéristiques, ils sont délicieux ou pas. Ainsi, doter ce modèle de cette capacité, en terme technique se dit entrainer le modèle.

Précisons qu'un Perceptron est composé d'une couche d'entrée, appelée aussi vecteur de n neurones qui correspond chacune à une variable d'entrée (Dans notre exemple, le fruit 2 a pour variables d'entrée (1,0,1) et ces neurones vont transmettre la valeur de leur entrée à la couche suivante, dans le cas échéant, à la couche de sortie. Toutefois, pour ajouter de la flexibilité à l'algorithme du perceptron, en permettant de varier le seuil de déclenchement du neurone par l'ajustement des poids lors de l'apprentissage, la nouvelle version du perceptron intègre l'ajout d'une unité appelée biais qui prend souvent -1 et 1 comme valeur [2]. Ce qui fait que le vecteur fruit 2 devient $\vec{x}(1,1,0,1)$, si on ajoute le biais de 1. Dans le cas de neurone formel, le biais joue le rôle de seuil d'activation. Ayant des valeurs d'entrées, le réseau ne manque que des poids qui pondèrent les entrées et peuvent être modifiés par apprentissage. On peut nommer le poids W. Le poids $W_j i$ est donc la connexion entre le neurone d'entrée j et le neurone de sortie i dont le processus d'activation de la sortie s'illustre de la manière suivante :

Si nous retournons à l'exemple de notre dataset des fruits ci-haut, nous pouvons appliquer l'algorithme de perceptron en ceci pour le cas du fruit 1 :
Observation du fruit 1 :
$\vec{x} = fruit1 = (1,0,0)$ et $\vec{W} = (0,0n0)$ sont les poids correspondant à chaque variable d'entrée x_i.
Le biais est initialisé ici à 1 et vecteur de poids de connexion à 0, car l'algorithme d'entrainement nécessite l'initialisation aléatoire, au début, du vecteur de poids de connexion [3]. Signalons que la valeur du vecteur de poids de connexion n'est pas nécessairement 0, les poids sont initialisés avec des valeurs aléatoires tirées de l'intervalle entre $[-1,+1]$ [10].

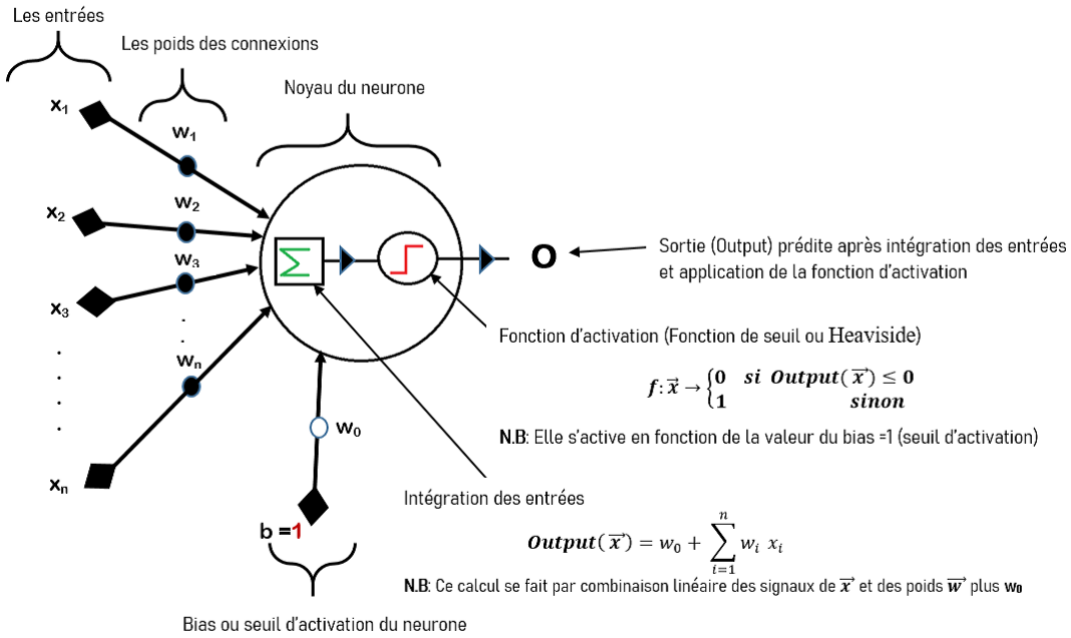


FIGURE 2.14 – Représentation du processus d'activation d'un neurone formel

Afin d'entrainer notre modèle pour l'observation du fruit, nous devons préciser que l'entrainement se fait en suivant cet algorithme :

1. Insérer les valeurs d'entrée $(\vec{x})$ d'une observation.
2. Initialiser aléatoirement les poids $(\vec{W})$. Calculer la combinaison linéair $(\vec{x})$et $(\vec{W})$ et l'observation plus W_0.
3. Faire vérifier si la valeur de sortie $(\hat{y})$ après l'activation marche avec le $target(y)$
   - Calculer l'erreur de prédiction pour l'observation
   - Mettre à jour les poids de connexion
4. Jusqu'à convergence entre y et $\hat{y}$.

Sur ce, la combinaison linéaire output $\vec{x} = W_0 + \sum i = 1nW_i.x_i = 0 + (1*0) + (1*0) + (0*0) = 0$. C'est ce 0 qui est dans le noyau du perceptron ci-dessous (Cf. Fig. 2). Si nous regardons la valeur d'output de la Fig. 2, nous constatons qu'elle vaut 0.
En fait, étant donné que le résultat de output $\vec{x}$ qui est égal à 0 est inférieur à la valeur du biais qui vaut 1.
C'est ainsi que selon la règle d'activation (Cf. Fig.1), le output $\hat{y}$ vaut 0.

Nous constatons que $\hat{y}$ ne correspond pas à y. En effet, la valeur de y pour le fruit 1 vaut 1, Cf. Il y a donc erreur. Pour calculer cette erreur, dans le cadre de neurone formel, on utilise la formule $E = y - \hat{y}$. Sur ce, $E = 1 - 0 = 1$.
Ce dernier indique réellement qu'il y a erreur de prédiction. Il faut donc ajuster les poids de connexion afin de permettre au perceptron de ne pas commettre l'erreur de classification de ce premier fruit. En ce sens, on applique la règle suivante d'ajustement :

$$\Delta W_j = \eta(y - \hat{y})x_j$$

poids ajusté sera calculé comme suit :

$$W_j^+ = W_j + \Delta W_j$$

Avec :
$y - \hat{y}$: Erreur de prédiction. C'est elle qui détermine s'il faut corriger ou non les paramètres.
$\hat{y}$: Valeur de sortie prédite par le modèle.

Features pour fruit 1	x_j	$Erreur = y - \hat{y} = 1$	η	$\Delta W_j = \eta(Erreur)x_j$	Wj	$W_j^+ = W_j + \Delta W_j$
Goût	1	1	0.25	0.25	0	0.25
Graine	1	1	0.25	0.25	0	0.25
Chair	0	1	0.25	0.	0	0

TABLE 2.3 – Résultat des poids ajustés dans le premier entrainement

y : Target cible. Pour le fruit 1, c'est 1.
x_j :c'est la force du signal. En fait, c'est la variable se trouvant dans l'échelle du poids que l'on veut ajuster. Dans notre cas, nous avons 1, 1, 0.
η :Taux d'apprentissage qui détermine l'amplitude de l'apprentissage. Le choix de ce taux intrigue souvent. En effet, quand c'est trop petit, il y aura la lenteur de convergence et quand c'est trop grand, Il y a une sorte d'oscillation [28]. En général, le taux d'apprentissage est choisi entre 0.05 et 0.15. Dans le cas de notre exemple, nous avons choisi 0.25.
W_j :c'est le poids précédent.
W_j^+ :le poids ajusté.

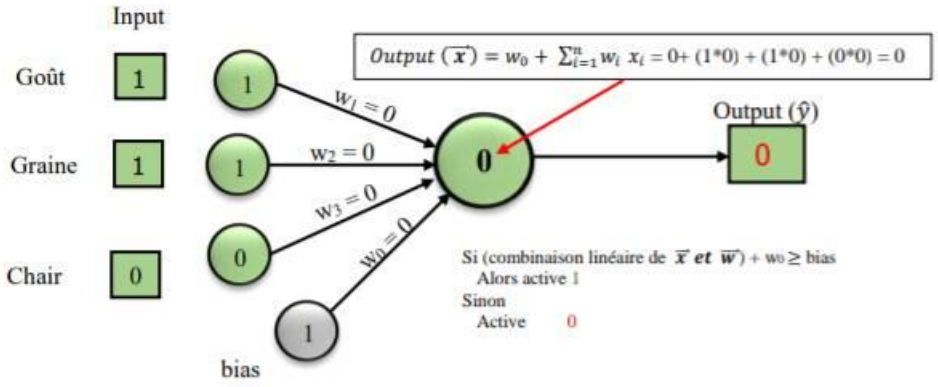


FIGURE 2.15 – Entrainement du perceptron avec l'observation du fruit

En remplaçant les nouvelles valeurs des poids (W_j^+) dans notre algorithme, nous obtenons le résultat suivant :

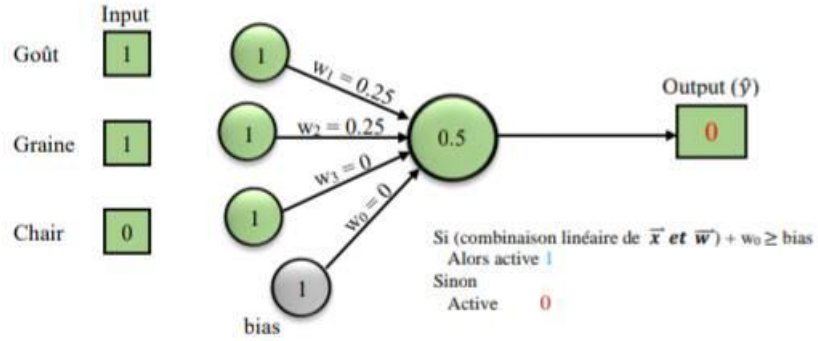


FIGURE 2.16 – Résultat du premier entrainement du perceptron avec l'observation du fruit 1

Le résultat de Fig. 3 montre que le perceptron a encore activé 0 car 0.5 est inférieur au biais. D'où, la continuation de l'entrainement du modèle jusqu'à ce qu'il apprenne et prédise le vrai résultat correspondant au target du fruit 1.

Features pour fruit 1	x_j	$Erreur = y - \hat{y} = 1$	η	$\Delta W_j = \eta(Erreur)x_j$	Wj	$W_j^+ = W_j + \Delta W_j$
Goût	1	1	0.25	0.25	0.25	0.5
Graine	1	1	0.25	0.25	0.25	0.5
Chair	0	1	0.25	0.	0	0

TABLE 2.4 – Résultat des poids ajustés dans le deuxième entrainement

On constate dans le diagramme ci-dessous que notre modèle a appris pour cette observation car la sortie $\hat{y}$ correspond au target y cible pour l'observation du fruit 1. On peut ainsi entrainer, avec le même processus, les autres observations des fruits.

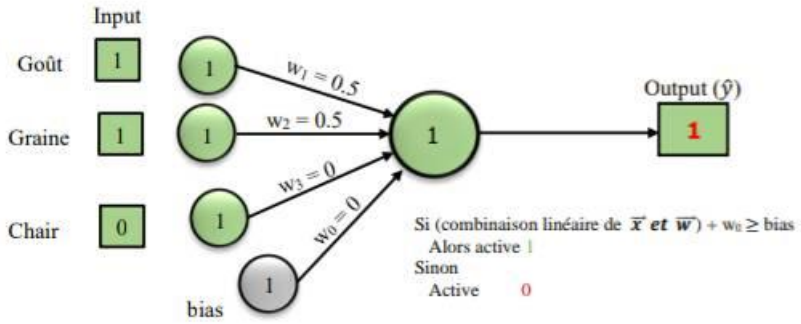


FIGURE 2.17 – Arrêt de l'apprentissage pour l'observation du fruit 1

En substance, nous venons là d'utiliser la fonction de seuil ou de Heaviside pour pouvoir prédire une étiquette binaire (soit 1, soit 0). En fait, le perceptron dans sa dimension unitaire est un classifieur linéaire qui classifie des données à partir d'une combinaison linéaire de ses entrées. De ce fait, il est incapable de classifier des données dans des classes non linéairement séparables. Ainsi, pour des cas de problèmes de classification multi-classe, on devrait modifier cette architecture que nous avons utilisé ci-haut de sorte à n'avoir non plus un seul neurone comme sortie mais plutôt C neurones dans la couche de sortie tel que C le nombre de différentes classes de sortie. En ce sens, les $j + 1$ neurones de la couche d'entrée seront tous connectés à chacun de neurone de sortie. Sur ce, on utilise la fonction softmax comme fonction d'activation [3].

2.8 Conclusion

Dans ce chapitre on a présenté le perceptron simple le plus ancien algorithme d'apprentissage qui permet de modifier les paramètres du neurone en s'appuyant sur le gradient de l'erreur entre la sortie réelle et la sortie souhaité,on a présenté sa défintion,architecture aussi les diférentes règles utilisées dans les algorithmes d'apprentissage,et on a terminé ce chapitre par un exemple d'entrainement de l'algorithme de perceptron.
Malheureusement,le perceptron pose de problème d'apprentissage car sa capacité de modélisation ne prend pas en compte les modèles non linéaires. Alors c'est un bon classifieur que pour les problèmes linéarement séparables.
nous faisons face aujourd'hui à des problèmes complexes non linéaires qui doivent être modélisés. D'où, l'importance de mise en place des perceptrons multicouche appelés aussi Multi-layer perceptron (MLP) afin de rendre l'apprentissage plus profond et ça ce qu'on va voir dans le prochain chapitre.

Chapitre 3

Perceptron multicouche

Sommaire

3.1 Introduction

Comme on a vue dans le chapitre précedant ,les premiers réseaux de neurones (perceptron simple) n'étaient pas capables de résoudre des problèmes non linéaires qui est un grand défaut dans le perceptron d'ou l'nteligence artificielle a été au point de mourrir heureusement tout a changé dans les années 80 lorsque Geoffrey Hinton l'un des chercheurs de deep learning a développé le Perceptron multicouche le premier véritable réseau de neurone artificielle .
Dans ce chapitre, nous présentons en détail le modèle de perceptron multicouche ,nous présentons d'abord l'architecture du réseau ainsi que la méthode d'apprentissage utilisé. et nous détaillons enfin l'algorithme de fonction de ce modèle.

Notre modèle se basait sur le réseau perceptron multicouche d'architecture feedforward,l'un des réseaux de neurones les plus utilisés pour des problèmes d'approximation, de classification et de prédiction, nous avons utilisé l'algorithme rétropropagation(backpropagation), cet algorithme est actuellement l'outil le plus utilisé dans le domaine de réseaux de neurones.

3.2 Perceptron Multicouche (PMC)

3.2.1 Définition d'un perceptron multicouche

Le Peceptron multicouche est une extension du perceptron simple, avec une ou plusieurs couches cachées entre l'entrée et la sortie. Chaque neurone dans une couche est connecté à tous les neurones de la couche précédente et de la couche suivante (excepté pour les couches d'entrée et de sortie) et il n'y a pas de connexions entre les cellules d'une même couche,les fonctions d'activation utilisées dans ce type de réseaux sont principalement les fonctions à seuil ou sigmoïdes,il peut résoudre des problèmes non-linéairement séparables donc il est particulièrement bien adapté au traitement des modèles complexes non linéaires et des problèmes logiques plus compliqués,il est capable d'approximer n'importe quelle fonction suffisamment régulière en utilisant une somme de fonctions sigmoïdes.
Il suit aussi un apprentissage supervisé selon la règle de correction de l'erreur(la régle de prapagation) qui consiste à déterminer comment la sortie du réseau varie en fonction des paramètres (W,b) présents en chaque couche,aprés grace à la méthode de descente de gradient,on peut mettre à jour les paramètres (W,b) de chaque couche de telle sorte à ce qu'ils minimisent l'erreur entre la sortie du modèle et la réponse attendue .
Donc le perceptron multicouche(PMC) s'agit d'un réseau à propagation directe (feedforward).

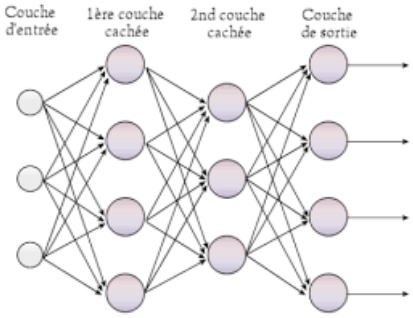


FIGURE 3.1 – perceptron multicouche à rétropropagation

Chaque couche est constituée d'un nombre variable de neurones, les neurones de la dernière couche (dite « de sortie ») étant les sorties du système global.
Un neurone de la couche n est connecté à tous les neurones de la couche n+1.
Le nombre de neurones dans la couche d'entrée est égal au nombre de variable du modèle. Pour la couche cachée, le nombre de neurones dépend généralement de l'application.
Il n'existe, malheureusement, pas de règles rigoureuses pour le choix du nombre de neurones cachées (ainsi pour le nombre de couches cachées).
Le Tableau montre l'impact du nombre de couches sur la région de décision pour un cas de classification.

3.2.2 Structure du réseau MLP

Une seule couche de neurones ne pouvant réaliser que des séparations linéaires, l'idée vient alors de rajouter des couches dites cachées pour réaliser un réseau de neurone multicouche. Dans une couche, les neurones ne sont pas connectés entre eux [12].
Un réseau à couches est une extension du célèbre perceptron avec une ou plusieurs couches intermédiaires appelées "cachées". Le perceptron multicouches (Multi Layered Perceptron - MLP) sont les réseaux de neurones les plus connus. Un perceptron est un réseau de neurones artificiel du type « feedforward », c'est-à-dire à propagation directe. Le schéma de la figure 3.2 montre un réseau à trois couches possédant trois entrées et une sortie. La première est celle des entrées (elle n'est cependant pas considérée comme couche neuronale par certains auteurs car elle est linéaire et ne fait que distribuer les variables d'entrées). La deuxième est dite couche cachée et constitue le cœur du réseau de neurones. La troisième, constituée ici par un seul neurone est la couche de sortie, on numérote les couches de l'entrée vers la sortie donc sur le schéma de gauche à droite, la couche cachée a trois neurones aussi on notera par la suite un tel réseau .
Nous pouvons remarquer sur la figure 3.2, des termes m 0x en entrée des neurones (le terme en exposant représente, non pas la fonction puissance, mais plutôt l'indice (m) de la couche du réseau de neurones).

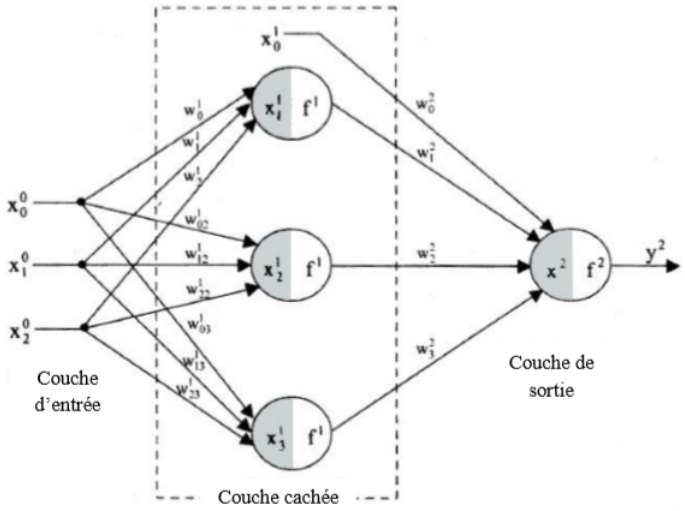


FIGURE 3.2 – Réseau de neurones de type perceptron à une couche cachée.

En fait, sur chaque neurone, en plus de ses entrées qui le lient avec les neurones précédents, on ajoute une entrée particulière que l'on appelle polarisation du neurone, elle correspond à un biais qui joue un rôle de translation du domaine d'activité du neurone. Sa valeur est donc liée à la fonction d'activation puisqu'elle permet le déplacement de cette fonction.

Afin de garder une notation généralisée, nous représentons ces biais comme le produit d'une entrée x_0^m par les poids W_{0j}^m . Nous fixons l'entrée x_0^m à l'unité, le poids porte alors l'information sur la polarisation du neurone.
L'un des problèmes de l'utilisation des réseaux multicouches (MLP) consiste dans le choix de son architecture. La détermination du nombre de couches nécessaires est fondamentale et à rendre minimale (pour des raisons évidentes de vitesse de calcul mais aussi de capacité de généralisation).

Le perceptron multicouche (MLP) est très utilisé en identification et en contrôle. Avec une couche cachée, il constitue un "approximateur universel". De récentes recherches montrent qu'il peut entrainer de manière à approximer n'importe quelle fonction entrées/sorties sous réserve de mettre suffisamment de neurones dans la couche cachée et d'utiliser des sigmoïdes pour les fonctions d'activation. Bien entendu, les théorèmes mathématiques ne démontrent pas qu'un réseau à une seule couche cachée est optimal.

Malheureusement, il n'existe pas de règle générale qui donne le nombre de neurones à retenir pour la couche cachée. Le résultat le plus important est certainement le théorème de HechtNielsen : Toute fonction continue peut être implémentée exactement comme un réseau de neurones à trois couches ayant n cellules en entrée, $2n+1$ cellule en couche cachée et m cellules de sortie. Il faut bien noter que ce théorème ne donne aucune indication quant au nombre de connexion (le réseau n'est pas toujours totalement connecté), et ne garantit pas que ce nombre de Couche cachée, Couche d'entrée et Couche de sortie Figure 3.2.

3.2.3 Prédire la valeur de sortie de la fonction SIGMOÏDE

3.2.3.1 Fonction d'activation

Il existe plusieurs types de fonctions d'activation. On parle de fonction d'activation du perceptron non lisse lorsque le taux d'erreur du modèle est une fonction discontinue des paramètres de poids. Avec ce type de fonctions, l'ajustement des poids optimaux en minimisant la fonction de perte est souvent difficile. D'où, l'application d'une fonction d'activation continue afin de résoudre ce problème. Dans ce sens, la fonction sigmoïde logistique est un choix approprié à cet effet car elle est capable d'apprendre les limites de décision non linéaires pour les espaces séparables non linéaires. En fait, les fonctions d'activation dans le PMC doivent être non linéaires, continuellement différenciables et non décroissantes de façon monotone [37].

En outre, il est souhaitable de choisir une fonction d'activation dont la dérivée serait facilement calculée. En faisant ainsi usage d'une fonction d'activation non linéaire, par exemple sigmoïde, le PMC est un approximateur de fonction universelle. C'est-à-dire qu'un réseau de perceptron multicouche à une seule couche cachée associée à un nombre suffisamment grand et fini de neurones peut approcher et apprendre toute fonction continue sur un domaine compact avec une précision arbitraire[21].

3.2.3.2 Nécessité de la fonction SIGMOÏDE

Lors de la rétro-propagation en PMC, l'objectif est de minimiser la fonction coût. Ainsi, les fonctions sigmoïde et tangente hyperbolique, calculent toutes deux leurs dérivés de manière très simple et efficace, ce qui justifie leur utilisation courante lors de l'optimisation du gradient dans un PMC. Bien que les fonctions ReLU sont utilisées dans des réseaux de neurones modernes et permettent un apprentissage plus rapide et plus efficace des réseaux

neuronaux profonds sur des données complexes et à haute dimension, en calculant la fonction f (x) = max (0, x) et en seuillant simplement la matrice d'entrée à zéro, en ne nécessitant pas de calcul coûteux [37],la fonction sigmoïde est le plus utilisé car elle est dérivable en tout point .

On utilise ainsi la dérivée de la fonction Sigmoïde pour calculer le coefficient d'erreur. Elle permet de calculer les erreurs en faisant une marche en arrière tout en parcourant toutes les couches en partant de la sortie à l'entrée pour voir comment l'erreur a été propagée sur le réseau neuronal afin d'ajuster les poids des connexions des neurones .

3.2.4 Avantages et inconvénients de réseaux multicouche

Les réseaux multicouches à rétropropagation du gradient sont des performants pour la classification non linéare ,la compression de données,l'approximation de fonction ...
cependant,ces réseaux présentent certains inconvénients comme :
le temps d'apprentissage peut etre long(de quelques minutes à plusieurs heures suivant la complexité du problème traité).
il n'existe pas de méthodologie formelle pour la conception et la construction de ce type de réseau.
les choix des paramètres caractéristiques du réseau (nombre de couches,nombre de neurones par couche, pas d'apprentissage, fonction d'activation) se font alors par tatonnement pour obtenir les performances recherchées.

Malgré ces difficultés,ils possèdent des capacités remarquables d'apprentissage et de reconnaissance des formes.
Pour cela ,ils sont particulièrement bien adaptés à la résolution de problèmes de diagnostic industriel et médical,comme le montrent les exemples d'utilisation pour la classification de signaux en médcine,et pour la diagnostic des machines tournantes.

3.3 Apprentissage du PMC

Dans le cas de l'apprentissage supervisé, le corpus d'entrainement est composé de N entrées X pour lesquelles on connait les sorties désirées Y. Ainsi le vecteur X est fourni en entrée du R.N et ce dernier doit estimer en sortie les valeurs désirées Y connues par l'intermédiaire de son vecteur S résultant d'une série de calcul interne figure 3.3. Ce type d'apprentissage est une métaphore du « professeur » qui donne des « exemples » et fournit une correction par le calcul de l'« erreur de prédiction » permettant au réseau de « corriger » itérativement la valeur de ses poids. Ainsi on cherche à minimiser l'erreur $E = S - Y$.
L'apprentissage supervisé des $R.N$ est donc formulé comme un problème d'optimisation paramétrique dans lequel on cherche les valeurs optimales des poids des connexions du réseau pour un comportement précis fourni par les exemples du corpus d'apprentissage.

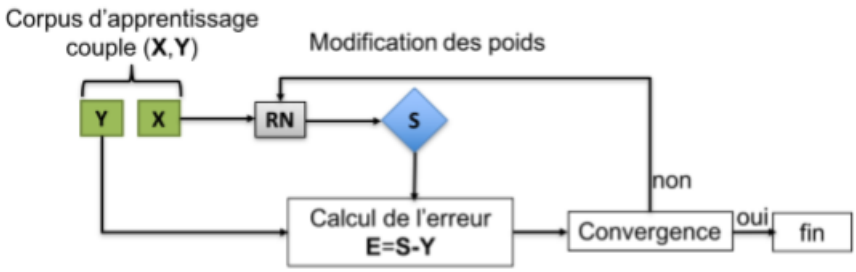


FIGURE 3.3 – Schéma explicatif de l'apprentissage supervisé

3.3.1 Apprentissage supervisé par perceptron multicouche

Parmi les réseaux qui fonctionnent avec un apprentissage supervisé, nous pouvons citer le perceptron multicouche

Le perceptron multicouche ("multilayer perceptron") est une généralisation de ces modèles :
- En régression il permet de traiter les cas non linéaires de régression.
- En classification, il permet de d´eterminer des fonctions de décision non linéaires permettant de résoudre le problème "XOR" précédent par exemple.

La fonction objectif est de deux types selon le problème traité :

- Pour **la régression**, on utilise la somme des carrés des résidus :

$$err(g) = \sum_{i=1}^{n} (y_i - g(x_i))^2$$

- Pour **la catégorisation**, on utilise la cross-entropie d´efinie par :

$$err(g) = \sum_{i=1}^{n} \sum_{l=1}^{q} -y_{il} \log(g_l(x_i))$$

3.3.1.1 Cas de la régression non linéaire

-Un seul neurone dans la 2 ème couche c-à-d $q = 1$ Fonction objectif :

$$err(g) = \sum_{i=1}^{n} (y_i g(x_i))^2$$

-Fonction d'activation de la 1'ere couche.
la fonction sigmoid :

$$h_1(s_k(x_i)) = 1/(1 + exp(a_k^T x_i)) = z_k$$

- Remarque : on a la propriété suivante :

$$\frac{\partial sigmoid(x)}{\partial x} = sigmoid(x)(1 sigmoid(x))$$

- Fonction d'activation de la 2 ème couche, la fonction identité :

$$h_2(s(z)) = b^T z = g$$

- Erreur associée à l'objet X_i :

$$erri = (y_i b^T z)^2$$

Dérivées partielles par rapport à b (2ème couche) :

$$\frac{\partial err_i}{\partial b_k} = \underbrace{2(y_i - b^T z)}_{\partial err_i/\partial g} \underbrace{z_k}_{\partial g/\partial b_k}$$

- Dérivées partielles par rapport à a_k (1 ère couche) :

$$\frac{\partial err_i}{\partial a_{kj}} = \underbrace{2(y_i - b^T z)}_{\partial err_i/\partial g} \underbrace{b_k}_{\partial g/\partial z_k} \underbrace{z_k(1 - z_k)}_{\partial z_k/\partial s_k} \underbrace{x_{ij}}_{\partial s_k/\partial a_{kj}}$$

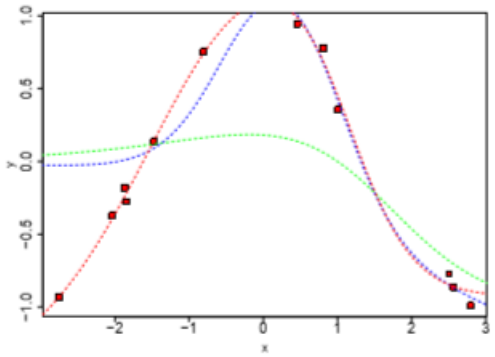


FIGURE 3.4 – Résultats (version on-line) avec m = 3 après 100, 200 et 400 itérations

3.3.1.2 Cas de la régression catégorisation avec q classes

- q neurones dans la 2 ème couche.
- Fonction objectif :

$$err(g) = \sum_{i=1}^{n} \sum_{l=1}^{q} -y_{il} \log(g_l(x_i))$$

- Fonction d'activation de la 1'ere couche,
la fonction sigmoid :

$$h_1(s_k(x)) = \frac{1}{1 + exp(-a_k^T z)} = z_k$$

- Fonction d'activation de la 2 ème couche,
la fonction softmax :

$$h_2(s_l(z)) = \frac{exp(b_l^T z)}{\sum_{l=1}^{q} exp(b_l^T z)} = g_l$$

Remarque : on a aussi la propriété suivante :

$$\frac{\partial softmax(x)}{\partial x} = softmax(x)(1 - softmax(x))$$

- Erreur associée à l'objet X_i :

$$err_i = \sum_{l=1}^{g} -y_{il} \log \frac{exp(b_l^T z)}{sum_{l=1}^{q} exp(b_l^T z)}$$

- Dérivées partielles par rapport à b (2 ème couche) :

$$\frac{\partial err_i}{\partial b_{l,k}} = \sum_{l'=1}^{q} \frac{\partial err_i}{\partial g_{l'}} \frac{\partial g_{l'}}{\partial b_{l,k}} = \sum_{l'=1}^{q} \frac{\partial err_i}{\partial g_{l'}} \frac{\partial g_{l'}}{\partial s_l} \frac{\partial s_l}{\partial b_{l,k}}$$

- Puis en regroupant le tout, on obtient finalement :

$$\frac{\partial err_i}{\partial b_{l,k}} = -(y_{il} - g_l) z_k$$

- Rappelons que :

$$z_k = h_1(s_k(x)) = sigmoid(s_k(x)) = \frac{1}{1 + exp(a_k^T x)}$$

- Dérivées partielles par rapport à a_k (1 ère couche) :

$$\frac{\partial err_i}{\partial a_{kj}} = \sum_{l'=1}^{q} \frac{\partial err_i}{\partial g_{l'}} \sum_{l''=1}^{q} \frac{\partial g_{l'}}{\partial s_{l''}} \frac{\partial s_{l''}}{\partial z_k} \frac{\partial z_k}{\partial s_k} \frac{\partial s_k}{\partial a_{k,i}}$$

- On obtient le résultat suivant :

$$\frac{\partial err_i}{\partial a_{kj}} = \sum_{l''=1}^{q} \underbrace{\left(\sum_{l'=1}^{q} \frac{\partial err_i}{\partial g_{l'}} \frac{\partial g_{l'}}{\partial s'_{l'}} \right)}_{-(y_{il''} - g_{l''})} \frac{\partial s_{l''}}{\partial z_k} \frac{\partial z_k}{\partial s_k} \frac{\partial s_k}{\partial a_{k,j}}$$

$$= \sum_{l''=1}^{q} -(y_{il''} - g_{l''}) \underbrace{b_{l'',k}}_{\partial s_{l''}/\partial z_k} \underbrace{z_k(1 - z_k)}_{\partial z_k/\partial s_k} \underbrace{x_j}_{\partial s_k/\partial a_{k,j}}$$

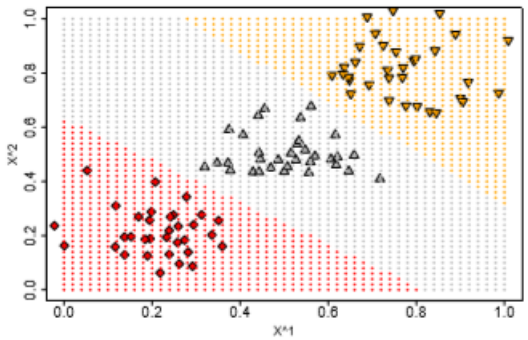


FIGURE 3.5 – Résultats (version on-line) avec m = 5 et q = 3 après 100 itérations

Pour effectuer l'apprentissage, trois éléments sont essentiels. La Force du signal,l'Erreur et le Taux d'apprentissage. Toutefois, il est important de savoir, certaines notions élémentaires interviennent lors de l'apprentissage d'un modèle de perceptron multicouche. Rappelons que l'algorithme de rétro-propagation de gradient est l'un des algorithmes les plus répandus dans les réseaux de neurones de type feedforward [17].

3.3.2 Régression logistique

La régression logistique est un algorithme supervisé de classification. La régression logistique est un modèle de classification linéaire qui est le pendant de la régression linéaire, quand Y ne

doit prendre que deux valeurs possibles (0 ou 1). Comme le modèle est linéaire.
La fonction qui remplit le mieux ces conditions est la fonction sigmoïde. Le résultat obtenu par la fonction sigmoid est interprété comme la probabilité que l'observation X soit d'un label (étiquette) 1.
Le principe de la régression logistique est simple il consiste à découper le problème de classification multi-classes en une multitude de problèmes de classification binaires.

3.4 PMC dans le cadre de la classification supervisée

3.4.1 Perceptron multicouche et classification

Le Perceptron MultiCouche (PMC) est un des réseaux de neurones les plus utilisés actuellement, pour la classification supervisée notamment .
On fait dans un premier temps une synthèse des résultats acquis en matière de capacités de représentation dont jouit potentiellement l'architecture PMC, indépendamment de tout algorithme d'apprentissage. Puis on montre pourquoi la minimisation d'une erreur quadratique sur la base d'apprentissage semble être un critère mal approprié, bien que certaines propriétés asymptotiques soient aussi exhibées .
Dans un second temps, l'approche bayesienne est analysée lorsqu'on ne dispose que d'une base d'apprentissage de taille finie. A l'aide de certains estimateurs de densité, dont les propriétés remarquables sont résumées, il est possible de construire un réseau de neurones stratifié réalisant la classification bayesienne.
Cette technique de discrimination directe semble être supérieure au PMC classique à tous points de vue en dépit de la similarité des architectures [7].

Pour les fonctions non linéaurement séparables, il faut établir un réseau de neurone multicouche.

3.4.2 Problème de « XOR »

Le problème **XOR** (**OU exclusif**) montre comment l'introduction de la fonction non linéaire de sortie des neurones formels et de l'architecture en couches permet d'obtenir des surfaces séparatrices non linéaires. Plaçons nous dans $IR2$, avec quatre points d'apprentissage situés au quatre coins du carré unité. Chaque paire de points opposés en diagonale forme une classe, Les deux classes ne sont pas linéairement séparables dans le plan $\{x1, x2\}$ figure 1.6. Les exemples ont donc la forme suivante :

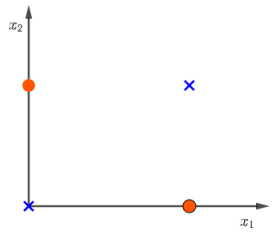


FIGURE 3.6 – Le problème XOR.

Dans ce problème on va donc utiliser 2 neurones dans la couche d'entrée, 1 neurone dans la couche de sortie et 2 neurones dans la couche cachée.

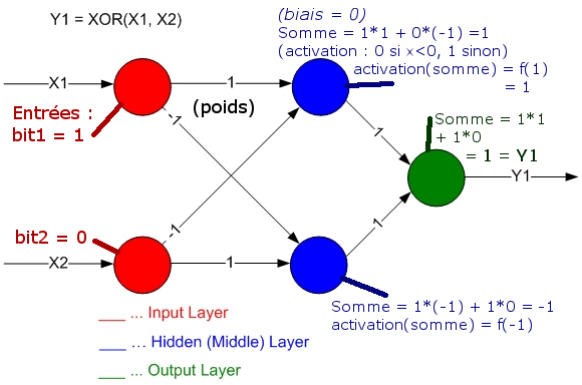


FIGURE 3.7 – Perceptron du problème XOR.

Pour la porte XOR, la table de vérité sera la suivante :

Entrées		Sorties
x_1	x_2	$x_1 XOR x_2$
0	0	0
0	1	1
1	0	1
1	1	0

On voit sur ce tableau que les points d'entrée ont été re-décrits dans le plan $y1, y2$. Les deux croix qui en bleu ont conservés leurs coordonnées $(0, 0)$ et $(1, 1)$. Mais les deux points qui en rouge sont maintenant confondus en $(1, 0)$.
Il devient ainsi possible de les séparer des points par une séparatrice linéaire.

Une combinaison de séparateurs linéaires permet de produire un séparateur global non-linéaire (Rumelhart, 1986).

Une interprétation est possible en logique booléenne : au lieu de considérer les valeurs 0 et 1 comme des coordonnées numériques, prenons-les comme des valeurs logiques. Dans ce cas, on peut interpréter les sorties intermédiaires et la sortie finale comme des fonctions booléennes sur les entrées. Le réseau réalise la fonction XOR (le OU exclusif), qui vaut 0 pour les deux points et 1 pour les deux points . Ceci est rendu possible par les non-linéarités du système de calcul (les sorties des unités sont bornées dans]0,1[) la figure 3.8.

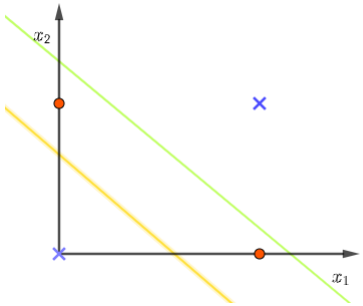


FIGURE 3.8 – Résoudre le problème XOR avec un PMC à une couche cachée.

Une des façons de résoudre le problème XOR avec un PMC à une couche cachée. La zone affectée à la classe 1 (les poins rouge)est le « couloir » compris entre les deux droites en pointillé, celle affectée à la classe 2 (les croix bleues) est à l'extérieur. La première droite répond à l'équation $x1 + x2 - 0.5 = 0$ et réalise un OU logique. La seconde répond à l'équation $x1 + x2 - 0.5 = 0$ et réalise un ET logique. Elles sont réalisées par la première couche du réseau. La seconde couche combine les deux décisions linéaires en une décision non linéaire.

Le PMC résoudre éfficacement le problème XOR en visualisant les points de données dans plusieurs dimensions et en construisant ainsi une équation à n variables pour s'adpter aux valeurs de sortie.

3.5 Algorithme d'apprentissage pour les réseaux PMC

3.5.1 L'apprentissage par retropropagation d'erreur

Dans le perceptron multicouche à rétropropagation, les neurones d'une couche sont reliés à la totalité des neurones des couches adjacentes.
Ces liaisons sont soumises à un coefficient altérant l'effet de l'information sur le neurone de destination.
Ainsi, le poids de chacune de ces liaisons est l'élément clef du fonctionnement du réseau.
la mise à jour des poids applicables à chacune des connexions inter-neuronales est déterminée par l'algorithme de rétropropagation.

Un réseau de neurone multicouche suit un apprentissage supervisé Multicouche direct indique que le résultat (la sortie) ne rétroagit pas, il progresse de l'entrée vers la sortie à sens unique. La fonction d'activation utilisée est le sigmoïde. Les réseaux neurologiques avec la fonction d'activation de type sigmoïde s'appellent Multi-Layer Perceptron (MLP).
Le gradient permet de diriger la solution vers une minimisation de la fonction erreur.

L'objectif est d'atteindre le minimum global ; dans ce cas la fonction erreur est extrêmement complexe ; il y a alors un risque pour que le réseau converge vers un minimum local.
Pour contrôler le gradient et éviter des variations brusques, l'expérience a montré qu'il faut faire quelques tests (5 à 10 fois) au début et prendre la solution qui présente une variation stable du gradient.
Les poids initiaux sont aléatoires. Ceci assure que la solution débute en des endroits différents dans l'espace d'erreurs et ira dans différentes directions.

Apprentissage :

La procédure d'apprentissage repose sur l'idée de propager vers les couches internes ,l'erreur commise en sortie pour modifier les poids synaptique.
C'est un apprentissage supervisé.Pour cela,on dispose d'un ensemble d'exemples(base d'apprentissage),constitué de couple (entrée,sortie désirée).
Lors de l'apprentissage ,on présente les exemples au réseau qui calcule les sorties correspondantes.Ces calculs s'effectuent de proche en proche depuis la couche d'entrée vers la couche de sortie(phase de relaxation ou de propagation avant).
L'erreur entre la sortie réelle et la sortie désirée est calculée par une somme quadratique des erreurs sur chaque neurone de sortie,cette erreur est ensuite rétropropagée à travers le réseau

donnant lieu à une modification des poids synaptiques.

Pour la couche de sortie, on peut appliquer l'apprentissage du perceptron, mais comment modifier les poids pour les connexions qui ne sont pas en relation avec un neurone de sortie ?
Le problème est l'obtention d'une estimation de la valeur désirée pour chaque neurone de la couche cachée.
Ils utilisent pour modifier leurs poids, un algorithme de rétropropagation du gradient (backpropagation),il s'agit toujours de minimiser l'erreur quadratique.

3.5.1.1 Apprentissage par rétro-propagation

La phase de développement d'un réseau à rétro-propagation du gradient se fait par un algorithme à apprentissage supervisé. Cet algorithme vise à faire des regroupements entre les matrices de donnée d'entrée et les matrices donnée de sortie souhaités [20].
En fait, l'idée de base de cet algorithme est que l'on veut minimiser le critère d'erreur quadratique en fonctionne des poids de connexions de réseau.

$$E = \frac{1}{2}\sum(y_d - y_i)^2$$

Donc, l'erreur définie par les écarts entre les outputs désirés et les outputs obtenus doit être minimisée par rapport aux poids de connexions.
Les réseaux de neurones multicouches à rétro propagation du gradient, par leurs capacités remarquables d'apprentissage et de reconnaissance des formes constituent de manière incontestable l'outil privilégié du diagnostic industriel à base de signatures externes. C'est la raison pour laquelle, il est important de modifier les conditions initiales du réseau (poids des connexions nombre des cellules et des couches) pour vérifier la robustesse des résultats, pour la résolution de problèmes de classification avec rejet technique indispensable en cas d'évolution lent des signatures de réseaux de neurones [4].
Les réseaux de neurones sont bien adaptés à la résolution des problèmes de diagnostic, utilisant la classification automatique des signaux et des formes.
Dans ce contexte on distingue plusieurs applications des réseaux de neurones pour le diagnostic des défaillances et en particulier, pour le diagnostic des pannes des machines industrielles.

3.6 Algorithme de rétro-propagation du gradient (RPG)

3.6.1 Historique

Des chercheurs de différents domaines ont développé plusieurs variantes de cet algorithme. La première formule de la version actuelle a été proposée par Werbos en 1974. Il a été appliqué aux réseaux multicouches par Rumlethart en 1986.[4]

L'algorithme de rétro propagation du gradient (PG) est à la base des premiers succés des réseaux de neurones. Sa mise en applicationa permis au domaine du connexionnisme de sortir de la période de silence qui a régné après la sortie du livre "Perceptron" de Minsky et Papert (1969). IL figure aujourd'hui parmi les algorithmes d'apprentissage les plus utilisés. Il a été appliqué avec succés a une grande variété de problème [22].

3.6.2 Définition de l'algorithme

Cette methode est une technique de calcul des dérivées qui peut être appliquée à n'importe qu'elle structure de fonctions dérivables. Cette méthode est basée mathématiquement sur l'algorithme de descente du gradient et utilise les règles de dérivation des fonctions dérivables. De telles façons l'erreur commise en sortie du réseau sera rétropropagée vers les couches cachées d'où le nom de rétropropagation.

La technique de rétropropagation du gradient consiste à corriger les erreurs selon l'importance de la contribution de chaque élément aux erreurs. Dans le cas des réseaux de neurones, les poids synaptiques qui contribuent plus à une erreur seront modifiés de manière plus importante que les poids qui provoquent une erreur marginale.

Ce principe fonde les méthodes de type algorithme du gradient, qui sont utilisées dans des réseaux de neurones multicouches comme les perceptrons multicouches. L'algorithme du gradient a pour but de converger de manière itérative vers une configuration optimale des poids synaptiques. Cet état peut être un minimum local de la fonction, ou, idéalement, le minimum global de cette fonction (dite fonction de coût). La figure 1.4 montre une illustration du principe.

Normalement, la fonction de coût est non linéaire au regard des poids synaptiques. Elle dispose également d'une borne inférieure et moyennant quelques précautions lors de l'apprentissage, les procédures d'optimisation finissent par aboutir à une configuration stable au sein du réseau de neurones.[5]

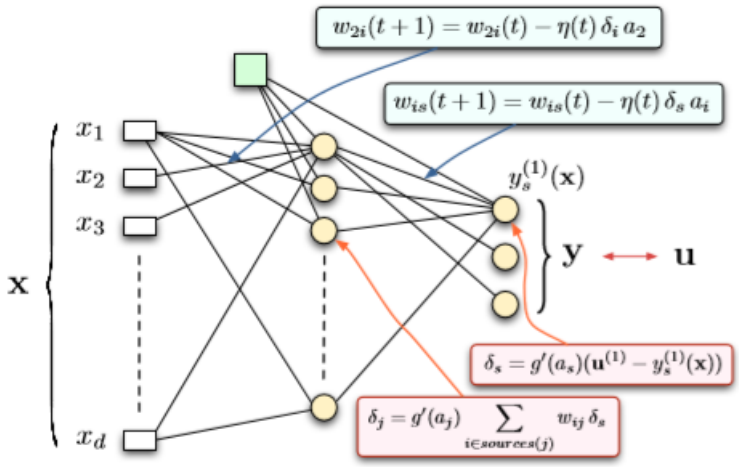


FIGURE 3.9 – Schéma du modèle de la rétropropagation de l'erreur. La modification à apporter aux poids entre la couche source(j) et le neurone formel j ne peut être calculée que si on connaît déjà la modification qu'il faut apporter aux poids w(j,k) entre j et les éléments de dest(j).

3.6.3 Fonctionnement de l'algorithme

L'algorithme de la rétro-propagation altère les coefficients synaptiques (w_i) du réseau dans le sens inverse du gradient du critère d'erreur, en utilisant seulement les données d'entrée/sortie ; A chaque itération, on retire un exemple d'apprentissage (x_i, yi) et on calcule une nouvelle

estimation du poids synaptique wi. Cette itération consiste en deux phases [11] :

-**Propagation :** à chaque itération, un élément de l'ensemble d'apprentissage est introduit à travers la couche d'entrée. L'évaluation des sorties du réseau se fait couche par couche, de l'entrée vers la sortie .

-**Rétro-propagation :** cette étape est similaire à la précédente. Cependant, les calculs s'effectuent dans le sens inverse (de la sortie vers l'entrée). A la sortie du réseau, on forme le critère de performance E en fonction de la sortie réelle de système et sa valeur désirée. Puis, on évalue le gradient de E par rapport aux différents poids en commençant par la couche de sortie et en remontant vers la couche d'entrée. La figure (3.10) montre le processus de rétropropagation.

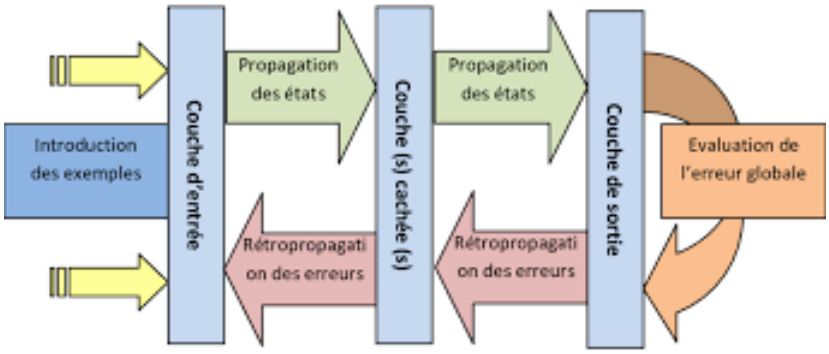


FIGURE 3.10 – Apprentissage des réseaux de neurone par l'algorithme de rétropropagation

La propagation directe

1. La forme X_k est présentée à l'entrée du réseau
2. Calcul des $hj(k), j = 1, 2, .., J$, et $y_m(k), m = 1, 2, .., M$
3. Calcul des $\delta m(k), m = 1, 2, .., M$

La rétropropagation

1. Rétropropagation et calcul de j, j=1,2, . . . , J
2. Actualisation des poids Wmj
3. Actualisation des poids Vjn

La plupart des algorithmes d'apprentissages permettent d'obtenir les poids par minimisation d'une fonction de coût dérivable. La méthode la plus simple pour procéder à une minimisation est la méthode du gradient.
La mise à jour de l'ensemble des poids d'une couche nécessite la connaissance des erreurs associées à chaque neurone de la couche suivante. On appliquera l'algorithme d'ajustement des poids en partant de la dernière couche (pour laquelle les erreurs sont connues) vers la première, d'où l'appellation de cet algorithme : Algorithme de rétropropagation du gradient de l'erreur « backpropagation Algorithme ».

Algorithme 14 : Apprentissage du perceptron multicouche

début
  tant que *l'apprentissage n'a pas convergé* **faire**
    tirer au hasard un point d'apprentissage
    pour *chaque couche, en partant de celle du haut* **faire**
      pour *chaque neurone formel de cette couche* **faire**
        calculer δ_j
        pour *chaque connexion* $w(i,j)$ *menant au neurone formel* j **faire**
          calculer $\Delta w(i,j) = \alpha\, \delta_j\, y_i$
        fin pour
      fin pour
    fin pour
    pour *chaque connexion* $w(i,j)$ **faire**
      $w(i,j) \leftarrow w(i,j) + \Delta w(i,j)$
    fin pour
  fin tant que
fin

Un bon fonctionnement du perceptron multicouches avec l'algorithme de rétropropagation comme classifieur des données nécessite :
- Une configuration adéquate du réseau.
- L'algorithme doit converger vers un minimum global.
- Un bon choix des données d'apprentissage.
- Un apprentissage souhaité.

3.6.4 Cas de PMC à une couche cachée

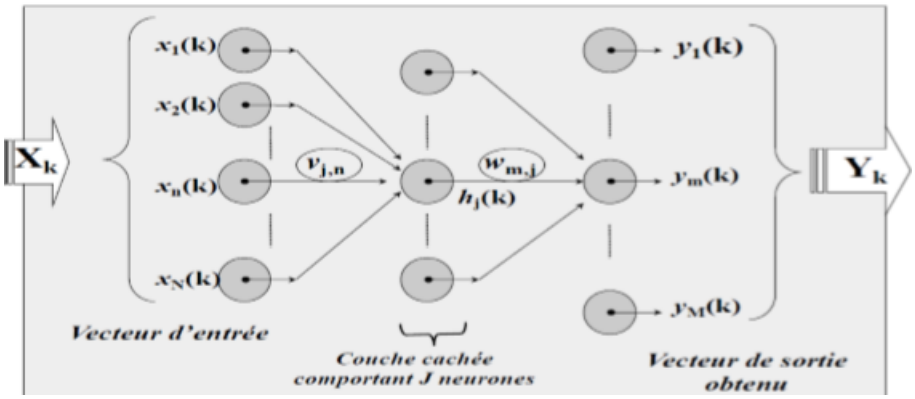


FIGURE 3.11 – Schéma de PMC à une couche cachée

Fonction du coût : Erreur quadratique instantanée

$$E_k = \frac{1}{2}\sum_{m=1}^{M}(d_m(k) - y_m(k))^2$$

Pour les poids synaptiques W_{mj} **:**

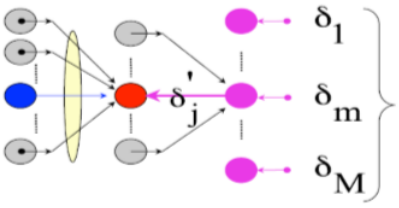
$$\Delta w_{mj} = \eta h_j(k)\delta_m(k)$$
$$\delta_m(k) = (d_m(k) - y_m(k))f'(net_m(k))$$

$$\delta_m(k) = (d_m(k) - y_m(k))f'\left(\sum_{j=0}^{J} w_{mj}h_j(k)\right)$$

Pour les poids synaptiques V_{jn} :

$$\Delta V_{jn} = \eta x_n(k)\delta_j(k)$$

$$\delta_j(k) = f'(net_j(k))\sum_{m=1}^{M}\delta_m(k)w_{mj}$$



La propagation directe
- calcul des Z_j
calculer l'activation Z_{inj} et la sortie Z_j de chaque neurone caché :
$Z_{inj} = v_{0j} + somme(x_i \times v_{ij})$;
$Z_j = f(Z_{inj})$;

- Calcul des Y_m
calculer l'activation Y_{inm} et la sortie Y_m de chaque neurone de sortie.
$Y_{inm} = W_{0m} + somme(Z_j \times W_{jm})$;
$Y_m = f(Y_{inm})$;

La rétropropagation
Adaptation des poids synaptiques W_{jk} :

$$\Delta W_{mj} = \eta z_j \delta_m$$

$$\delta_m = (t_m - y_m)f'(y_i n_m) = (t_k - y_k)f'(W_{m0} + \sum_{j=1}^{J} z_j W_{mj})$$

Adaptation des poids synaptiques V_{ij} :

$$\Delta V_{jn} = \eta x_n \delta_j$$

$$\delta_j = \delta_i n_j f'(z_i n_j) = \left(\sum_{m=1}^{M}\delta_m W_{m,j}\right) f'\left(V_{j0} + \sum_{n=1}^{N} X_n V_{jn}\right)$$

3.6.4.1 L'algorithme

1. **début** initialisation des poids du PMC
2. **faire propagation directe :** pour chaque vecteur de la base d'apprentissage :
   a) Affecter $X_k = x_n(n = 1...N)$; envoyer aux neurones cachés

b) Chaque neurone caché calcule son entrée nette :

$$Z\_in_j = V_{j0} + \sum_{n=1}^{N} X_n V_{jn}$$

c) Appliquer sa fonction de sortie :

$$Z_j = f(Z\_in_j)$$

d) Chaque neurone de sortie calcule son entrée :

$$y\_in_m = W_{m0} + \sum_{j=1}^{j} Z_j W_{mj}$$

e) Appliquer sa fonction de sortie :

$$y_m = f(y\_in_m)$$

rétropropagation : chaque neurone de sortie reçoit son étiquette t_m

f) Calculer les signaux d'erreur de sortie :

$$\delta_m = (t_m - y_m) f'(y_i n_m)$$

g) Calculer les modifications de poids - couche de sortie :

$$\Delta W_{mj} = \eta z_j \delta_m$$

h) Rétropropager les signaux d'erreur δ_m vers la couche cachée précédente

i) Chaque neurone caché calcule sa contribution à partir de la rétropropagation des signaux d'erreur de sortie

$$\delta\_in_j = \sum_{m=1}^{M} \delta_m w_{mj}$$

j) Chaque neurone caché calcule son signal d'erreur

$$\delta_j = \delta\_in_j f'(z\_in_j)$$

k) Calculer les modifications de poids - couche cachée

$$\Delta v_{jn} = \eta x_n \delta_j$$

l) Mise à jour des poids et polarisation

$$v_{jn}(nouveau) \longleftarrow \Delta v_{jn} + v_{jn}(ancien)$$

$$w_{mj}(nouveau) \longleftarrow \Delta w_{mj} + w_{mj}(ancien)$$

jusqu'à critère d'arrêt satisfait

3. **retourner** les poids du réseau PMC
4. **fin**

3.6.5 Cas de PMC à plusieurs couches cachés

Les poids dans le réseau de neurones sont au préalable initialisés avec des valeurs aléatoires. On considère ensuite un ensemble de données qui vont servir à l'apprentissage. Chaque échantillon possède ses valeurs cibles qui sont celles que le réseau de neurones doit à terme prédire lorsqu'on lui présente le même échantillon.

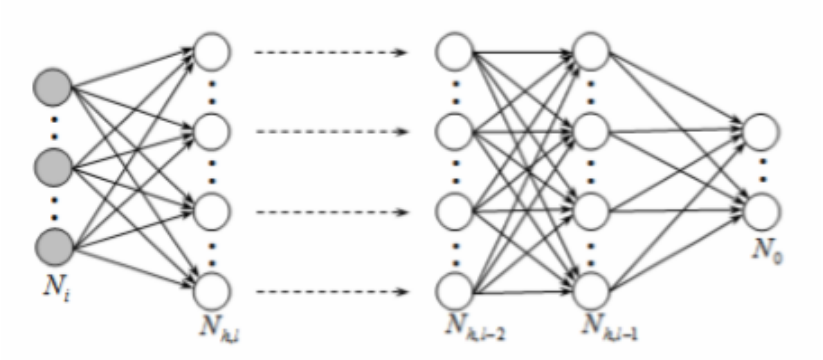


FIGURE 3.12 – Structure de réseau PMC à plusieurs couches cachés

Soient un échantillon $\overrightarrow{x}$ que l'on présente à l'entrée du réseau de neurones et $\overrightarrow{t}$ la sortie recherchée pour cet échantillon.
On propage le signal en avant dans les couches du réseau de neurones : $x_k^{(n-1)} \longmapsto x_j^{(n)}$, avec n le numéro de la couche.
La propagation vers l'avant se calcule à l'aide de la fonction d'activation g, de la fonction d'agrégation h (souvent un produit scalaire entre les poids et les entrées du neurone) et des poids synaptiques w_{jk} entre le neurone $x_k^{(n-1)}$ et le neurone $x_j^{(n)}$. La notation est alors inversée : w_{jk} indique bien un poids de k vers j.

Cas de la couche de sortie :

L'algorithme de rétropropagation procède à l'adaptation des poids neurone par neurone en commençant par la couche de sortie. Soit l'erreur observée $e_j(n)$ pour le neurone de sortie j et la donnée d'entrainement n :

$$e_j(n) = d_j(n) - y_j(n) \tag{3.1}$$

où $d_j(n)$ correspond à la sortie désirée du neurone j et $y_j(n)$ à sa sortie observée.

– La variable n représentera toujours la donnée d'entrainement c'est-à-dire le couple contenant un vecteur d'entrées et un vecteur de sorties désirées.
– L'objectif de l'algorithme est d'adapter les poids des connexions du réseau de manière à minimiser la somme des erreurs sur tous les neurones de sortie.
– L'indice j représentera toujours le neurone pour lequel on veut adapter les poids. Soit $E(n)$ la somme des erreurs quadratiques observées sur l'ensemble C des neurones de sortie :

$$E(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n) \tag{3.2}$$

Démonstration :

$$\begin{aligned}
\frac{\partial E}{\partial w} &= \frac{\partial E}{\partial y}\frac{\partial y}{\partial h}\frac{\partial h}{\partial w} \\
\frac{\partial E}{\partial w_{ab}^{l}} &= \sum_i (y_i - t_i) g'^{(n)}(h_i^{(n)}) \sum_K w_{ik}^{(n)} \frac{\partial x_k^{(n-1)}}{\partial w_{ab}^{(l)}} \\
\frac{\partial E}{\partial w_{ab}^{l}} &= \sum_K \frac{\partial x_k^{(n-1)}}{\partial w_{ab}^{(l)}} \sum_i w_{ik}^{n} g'^{(n)}(h_i^{(n)})(y_i - t_i) \\
\frac{\partial E}{\partial w_{ab}^{l}} &= \sum_K \frac{\partial x_k^{(n-1)}}{\partial w_{ab}^{(l)}} \sum_i w_{ik}^{n} e_i^{n}
\end{aligned}$$

En utilisant la même technique sur la dérivée partielle de $x_k^{(n-1)}$, on obtient :

$$\frac{\partial E}{\partial w_{ab}^{(l)}} = \sum_K \frac{\partial x_k^{(n-2)}}{\partial w_{ab}^{(l)}} \sum_i w_{ik}^{(n-1)} e_i^{(n-1)}$$

En itérant l'algorithme jusqu'à la couche l, on arrive à :

$$\frac{\partial E}{\partial w_{ab}^{(l)}} = \sum_K \frac{\partial x_k^{(l)}}{\partial w_{ab}^{(l)}} \sum_i w_{ik}^{(l+1)} e_i^{(l+1)} = x_b^{l-1} e_a^{l}$$

On voit qu'en définissant la suite des $e_i^{(l)}$ comme nous l'avons fait, on peut facilement obtenir la dérivée de l'énergie par rapport aux poids synaptiques d'un neurone à distance $n - l$ de la sortie [5].

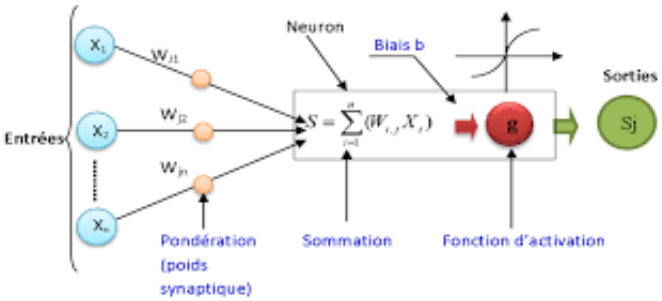


FIGURE 3.13 – Modèle du neurone j.

La sortie $y_j(n)$ du neurone j est définie par :

$$y_j(n) = g[v_j(n)] = g\left[\sum_{i=0}^{r} w_{ji}(n) y_i(n)\right] \tag{3.3}$$

où g est la fonction d'activation du neurone, $v_j(n)$ est la somme pondérée des entrées du neurone j, $w_{ji}(n)$ est le poids de la connexion entre le neurone i de la couche précédente et le neurone j de la couche courante, et $y_i(n)$ est la sortie du neurone i. On suppose ici que la couche précédente contient r neurones numerotés de 1 à r, que le poids $w_{j0}(n)$ correspond au biais du neurone j et que l'entrée $y0(n) = 1$. La figure 2 illustre l'equation 5.

– L'indice i representera toujours un neurone sur la couche précédente par rapport au ´ neurone j ; on suppose par ailleurs que cette couche contient r neurones.

Pour corriger l'erreur observee, il s'agit de modifier le poids $wji(n)$ dans le sens opposé au gradient $\frac{\partial E(n)}{\partial w_{ji}(n)}$de l'erreur (voir figure 3).

– Cette derivée partielle représente un facteur de sensibilité : si je change un peu $w_{ji}(n)$, est-ce que ça change beaucoup $E(n)$? Si oui, alors je vais changer beaucoup $w_{ji}(n)$ dans le sens inverse de cette derivée car cela devrait me rapprocher beaucoup du minimum local. Sinon, je dois changer seulement un peu $w_{ji}(n)$ pour corriger l'erreur car je suis tout près de ce minimum !

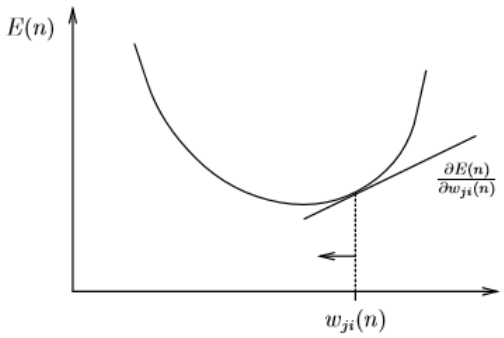


FIGURE 3.14 – Gradient de l'erreur totale

Puisqu'il y a r neurones sur la couche precédant la couche de sortie, il y a aussi r poids a adapter, et il importe donc de remarquer que la courbe de la figure 3 correspond en fait a une hyper-surface de $r+1$ dimensions ! Par la regle de chainage des derivées partielles, qui nous dit que

$$\frac{\partial f(y)}{\partial x} = \frac{\partial f(y)}{\partial y}\frac{\partial y}{\partial x}$$

on obtient :

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)}.\frac{\partial e_j(n)}{\partial y_j(n)}.\frac{\partial y_j(n)}{\partial v_j(n)}.\frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{3.4}$$

et on exprime la variation de poids $\Delta w_{ji}(n)$ sous la forme suivante :

$$\Delta w_{ji}(n) = -\eta\frac{\partial E(n)}{\partial w_{ji}(n)} \tag{3.5}$$

avec $0 \leq \eta \leq 1$ représentant un taux d'apprentissage ou gain de l'algorithme. Evaluons maintenant chacun des termes du gradient.

$$\begin{aligned}\frac{\partial E(n)}{\partial e_j(n)} &= \frac{\partial\left[\frac{1}{2}\sum_{j\in C} e_j^2(n)\right]}{\partial e_j(n)} \\ &= \frac{1}{2}.\frac{\partial e_j^2(n)}{\partial e_j(n)} \\ &= e_j(n)\end{aligned}$$

$$\begin{aligned}\frac{\partial e_j(n)}{\partial y_j(n)} &= \frac{\partial[d_j(n) - y_j(n)]}{\partial y_j(n)} \\ &= -1\end{aligned}$$

$$\begin{aligned}
\frac{\partial y_j(n)}{\partial v_j(n)} &= \frac{\partial\left[\frac{1}{1+e^{-v_j(n)}}\right]}{\partial v_j(n)} \\
&= \frac{e^{-v_j(n)}}{\left[1+e^{-v_j(n)}\right]^2} \\
&= y_j(n)\left[\frac{e^{-v_j(n)}}{1+e^{-v_j(n)}}\right] \\
&= y_j(n)\left[\frac{e^{-v_j(n)}+1}{1+e^{-v_j(n)}} - \frac{1}{1+e^{-v_j(n)}}\right] \\
&= y_j(n)[1-y_j(n)]
\end{aligned}$$

Et finalement :

$$\begin{aligned}
\frac{\partial v_j(n)}{\partial w_{ji}(n)} &= \frac{\partial\left[\sum_{l=0}^{r} w_{jl}(n)y_l(n)\right]}{\partial w_{ji}(n)} \\
&= \frac{\partial[w_{ji}(n)y_i(n)]}{\partial w_{ji}(n)} \\
&= y_i(n)
\end{aligned}$$

Nous obtenons donc :

$$\frac{\partial E_j(n)}{\partial w_{ji}(n)} = -e_j(n)y_j(n)[1-y_j(n)]y_i(n) \tag{3.6}$$

et la règle dite du “delta” pour la couche de sortie s’exprime par :

$$\Delta w_{ji}(n) = -\eta\frac{\partial E(n)}{\partial w_{ji}(n)} = \eta\delta_j(n)y_i(n) \tag{3.7}$$

Avec :

$$\delta_j(n) = e_j(n)y_j(n)[1-y_j(n)] \tag{3.8}$$

qui correspond à ce qu’on appelle le “gradient local”.
– Jusqu’ici, nous avons trait´e seulement le cas de la couche de sortie ! Il reste maintenant a faire l’adaptation des poids sur les couches cachées.

Cas d’une couche cachée :

Considérons maintenant le cas des neurones sur la dernière couche cachée (le cas des autres couches cachées est semblable) [9].

– La variable n désignera toujours la donnée d’entrainement c’est-à-dire un couple de vecteurs d’entrées et de sorties désirées.
– L’objectif sera toujours d’adapter les poids de la couche courante en minimisant la somme des erreurs sur les neurones de la couche de sortie.
– Les indices i et j désigneront respectivement (comme pr´ec´edemment) un neurone sur la couche précédente et un neurone sur la couche courante.
– L’indice k servira maintenant à désigner un neurone sur la couche suivante. Reprenons l’expression de la dérivée partielle de l’erreur totale $E(n)$ par rapport à w_{ji} mais en ne dérivant plus par rapport à l’erreur $e_j(n)$ car celle-ci est maintenant inconnue :

$$\frac{\partial E(n)}{\partial W_ij(n)} = \frac{\partial E(n)}{\partial e_j(n)}.\frac{\partial e_j(n)}{\partial \upsilon_j(n)}.\frac{\partial \upsilon_j(n)}{\partial W_ij(n)} \tag{3.9}$$

Par rapport aux résultats obtenus pour la couche de sortie, les deux derniers termes de cette équation restent inchangés, seul le premier terme requiert d'être évalué :

$$\frac{\partial E(n)}{\partial y_j(n)} = \frac{\left[\frac{1}{2}\sum_{k\in C} e_k^2(n)\right]}{\partial y_j(n)} \tag{3.10}$$

Notre problème ici, contrairement au cas des neurones de la couche de sortie, est que tous les $e_k(n)$ dans la somme ci-dessus dépendent de $y_j(n)$. On ne peut donc pas se débarrasser de cette somme ! Néanmoins, nous pouvons écrire :

$$\begin{aligned}
\frac{\partial E(n)}{\partial y_j(n)} &= \sum_{k\in C}\left[e_k(n).\frac{\partial e_k(n)}{\partial y_j(n)}\right] \\
&= \sum_{k\in C}\left[e_k(n).\frac{\partial e_k(n)}{\partial v_k(n)}.\frac{\partial v_k(n)}{\partial y_j(n)}\right] \\
&= \sum_{k\in C}\left[e_k(n).\frac{\partial[d_k(n)-g(v_k(n)]}{\partial v_k(n)}.\frac{\partial[w_{kl}(n)y_l(n)]}{\partial y_j(n)}\right]
\end{aligned}$$

Et en substituant l'équation 10 on obtient :

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_{k\in C}\delta_k(n)w_{kj}(n) \tag{3.11}$$

En substituant l'équation (1.28) dans l'équation (1.26), on obtient :

$$\frac{\partial E(n)}{\partial W_ij(n)} = -y_j(n).[1-y_j(n)].[\sum_{k\in C}\delta_k(n).W_kj(n)].y_i(n) \tag{3.12}$$

Et :

$$\Delta W_{ji}(n) = -\eta\frac{\partial E(n)}{\partial W_{ji}(n)} = \eta\delta_j(n)y_i(n) \tag{3.13}$$

Avec :

$$\delta_j(n)[1-y_j(n)]\sum_{k\in C}\delta_k(n).W_kj(n) \tag{3.14}$$

On peut démontrer que les équations 15 et 16 sont valides pour toutes les couches cachées [9]. Notez bien, cependant, que dans le cas de la première couche cachée du réseau, puisqu'il n'y a pas de couche précédente de neurones, il faut substituer la variable $y_i(n)$ par l'entrée $x_i(n)$. Nous résumons la méthode rétropropagation dans l'algorithme suivant :

1. Initialiser tous les poids à de petites valeurs aléatoires dans l'intervalle [-0.5, 0.5] ;
2. Normaliser les données d'entrainement ;
3. Permuter aléatoirement les données d'entrainement ;
4. Pour chaque donnée d'entrainement n :
a. Calculer les sorties observées en propageant les entrées vers l'avant ;
b. Ajuster les poids en rétropropageant l'erreur observée :

$$W_ij(n) = W_ij(n-1) + \Delta W_ij(n) = Wij(n-1) + \eta\delta_j(n)y_i(n) \tag{3.15}$$

Où le« gradient local » est défini par :

$$\delta_j(n) = \begin{cases} e_j(n)y_j(n)[1-y_j(n)] sij \in couche\ de\ sortie \\ y_j(n)[1-y_j(n)]\sum_{k\in C}\delta_k(n).W_kj(n) sinonj \in couche\ cache \end{cases} \quad (3.16)$$

Avec $0 \leq \eta \leq 1$ représentant, le taux d'apprentissage et $Y_i(n)$ représentant soit la sortie du neurone i sur la couche précédente, si celui-ci existe, soit l'entrée i autrement. 5. Répéter les étapes 3 et 4 jusqu'à un nombre maximum d'itérations ou jusqu'à ce que la racine de l'erreur quadratique moyenne (EQM) soit inférieure à un certain seuil.

Algorithm 3.7 *Algorithme de rétropropagation du gradient*

— **Entrée** : Un échantillon S de $\mathbf{R}^n \times \mathbf{R}^p$
Un perceptron PMC avec une couche d'entrée $C_0, q-1$ couches cachée $C_1, ..., C_{q-1}$et une couche de sortie C_q, n cellules
initialisation aléatoire des poids w_{ij} dans $[0,5; 0,5]$, pour i entre 0 et n
— **Sortie** : Un PMC défini par la structure initiale choisie et les w_{ij}.

1. **Répéter**
   Prendre un exemple $(\vec{x}, c)$ dans S
   calculer la sortie o du perceptron pour l'entrée $\vec{x}$.
2. **Pour** toute cellule de sortie i $d_i \longleftarrow o_i(1-o_i)(c_i-o_i)$ **Fin Pour**
3. **Pour** chaque couche de $q-1$ à 1
4. **Pour** chaque cellule i de la couche courante $d_i \longleftarrow o_i(1-o_i)\sum_{k\in Succ(i)} d_k w_{ki}$
   Fin Pour
   Fin Pour Mise à jour des poids
5. **Pour** tout poids$w_{ij} \longleftarrow w_{ij} + ed_i x_{ij}$.
   Fin pour
   Fin Répéter

3.6.6 Exemples

Exemple 1 :

Considérons le réseau à deux entrées x_1 et x_2 de la figure 3.15. Il possède une couche cachée composée des neurones formels numérotés 3 et 4. Sa couche de sortie est composée d'un seul neurone formel, numéroté 5. Il y a une valeur d'offset fixée à 1 pour le vecteur d'entrée et une autre pour la couche cachée.
Fixons maintenant (figure) la valeur des poids comme suit :

$$w(0,3) = 0.2 \quad w(1,3) = 0.1 \quad w(2,3) = 0.3$$
$$\text{w(0,4)} = 0.3 \quad \text{w(1,4)} = 0.2 \quad \text{w(2,4)} = 0.4$$
$$\text{w(0,5)} = 0.4 \quad \text{w(3,5)} = 0.5 \quad \text{w(4,5)} = 0.4$$

et prenons pour vecteur d'entrée $X = \begin{pmatrix} 1 \\ \\ 1 \end{pmatrix}$

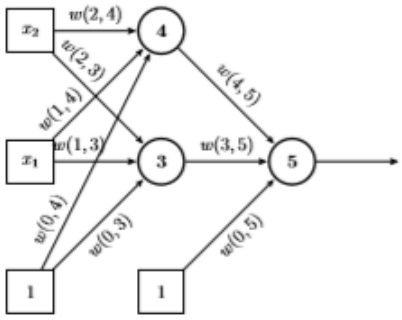


FIGURE 3.15 – Un exemple de réseau multicouche : la notation des neurones formels et des poids des connexions. Les entrés sont des carrés, les neurones sont des cercles.

La propagation des calculs s'effectue alors comme indiqué sur la table ci-dessous :

Neurone formel j	σ_j	y_j
3	$0.2 + 0.1 \times 1 + 0.3 \times 1 = 0.6$	$\frac{1}{1+e^{-0.6}} \simeq 0.65$
4	$-0.3 + -0.2 \times 1 + 0.4 \times 1 = -0.1$	$\frac{1}{1+e^{0.1}} \simeq 0.48$
5	$0.4 + 0.5 \times 0.65 - 0.4 \times 0.48 = 0.53$	$\frac{1}{1+e^{-0.53}} \simeq 0.63$

TABLE 3.1 – La propagation des calculs.

en supposant que la sortie désirée au vecteur d'entrée $x^T = (1, 1)$ vaille $u = 0$. Après modification des poids sur cet exemple, son nouveau passage dans le réseau doit conduire à une sortie inférieure à la valeur précédente, qui était de 0.63.
Pour le neurone formel de sortie, on a :

$$\Delta w_{(i,j)} = \alpha \delta_j y_i$$

avec :

$$\delta_j = (u_j - y_j)\ y_j\ (1 - y_j)$$

On prend d'abord : $i = 3$ et $j = 5$, ce qui mène à :

$$\delta_5 = (0 - 0.63) \times 0.63 \times (1 - 0.63) = -0.147$$

d'où :

$$\Delta w(3, 5) = -0.147 \times 0.65 \simeq -0.1$$

en fixant la valeur α à 1. De même, pour $i = 4$, on obtient :

$$\Delta w(4, 5) = 0.48 \times -0.147 \simeq -0.07$$

$$\Delta w(0, 5) = -0.147 \times 1 = -0.147$$

Pour le neurone formel caché noté 4, on a d'abord, puisque $dest(4) = 5$:

$$\delta_4 = y_4 \times (1 - y_4) \times \delta_5 \times w(4, 5) = 0.48 \times (1 - 0.48) \times -0.147 \times -0.4 \simeq 0.015$$

D'où :

$$\Delta w(1, 4) = 0.015 \times 1. = 0.015$$

$$\Delta w(2, 4) = 0.015 \times 1. = 0.015$$

$$\Delta w(0, 4) = 0.015 \times 1. = 0.015$$

De même, puisque $dest(3) = 5$:

$$\delta_3 = y_3 \times (1 - y_3) \times \delta_5 \times w(3,5) = 0.65 \times (1 - 0.65) \times -0.147 \times 0.5 \simeq -0.017$$

D'où :

$$\Delta w(1,3) = 0.016 \times 1. = -0.017$$
$$\Delta w(2,3) = 0.016 \times 1. = -0.017$$
$$\Delta w(0,3) = 0.016 \times 1. = -0.017$$

Après modification, les poids deviennent donc :

$$w(0,5) + \Delta w(0,5) = 0.4 - 0.147 \simeq 0.25$$
$$w(3,5) + \Delta w(3,5) = 0.5 - 0.1 = 0.4$$
$$w(4,5) + \Delta w(4,5) = -0.4 - 0.07 = -0.47$$

$$w(0,3) + \Delta w(0,3) = 0.2 - 0.017 = 0.183$$
$$w(1,3) + \Delta w(1,3) = 0.1 - 0.017 = 0.083$$
$$w(2,3) + \Delta w(2,3) = 0.3 - 0.017 = 0.283$$

$$w(0,4) + \Delta w(0,4) = -0.3 + 0.015 = -0.285$$
$$w(1,4) + \Delta w(1,4) = -0.2 + 0.015 = -0.185$$
$$w(2,4) + \Delta w(2,4) = 0.4 + 0.015 = 0.415$$

Dans le réseau modifié, le calcul sur le vecteur d'entrée devient par conséquent :

Neurone formel j	σ_j	y_j
3	$0.183 + 0.083 \times 1 + 0.283 \times 1 = 0.55$	$\frac{1}{1+e^{0.65}} \simeq 0.63$
4	$-0.285 + -0.185 \times 1 + 0.415 \times 1 = -0.055$	$\frac{1}{1+e^{-0.055}} \simeq 0.51$
5	$0.25 + 0.4 \times 0.63 - 0.47 \times 0.51 = 0.26$	$\frac{1}{1+e^{0.205}} \simeq 0.56$

TABLE 3.2 – Le calcul sur le vecteur d'entrée.

Si on compare la valeur de sortie à celle du tableau de la section 2.1 on constate qu'elle est passée de 0.63 avant apprentissage à 0.56 après : elle s'est rapprochée de la valeur désirée 0.

Exemple 2 :

Considérons un réseau de neurones avec deux unités d'entrée, trois unités cachées et une seule unité de sortie Figure , c'est-à-dire $n = 2$, $l = 3$ et $m = 1$. Ce réseau est censé fonctionner en fonction "XOR". Quatre modèles d'apprentissage qui sont des vecteurs $2 - D$ $(n = 2)$ sont les entrées. Les sorties sont quatre scalaires $(m = 1)$. Si une représentation binaire est considérée, les motifs sont $(1, 1)$, $(1, -1)$, avec des cibles $0, 1, 1, 0$ respectivement et la fonction d'activation est la sigmoïde binaire [14].

Pour une représentation bipolaire, les modèles d'entrée sont $(1, 1)$, $(1, -1)$, $(-1, 1)$, $(-1, -1)$ avec des cibles $-1, 1, 1, -1$ respectivement et l'activation fonction est le sigmoïde.

Les poids initiaux choisis aléatoirement entre $-0,5$ et $0,5$ sont :

$$v_{01} = 0.396, \quad v_{02} = -0.030, \quad v_{03} = -0.401$$

$$v_{11} = -0.066, \quad v_{12} = -0.046, \quad v_{13} = 0.414$$

$$v_{21} = -0.017, \quad v_{22} = 0.220, \quad v_{23} = 0.231$$

et

$$w_{01} = 0.118, w_{11} = -0.173, w_{21} = 0.204, w_{31} = -0.271$$

Le triage du réseau se poursuit jusqu'à ce que l'erreur quadratique totale soit suffisamment petite, c'est-à-dire

$$\sum_{p=1}^{M} \sum_{k=1}^{m} (y_k^{(p)} - t_k^{(p)})^2 < \epsilon$$

Pour une tolérance $\epsilon = 0.05$ et un taux d'apprentissage $\alpha = 0.1$ on obtient les résultats suivants :

a) Représentation binaire : 15342 itérations sont nécessaires pour la convergence et les sorties finales sont :

$$y_1^{(1)} = 0.12, y_1^{(2)} = 0.870, y_1^{(3)} = 0.899, y_1^{(4)} = 0.102$$

b) Représentation bipolaire : 823 itérations sont nécessaires pour la convergence et les sorties finales sont :

$$y_1^{(1)} = -0.883, y_1^{(2)} = 0.910, y_1^{(3)} = 0.895, y_1^{(4)} = -0.869$$

3.6.7 Le nombre de neurones dans la couche cachée

La détermination du nombre de neurones dans la couche cachée est délicate. Comme indiqué dans [6], pour un problème donné et des échantillons de taille fixe, un réseau «sous-estimé» (le nombre de neurones dans la couche cachée ou le nombre de couches cachées est insuffisant), aura un nombre de degrés de liberté trop faible ; la fonction d'erreur aura donc une composante de biais importante et un terme de variance faible ; le modèle est stable, mais a une performance relativement faible. Par contre, un réseau «sur-estimé» (un grand nombre des couches cachées ou un grand nombre de neurones dans la couche cachée), possède un grand nombre de degrés de liberté et l'optimisation à partir d'échantillons d'apprentissage différents conduira à des solutions pouvant être différentes, ce qui correspond à une composante de variance importante. Le modèle neuronal devient instable. Sous l'hypothèse que le RPM ne dispose que d'une seule couche cachée, nous utilisons la technique de rééchantillonnage pour la sélection du réseau optimal. Dans le cas où le réseau possède deux couches cachées, nous utilisons la procédure de détermination de la topologie optimale du réseau, décrite dans [54]. Au delà de trois couches cachées, le nombre optimal de neurones contenus par les couches cachées ne peut être fixé qu'expérimentalement. Par conséquent, un nombre trop élevé de neurones entraîne d'inévitables redondances et donc pénalise le temps de calcul inutilement employé . Différentes études sont effectuées selon le cas.

3.6.8 Problèmes liés à l'algorithme de rétropropagation

L'apprentissage par l'algorithme rétropropagation pose plusieurs problèmes, les principaux qu'on peut citer sont comme suit [51, 11] :

1. **L'architecture du réseau :** il n'existe pas de règle générale pour déterminer la structure des réseaux (le nombre de couches cachées et le nombre de neurones par couche), sachant que le problème critique pendant l'apprentissage est de trouver un réseau assez large pour bien apprendre mais également assez petit pour bien généraliser. On ne sait (presque) pas dimensionner correctement le réseau. Les couches entrées et sorties sont bien sur imposées, puisque le nombre de neurones qu'elles admettent dépend du problème posé. Mais que dire surtout du nombre de neurones des couches cachées ? Si chaque neurone a une fonction de sortie de type sigmoïde (ex : tanh), et plus généralement si cette fonction est non linéaire (sauf polynomiale) et dérivable, alors Cybenko a démontré en 1989 qu'un réseau multicouche est capable d'approximer n'importe quelle fonction, à condition que le nombre de neurones dans la couche cachée soit suffisant.
   — Une méthode par élagage consiste à initialiser le réseau avec un très grand nombre de neurones dans la couche cachée et à supprimer ceux d'entre eux dont les poids synaptiques sont très faibles et n'influencent pas trop le comportement du réseau.
   — Une méthode par construction consiste à rajouter des neurones dans la couche cachée au fur et à mesure des "besoins", i.e. lorsque l'erreur globale du réseau ne diminue plus.
2. **Problème des valeurs initiales des poids du réseau :** un autre problème est le temps de convergence de l'algorithme de rétropropagation. En effet, plus la somme pondérée des entrées d'un neurone est forte, plus le neurone se trouve dans la zone de saturation de sa fonction d'activation (tanh) , donc plus la dérivée ' est faible (i.e. la pente de la fonction en zone de saturation), et moins les poids du neurone sont modifiés. Il faut donc démarrer l'apprentissage en initialisant les poids du réseau à des valeurs suffisamment faibles qui placent la fonction d'activation dans sa zone linéaire : on choisit donc en général des valeurs initiales inférieures à 0.1.

3. **Le temps d'apprentissage :** le temps d'apprentissage augmente avec le nombre de couples d'apprentissage, ce qui diminue la vitesse de convergence.

4. **La convergence de l'algorithme :** aucune preuve mathématique sur la convergence de cet algorithme vers un minimum global n'existe du fait que cette méthode utilise la décente du gradient. La recherche d'un minimum global sur la surface de l'erreur dans le domaine des poids, peut présenter un problème si cette surface possède des minimums locaux qui peuvent ralentir (voire stopper) l'algorithme. Le choix d'apprentissage variable permet dans certains cas d'accélérer la convergence. Il arrive cependant qu'on reste au dessus du critère d'arrêt sans jamais l'atteindre. C'est souvent le signe que le mécanisme d'apprentissage est inadapté ou que l'architecture du réseau ne permet d'atteindre ce degré de précision. Dans ce cas il faut augmenter le nombre de neurones de la couche cachée ou changer de structure.

5. **Le pas de correction des poids :** si le pas de correction des poids est très petit, l'apprentissage nécessite alors un temps très important. Par contre si ce même pas est très grand le réseau devient oscillatoire, ce qui compromet sa convergence. La solution à ce problème est de choisir un pas variable initialisé à une grande valeur comprise entre 0 et 1, et qui sera diminuer jusqu'à une valeur minimale positive fixée au préalable.
6. **Le pas d'apprentissage :** le réglage du pas d'apprentissage) t(joue aussi un rôle important dans la vitesse de convergence. Ainsi il est préférable qu'il soit grand au début de l'apprentissage, et diminue au fur et à mesure que le réseau se rapproche de la solution. (La valeur du pas d'apprentissage est de l'ordre de 0.1 à 0.001).
7. **La saturation du réseau :** si les poids prennent des grandes valeurs, les sorties deviennent grandes et se rapprochent de la zone de saturation de la fonction d'activation. La solution à ce problème est un compromis à faire entre : 1- Le choix d'un pas de

correction petit ; 2- L'initialisation des poids à de très petites valeurs.

8. **Problème de sur-apprentissage :** il faut aussi donner suffisamment d'exemples bien répartis (i.e. représentatifs) pour que le réseau généralise correctement, mais pas trop pour qu'il ne fasse pas de sur-apprentissage (i.e. de l'apprentissage par cœur) au détriment des capacités de généralisation. Un moyen simple de vérifier qu'il n'y a pas sur-apprentissage consiste à comparer l'erreur quadratique globale du réseau qui décroît toujours, et l'erreur faite par le réseau sur la base de test qui diminue puis augmente lorsqu'il y a sur-apprentissage. La base de test ne doit jamais servir à l'apprentissage.

9. **Le paramètre :** appelé taux d'apprentissage, joue un rôle important. S'il est trop faible, la convergence est lente, et s'il est trop grand l'algorithme oscille entre des points différents à cause de l'existence de vallées et de plateaux à la surface de la fonction cout. Pour stabiliser la recherche des poids optimisant la fonction coût, une méthode consiste à ajouter un terme dit de "moment" ou « momentum » à l'expression d'adaptation des poids, l'idée est de donner une certaine "inertie" pour chaque poids, de sorte que sa mise à jour ne se fasse pas de manière brutale. Ceci permet alors d'utiliser un taux d'apprentissage relativement important sans pour autant augmenter les oscillations de la trajectoire sur la surface d'erreur. Le choix de ce facteur est cependant délicat, on peut d'ailleurs aboutir à des effets inverses, des oscillations ou un ralentissement de la convergence.

10. **Le nombre d'itération :** On ne peut pas prévoir le nombre d'itération nécessaire à l'apprentissage. L'algorithme de rétropropagation introduit la dérivée première des fonctions d'activation. Il est cependant tout à fait envisageable d'utiliser d'autres algorithmes qui ne nécessitent pas de dérivation comme par exemple les algorithmes génétiques.

3.7 conclusion

Dans de ce chapitre nous avons présenté la technique des réseaux de neurones en s'intéressant plus particulièrement aux perceptrons multicouches (PMC) qui peuvent être capables de résoudre le problème étudié. La création d'un perceptron multicouche pour résoudre un problème donné passe donc par l'inférence de la meilleure application possible tel que définie par un ensemble de données d'apprentissages constitués de paires de vecteurs d'entrées et de sorties désirées. Cette inférence peut se faire, entre autres, par l'algorithme dit de rétropropagation. Ce dernier dont nous avons parlé en détail. Par ailleurs, le modèle multicouche est le plus utilisé parmi les différents modèles des réseaux neuronaux, car il permet d'approximer n'importe quelle fonction de transfert donnée, après un choix approprié de ses paramètres.

Nous avons également mentionné les problèmes liés à l'algorithme de rétropropagation Le chapitre suivant traite le problème de classification non linéaire en utilisant l'architecture PMC.

Chapitre 4

Optimisation du réseau PMC par des algorithmes heuristiques

Sommaire

4.1 Introduction

Les réseaux de neurones artificiels sont fréquemment utilisés pour la classification, en raison de leur capacité à résoudre des problèmes non linéaires avec beaucoup de succès. Pour une tache de classification, le réseau neuronal artificiel cherche une structure de réseau avec un ensemble de paramètres à laide d'un algorithme d'apprentissage.

La classification est une méthode mathématique d'analyse de données, il est appliqué sur des données numériques (points, tableaux, images, sons...etc), pour faciliter l'étude d'une population d'effectifs importants.

Nous présenterons dans ce chapitre tout d'abord qu'est-ce que c'est la classification supervisée, nous présenterons les méthodes de classification ainsi que leurs différentes techniques et le domaine d'applications...stc.

Puis, nous donnerons quelques exemples des classification appliquées par le réseau perceptron multicouche à l'aide du logiciel Matlab.

Le perceptron multicouche (MLP) est un outil essentiel pour résoudre les problèmes de classification, d'identification et de généralisation. Les paramètres d'apprentissage de l'algorithme de rétropropagation et de la structure du réseau sont optimisés pour réussir un processus de mise à jour de poids plus rapide et efficace.

Malgré les nombreux avantages des réseaux de neurones, ils présentent un inconvénient principal qui est la décision des paramètres par essais et erreurs, ou approximativement. Si le taux d'apprentissage dans l'algorithme de rétropropagation est choisi plus grand, l'apprentissage devient plus rapide mais le risque d'oscillation apparaît. Mais, si le taux d'apprentissage est faible, le processus d'apprentissage prend peut-être très longtemps. La même situation est également observée dans le coefficient de quantité de mouvement. Le nombre de neurones cachés est déterminé de manière totalement aléatoire. La sélection d'une fonction d'activation différente pour chaque solution de problème peut donner une meilleure solution en fonction de la caractéristique de sortie. Parce que prendre la décision de toutes ces valeurs prend du temps, le développement d'un algorithme d'optimisation est nécessaire.

Il existe diverses études dans la littérature avec la colonie d'abeilles artificielles (ABC), l'algorithme de recuit simulé (SA) et l'algorithme génétique (GA) concernant les cas dans lesquels un paramètre est optimisé tandis que tous les autres sont maintenus constants. Il existe un nombre limité d'études qui optimisent plus d'un paramètre en utilisant un seul algorithme.

Les travaux que nous allons présenter dans ce chapitre visent à optimiser quatre paramètres, le nombre de neurones cachés, le taux d'apprentissage, le coefficient d'impulsion, de l'algorithme de rétropropagation et le type de la fonction d'activation utilisé.

4.2 Classification supervisée

La classification supervisée ou le classement automatique est la catégorisation algorithmique d'objets. Elle consiste à attribuer une classe ou catégorie à chaque objet (ou individu) à classer, en se fondant sur des données statistiques. Elle fait couramment appel à l'apprentissage automatique et est largement utilisée en reconnaissance de formes.

La classification supervisée consiste en l'affectation d'une nouvelle observation prise sur le système à un moment donné à l'une des classes élaborées lors de la phase d'apprentissage.

Un classifieur est une fonction de décision qui permet de réaliser la séparation des données. On ne s'interesse qu'aux méthodes de classification supervisées à deux classes dont on distingue deux types : Classification séparatrice et classification probabiliste.

4.2.1 Méthodes probabiliste

La méthode de classification probabiliste utilise une hypothèse sur la distribution des individus à classifier. Par exemple, on peut considérer que les individus de chacune des classes suivent une loi normale. Le problème qui se pose alors est de déterminer quels sont les paramètres des lois (moyenne, variance) et à quelle classe les individus ont le plus de chances d'appartenir. Les paramètres d'une loi peuvent être déterminés de maintes façons, et notamment en utilisant l'algorithme espérance-maximisation.

Considérons un problème de classification à deux classes $c = 2$. Les deux objets sont représentés par un vecteur forme X tel que $X = [x_1 x_1 ... x_1]^T$. Notons : $P = proba(c = c_1)$ la probabilité d'appartenance à priori d'un objet quelconque à la classe C_1. On déduit alors que $p_2 = proba(C = C_2) = 1 - p_1$, étant donné qu'on dispose seulement de deux classes.

Si l'objet appartient à la classe C_1, le vecteur forme (appelé dans ce contexe vecteur aléatoire) X possède la densité de pronanilité conditionnel $f_{x/C_1}(X)$, si l'objet appartient à la classe C_2, X possède la densité de probabilité conditionnel $f_{x/C_2}(X)$.

Avec :

$$p_1.f_{x/C_1}(X) + p_2.f_{x/C_2}(X) = f_X(X)$$

$f_X(X)$ est la densité de probabilité conditionnelle de (X).

La théorème de Bayes permet d'exprimer les probabilités à posteriori en fonction des probabilités à priori et des densités de probabilité conditionnelle comme suit :

$$proba(C = C_{i/X}) = \frac{p_i.f_{x/C_1}(X)}{f_X(X)} = \frac{p_i.f_{x/C_1}(X)}{p_1.f_{x/C_1}(X) + p_2.f_{x/C_1}(X)} \tag{4.1}$$

$i = 1, 2$
Le terme $proba(C = C_{i/X})$ se lit probabilité d'appartenance d'un objet à la classe C_i sachant que sa représentation est X, cette robabilité est appelée **probabilité à posteriori**.

4.2.2 Méthodes séparatistes

Séparation linéaire entre deux classes :

L'un des plus connu des réseaux de neurones utilisé comme classifieur linéaire est le perceptron. Le perceptron vise à définir une fonction discriminante linéaire séparant les 2 classes. La réponse (sortie) S fournie toujours l'une des deux valeurs (0.1) selon que l'objet appartient a une classe ou à une autre. Si on substitue la fonction d'activation seuil a une sigmoide définie au paragraphe dont la sortie prend des valeurs dans l'intervalle 01, on obtiendra un classifieur probabiliste fournissant les probabilités d'appartenance à posteriori dans chaque classe.

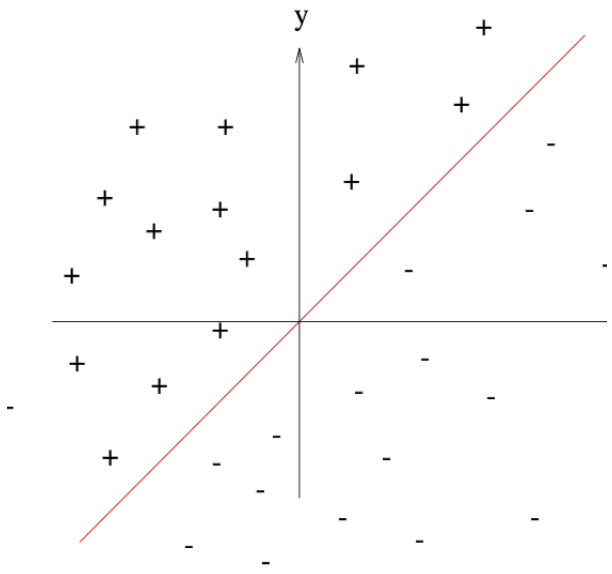


FIGURE 4.1 – Classification linéaire

4.2.3 Classification binaire

La classification binaire classe les données dans l'un des deux groupes. Ce type de classificateur est en fait utile pour plus d'applications que vous ne le pensez [27]. Certaines applications typiques incluent le filtrage du courrier indésirable (un courrier indésirable ou un courrier normal) et les approbations de prêt (approuver ou refuser).

Pour la classification binaire, un seul nœud de sortie est suffisant pour le neurone réseau. C'est parce que les données d'entrée peuvent être classées par la valeur de sortie, qui est supérieur ou inférieur au seuil. Par exemple, si la la fonction sigmoïde est utilisée comme fonction d'activation du nœud de sortie, les données peuvent être classées selon que la sortie est supérieure à 0,5 ou non. Comme le la fonction sigmoïde va de 0 à 1, nous pouvons diviser les groupes au milieu, comme indiqué sur la figure 4.2.

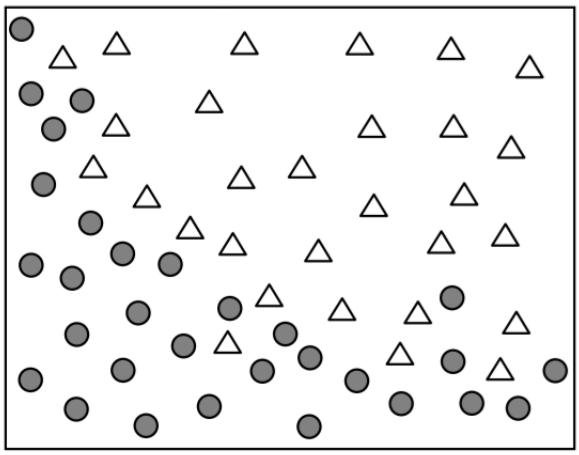


FIGURE 4.2 – Problème de classification binaire

Considérez le problème de classification binaire illustré sur la figure 4.2. Pour les coordonnées données (x, y), le modèle doit déterminer à quel groupe appartiennent les données. Dans ce cas, les données d'apprentissage sont fournies dans le format illustré comme suivant :

$\{5, 7, \Delta\}$
$\{9, 8, \}$
...
$\{6, 5, \}$

Les deux premiers chiffres indiquent respectivement les coordonnées x et y, et le symbole représente le groupe auquel appartiennent les données. Les données sont constituées de l'entrée et une sortie correcte telle qu'elle est utilisée pour l'apprentissage supervisé.

Maintenant, construisons le réseau de neurones. Le nombre de nœuds d'entrée est égal au nombre de paramètres d'entrée. Comme l'entrée de cet exemple consiste de deux paramètres, le réseau emploie deux nœuds d'entrée. Nous avons besoin d'un nœud de sortie car cela implémente la classification de deux groupes comme précédemment abordé. La fonction sigmoïde est utilisée comme fonction d'activation et la couche cachée a quatre nœuds.1 La figure 4.3 montre le réseau de neurones décrit.

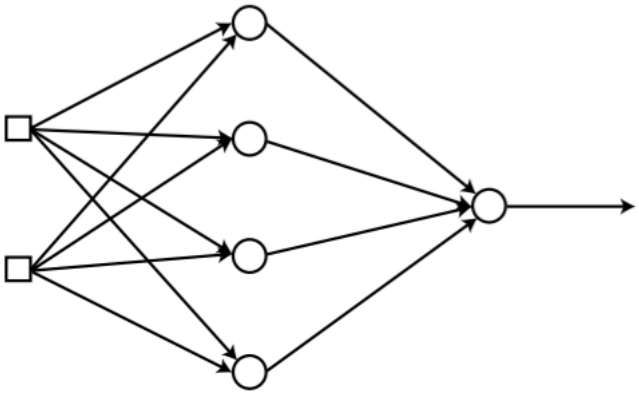


FIGURE 4.3 – Le réseau de neurone pour les données d'entrainement

Les données d'entraînement illustrées sont celles que nous utilisons pour entraîner le neurone réseau. Le réseau de neurones de classification binaire adopte généralement la fonction

d'entropie croisée de l'équation précédente pour l'entraînement. Vous n'êtes pas obligé de le faire tout le temps, mais cela est bénéfique pour la performance et le processus de mise en œuvre.

4.2.4 Classification multiclasse

Cette section explique comment utiliser le réseau de neurones pour gérer les classification de trois classes ou plus. Considérons une classification des données entrées de coordonnées (x, y) dans l'une des trois classes Figure 4.4.

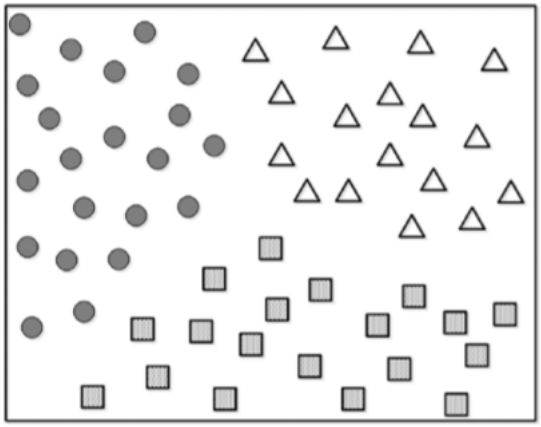


FIGURE 4.4 – problème de classification multiclasse

Nous devons d'abord construire le réseau de neurones. Nous utiliserons deux nœuds pour la couche d'entrée car l'entrée se compose de deux paramètres. Par souci de simplicité, les couches cachées ne sont pas prises en compte pour le moment. Nous devons également déterminer le nombre de nœuds de sortie [27]. Il est bien connu que faire correspondre le nombre de nœuds de sortie au nombre de classes est la méthode la plus prometteuse. Dans cet exemple, nous utilisons trois nœuds de sortie, car le problème nécessite trois classes. Figure 4.5 illustre le réseau de neurones configuré.

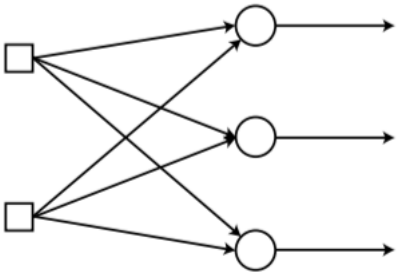


FIGURE 4.5 – Le réseau de neurone configuré pour les trois classes

4.2.5 Différentes méthodes de la classification

— **Classification hiérarchiques :** En classification hiérarchique ascendante. Le procède consiste à grouper les observations individuelles en classes par partie de la même classe. Les méthodes se distinguent par le choix de la distance entre les observations et la

définition de la stratégie d'agrégation [35].
Dans l'algorithme de base, le calcul de la distance (il s'agit plus exactement d'une quantité critère que l'on appelle distance par abus de langage) fait par récurrence à partir de la matrice des distances entre observations.

— **Classification non hiérarchiques :** La classification non hiérarchique ou partitionnement, aboutissant à la décomposition de l'ensemble de tous les individus en m ensemble disjoints ou classes d'équivalence, le nombre m de classes est fixé (6). Le résultat obtenu est alors une partition de l'ensemble des individus, un ensemble de parties, ou classes de l'ensemble I des individus telles que :
- Toute classe soit non vide.
- Deux classes distinctes sont disjointes.
- Tout individu appartient à une classe.
Cet algorithme porte le nom de "agrégation autour de centres variables". Une version légèrement différente, connue sous le nom de "nuées dynamiques" consiste à représenter chaque groupe non pas par son centre, mais par un ensemble de points (noyau) choisis aléatoirement à l'intérieur de chaque groupe. (6) On calcule alors une distance "moyenne" entre chaque observation et ces noyaux et l'on procède à l'affectation. Exemple de technique de typologie : les k-means.

— **Classification paramétrique :** en utilisant la fonction de discrimination optimisée par l'entraînement. Cette technique consiste à faire une hypothèse sur la forme analytique de la probabilité de distribution recherchée [26]. Puis l'estimation des paramètres (moyenne, covariance ...) de cette distribution avec apprentissage. La méthode la plus utilisée est la méthode Gaussienne.

— **Classification non paramétrique :** en utilisant la fonction discriminative basé sur l'estimation de la densité des différentes classes voisines de l'observation. Cette méthode est nommée k plus proche voisins (kpp). Elle consiste à mesurer la distance entre la position du nouvel objet et les classes voisines. Ce nouvel objet sera attribué à la meilleure classe représentative parmi les k observation délimités autour de ce nouvel objet.

— **Classification par réseau de neurones** qui permet d'estimer la probabilité à posteriori d'objets appartenant à une classe [26].

4.2.6 Domaine d'application de la classification

La classification joue un rôle important dans toutes les sciences et techniques qui font appel à la statistique multidimensionnelle [35].
Citons tout d'abord les sciences biologiques : botanique, zoologie, écologie,...Ces sciences utilisent également le terme de "taxinomie" pour désigner l'art de la classification. De même les sciences de la terre et des eaux : géologie, pédologie, géographie, étude des pollutions, font grand usage de classifications.

4.2.7 Quelques techniques de la classification

1. **k plus proches voisin :**
la méthode des k plus proches voisins est une méthode d'apprentissage supervisé.cette méthode consiste à prendre en compte (de façon identique) les k échantillons d'apprentissage dont l'entrée est la plus proche de la nouvelle entrée x, selon une distance à

définir. Puisque cet algorithme est basé sur la distance.

l'algorithme des k plus proches voisins $(k - NN)$ est une méthode non paramétrique utilisée pour la classification et la régression. Dans les deux cas, il s'agit de classer l'entrée dans la catégorie à laquelle appartient les k plus proches voisins dans l'espace des caractéristiques identifiées par apprentissage. Le résultat dépend si l'algorithme est utilisé à des fins de classification ou de régression :

-En classification $k - NN$, le résultat est une classe d'appartenance. Un objet d'entrée est classifié selon le résultat majoritaire des statistiques de classes d'appartenance de ses k plus proches voisins, (k est un nombre entier positif généralement petit). Si $k = 1$, alors l'objet est affecté à la classe d'appartenance de son proche voisin.
-En régression $k-NN$, le résultat est la valeur pour cet objet. Cette valeur est la moyenne des valeurs des k plus proches voisins.

La méthode $k - NN$ est basée sur l'apprentissage préalable, ou l'apprentissage faible, où la fonction est évaluée localement, le calcul définitif étant effectué à l'issue de la classification. L'algorithme $k-NN$ est parmi les plus simples des algorithmes de machines learning.

2. **Les Réseaux de Neurones :**
   Les réseaux de neurones proposent une simulation du fonctionnement de la cellule nerveuse à l'aide d'un automate : le neurone formel. Les réseaux neuronaux sont constitués d'un ensemble de neurones (nœuds) connectés entre eux par des liens qui permettent de propager les signaux de neurone à neurone. Grâce à leur capacité d'apprentissage, les réseaux neuronaux permettent de découvrir des relations complexes non-linéaires entre un grand nombre de variables, sans intervention externe. De ce fait, ils sont largement utilisés dans de nombreux problèmes de classification (ciblage marketing, reconnaissance de formes, traitement de signal,. . .) d'estimation (modélisation de phénomènes complexes,. . .) et prévision (bourse, ventes,. . .). Il existe un compromis entre clarté du modèle et pouvoir prédictif. Plus un modèle est simple, plus il sera facile à comprendre, mais moins il sera capable de prendre en compte des dépendances trop variées.
3. **Machine à vecteurs de support :**
   les SVM sont une généralisation des classifiées linéaires. les SVM ont été développés dans les années 1990 à partir des considérations théoriques de Vladimir Vapnik sur le développement d'une théorie statistique de l'apprentissage : la Théorie de VapnikChervonenkis. Les SVM ont rapidement été adoptés pour leur capacité à travailler avec des données de grandes dimensions, le faible nombre d'hyper paramètres, leurs garanties théoriques, et leurs bons résultats en pratique.

   Les machines à vecteurs de support ou séparateurs à vaste marge (en anglais Support Vector Machine, SVM) sont un ensemble de techniques d'apprentissage supervisé destinées à résoudre des problèmes de classification.

   Les SVM ont été appliqués à de très nombreux domaines (bio-informatique, recherche d'information, vision par ordinateur, finance...). Selon les données, la performance des machines à vecteurs de support est de même ordre, ou même supérieure, à celle d'un réseau de neurones ou d'un modèle de mixture gaussienne.

4.3 Application au réseau PMC

4.3.1 Résoudre le problème XOR par le PMC

Dans cet exemple nous allons résoudre le problème XOR par le perceptron multicouche, le but est de classer un ensemple de points de deux couleurs différentes en deux classes A et B, la classe A représente les plus noire et la classe B représente les carrés bleus figure 4.6.

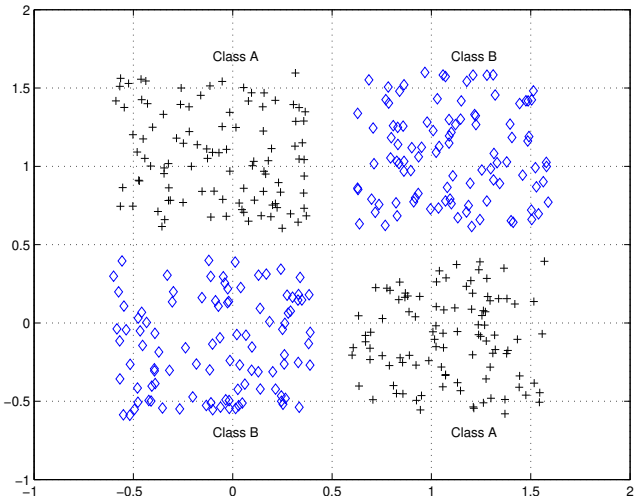


FIGURE 4.6 – Classification en deux classes par le perceptron multicouche

Un perceptron multicouche avec deux entrées doit être formé. Seulement un neurones de sortie représente les deux classes, deux couches cachés , la première couche contient 5 neurones et la deusième de 3 neurones figure 4.7.

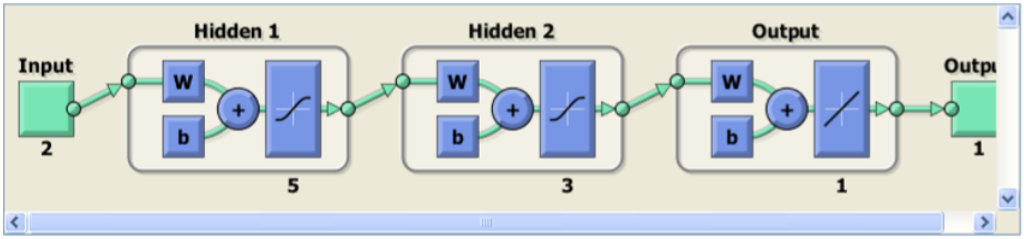


FIGURE 4.7 – Le percepreon multicouche entrainé

Nous allons entrer ces données dans l'ogiciel Matlab,

Listing 4.1 – Code source .

```
close all, clear all, clc, format compact
% number of samples of each class
K = 100;
% define 4 clusters of input data
q = .6; % offset of classes
A = [rand(1,K)-q; rand(1,K)+q];
B = [rand(1,K)+q; rand(1,K)+q];
```

```
C = [rand(1,K)+q; rand(1,K)-q];
D = [rand(1,K)-q; rand(1,K)-q];
% plot clusters
figure(1)
plot(A(1,:),A(2,:),'k+')
hold on
grid on
plot(B(1,:),B(2,:),'bd')
plot(C(1,:),C(2,:),'k+')
plot(D(1,:),D(2,:),'bd')
% text labels for clusters
text(.5-q,.5+2*q,'Class A')
text(.5+q,.5+2*q,'Class B')
text(.5+q,.5-2*q,'Class A')
text(.5-q,.5-2*q,'Class B')
% encode clusters a and c as one class, and b and d as another
   class
a = -1; %  a  |  b
c = -1; %  ------
b =  1; %  d  |  c
d =  1; %
% define inputs (combine samples from all four classes)
P = [A B C D];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B)) ...
     repmat(c,1,length(C)) repmat(d,1,length(D)) ];
% view inputs |outputs
%[P' T']
% create a neural network
net = feedforwardnet([5 3]);
% train net
net.divideParam.trainRatio = 1; % training set [%]
net.divideParam.valRatio   = 0; % validation set [%]
net.divideParam.testRatio  = 0; % test set [%]
% train a neural network
[net,tr,Y,E] = train(net,P,T);
% show network
view(net)

% Plot the network
figure(2)
plot(T','linewidth',2)
hold on
plot(Y','r--')
grid on
legend('Targets','Network response','location','best')
ylim([-1.25 1.25])

% generate a grid
span = -1:.005:2;
[P1,P2] = meshgrid(span,span);
pp = [P1(:) P2(:)]';
% simulate neural network on a grid
```

```
aa = net(pp);
% translate output into [-1,1]
%aa = -1 + 2*(aa>0);
% plot classification regions
figure(1)
mesh(P1,P2,reshape(aa,length(span),length(span))-5);
colormap cool
view(2)
```

Le traitement de ces données avec le logiciel Matlab donne la figure suivante (4.8) :

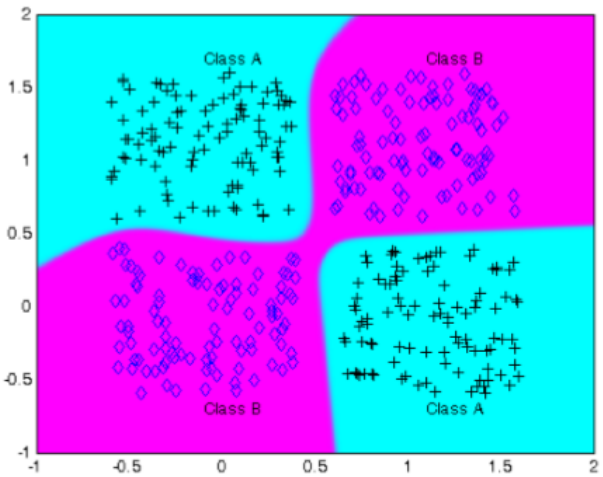


FIGURE 4.8 – Résultas de la classification

4.3.2 Classification avec algorithme de descente de gradient

Classification des secteurs avec l'algorithme de descente de gradient :
Etape 1 :Définition des secteurs

Listing 4.2 – Code source .

```
% Parameter
N = 24;
% Coordinates
V = linspace(-2,+2,N);
[X,Y] = ndgrid(V,V);
Coordinates = [X(:)';Y(:)'];
% Regions
Regions = 1*ge(Y,X)+2*(gt(Y,-X)|lt(abs(X+Y),1e-2))+1;
Regions = Regions(:)';
% Figure
Figure = figure('Color','w');
axis('off');
hold('on');
Colors = {'m','g','c','y'};
% Plot
for r = 1:4
    I = find(eq(Regions,r));
    plot(Coordinates(1,I),Coordinates(2,I),'o',...
        'MarkerEdgeColor',  'k',...
```

```
        'MarkerFaceColor',  Colors{r},...
        'MarkerSize',       6);
end
xlim([-2 +2]);
ylim([-2 +2]);
axis('equal');
drawnow();
```

Etape 2 : Entraînement avec algorithme de descente de gradient

Listing 4.3 – Code source .

```
% Probabilities
Probabilities = zeros(4,numel(Regions));
for n = 1:numel(Regions)
    Probabilities(Regions(n),n) = 1;
end
% Multi-layer perceptron
MLP = ...
    MultiLayerPerceptron('LengthsOfLayers', [2 4 4],...
                         'HiddenActFcn',     'linear',...
                         'OutputActFcn',     'softmax');
% Options
Options = ...
    struct('TrainingAlgorithm', 'GD',...
           'NumberOfEpochs',    50,...
           'MinimumMSE',        1e-2,...
           'SizeOfBatches',     10,...
           'SplitRatio',        1,...
           'Momentum',          0.9);
% Training
MLP.train(Coordinates,Probabilities,Options);
```

Etape 3 : Tracé de la carte en fonction des régions calculées par le perceptron multicouche

Listing 4.4 – Code source .

```
% Map colors
Colors = ...
    [1 0 0;...
     0 1 0;...
     0 0 1;...
     1 1 0];
% Coordinates of the coloration points
N2 = 300;
V = linspace(-2,+2,N2);
[X,Y] = ndgrid(V,V);
Coordinates2 = [X(:)';Y(:)'];
% Most probable regions
MLP.propagate(Coordinates2);
Probabilities2 = MLP.Outputs;
[~, Regions2] = max(Probabilities2,[],1);
% Coloration
Coloration = ...
    pcolor(get(Figure,'currentaxes'),...
           reshape(Coordinates2(1,:),N2,N2),...
```

```
            reshape(Coordinates2(2,:),N2,N2),...
            reshape(Regions2,N2,N2));
set(Coloration,...
    'EdgeColor','none',...
    'FaceAlpha',0.25);
colormap(Colors);
uistack(Coloration,'bottom');
```

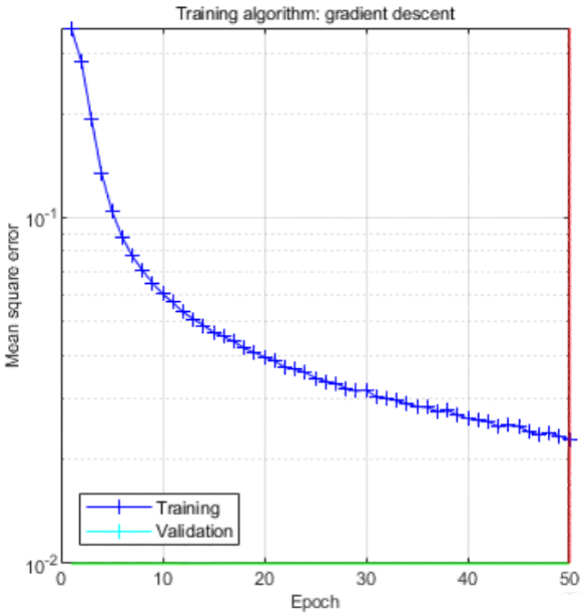


FIGURE 4.9 – L'entraînement de l'algorithme descente de gradient

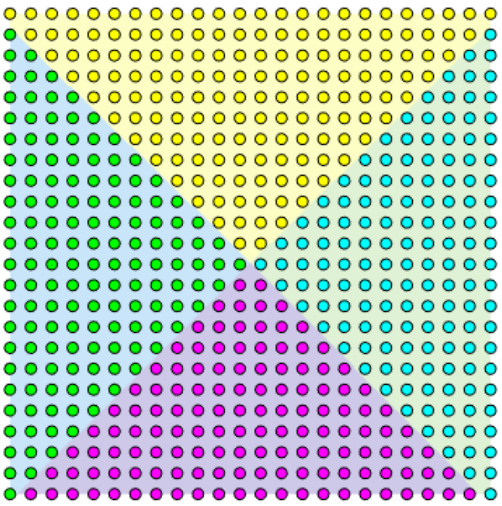


FIGURE 4.10 – Le résultat de classification

4.4 Algorithmes d'optimisation

4.4.1 Colonie d'abeilles artificielles ABC

L'algorithme de colonie d'abeilles artificielles est développé en modélisant le comportement de recherche de miel des abeilles.
Il est conçu par Karaboğa en 2005. Dans cet algorithme, chacune des sources de nourriture représente une solution pour le problème. Les abeilles de la colonie recherchent la source avec plus de quantité de nectar.
La quantité de nectar correspond à la valeur de fitness de la source de nourriture. L'algorithme est divisé par ces étapes :

ETAPE 1 : Initialisation En simple : N abeilles éclaireuses sont placées aléatoirement dans l'espace de recherche.
ETAPE 2 : Recrutement En simple : Les éclaireuses qui ont visité les emplacements avec le meilleur fitness font la danse frétillante : elles recrutent ainsi des butineuses qui auront pour but de chercher dans le voisinage une solution plus prometteuse. Les éclaireuses qui ont trouvé les meilleurs emplacements/solutions lors de la phase d'initialisation recrutent plus de butineuses tandis que les autres recrutent les butineuses restantes.
ETAPE 3 : Recherche locale En simple : Les butineuses actives recherchent dans le voisinage (précédemment visité par les éclaireuses) de nouvelles sources. Les abeilles butineuses recrutées sont aléatoirement dispersées dans le voisinage de la source initiale. Si l'une d'elles trouve une source avec un meilleur fitness que la précédente, elle devient éclaireuse.
ETAPE 4 : Rétrécissement de voisinage Si aucune butineuse ne trouve de solution avec un meilleur fitness que celle de la source (emplacement de l'eclaireuse), la taille du patch fleurs est rétréci. Les patchs sont généralement initialisés pour couvrir une grande région de l'espace de recherche, souvent tout l'espace, est progressivement réduits. Le mécanisme de rétrécissement du voisinage vise à se concentrer progressivement sur une zone de recherche étroite autour du pic de fitness. A chaque cycle de stagnation, la taille du patch fleur est diminuée en utilisant la formule heuristique suivante : $a(t+1) = 0,8a(t)$ où $a(t)$ est le côté de l'hyperbox.
ETAPE 5 : L'abandon du site Si la procédure de recherche locale ne parvient pas à apporter une amélioration dans ce patch fleur, la recherche est supposée avoir trouvé l'optimum de fitness locale. Dans ce cas, le patch de fleurs est abandonné et une nouvelle abeille éclaireuse est réinitialisée à un endroit aléatoire dans l'espace de recherche.
ETAPE 6 : Recherche globale A chaque cycle de l'algorithme des abeilles, la recherche globale est effectuée par ns - nb abeilles éclaireuses. Ces éclaireuses sont placées de manière aléatoire dans l'espace de recherche.

Mise à jour de la population À la fin de chaque cycle d'optimisation, une nouvelle liste d'éclaireuses est crée à partir des nb éclaireuses résultantes de la procédure de recherche locale (pour chaque patch fleur, l'abeille qui a atterri sur la meilleure solution), et les ns - nb éclaireuses résultantes de la procédure de recherche globale (les abeilles dispersées au hasard).

Critère d'arrêt L'algorithme est arrêté, soit quand une solution considérée comme acceptable a été trouvée, soit quand le nombre maximum de cycles a été atteint. Pour expliquer le principe de cet algorithme, voici un schéma bloc présentant ses différentes étapes :

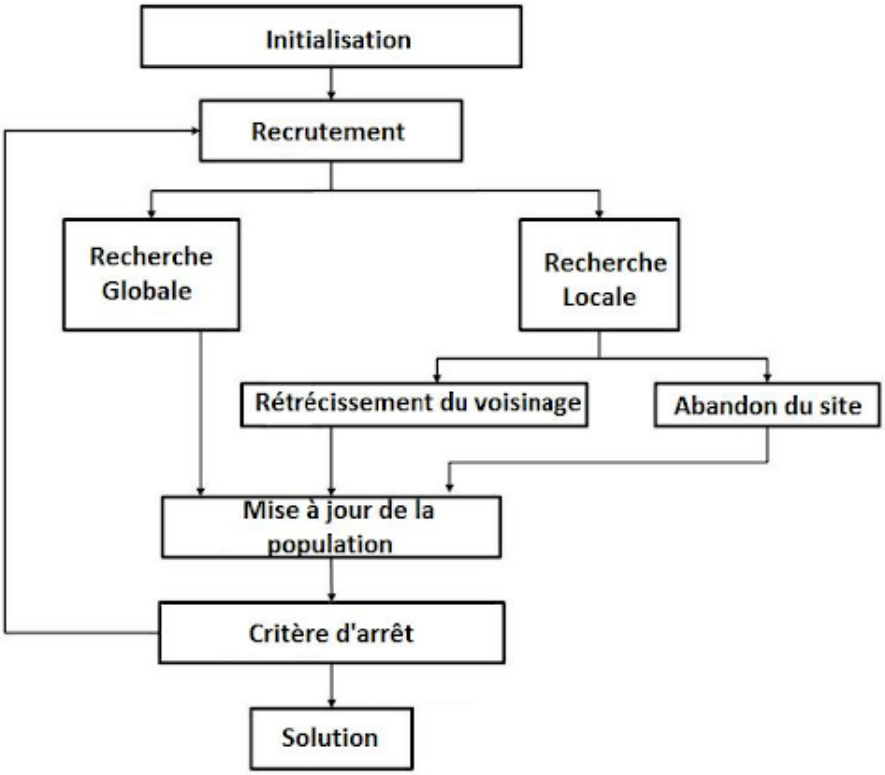


FIGURE 4.11 – Organigramme générale de l'algorithme de Colonie d'abeilles artificielles

4.4.2 Algorithme génétique

Les algorithmes génétiques sont les méthodes d'optimisation et de recherche évolutive les plus courante ,il s'appuyant sur des techniques dérivées de la génétique et de l´évolution naturelle,ils appartiennent à la famille des algorithmes évolutionnistes : un sous-ensemble des métaheuristiques.
ils ont été initialement développés et adaptés à l'optimisation par John Holland en 1975.
Un Algorithme Génétique est défini par les caractéristiques suivantes :

- Séquence/Chromosome/Individu (Codage binaire) : « Nous appelons une séquence (chromosome, individu) A de longueur L(A) une suite A = a1,a2,...,aL avec i [1,L], aiV = 0,1 ».
  Un chromosome est donc une suite de bits en codage binaire, appelé aussi chaîne binaire.
- Population : Un ensemble de chromosomes ou de points de l'espace de recherche.
- Gène : C'est une partie d'une solution au problème, donc d'un individu.
- Environnement : C'est l'espace de recherche.
- Fonction de fitness :C'est la fonction - positive - que nous cherchons à maximiser.

L'algorithme comporte quatre étapes principales : sélection, recombinaison, croisement et mutations.
La sélection sélectionne les individus, appelés parents, qui contribuent à la population de la génération suivante. Crossover combine deux parents pour former les enfants de la prochaine génération.
La mutation applique des changements aléatoires aux parents individuels pour former des enfants.
L'organigramme fonctionnel de la figure 4.12 illustre la structure de l'Algorithme Génétique. Les diverses phases et les mécanismes associés à chacune d'entre elles seront présentés dans les sections suivantes :

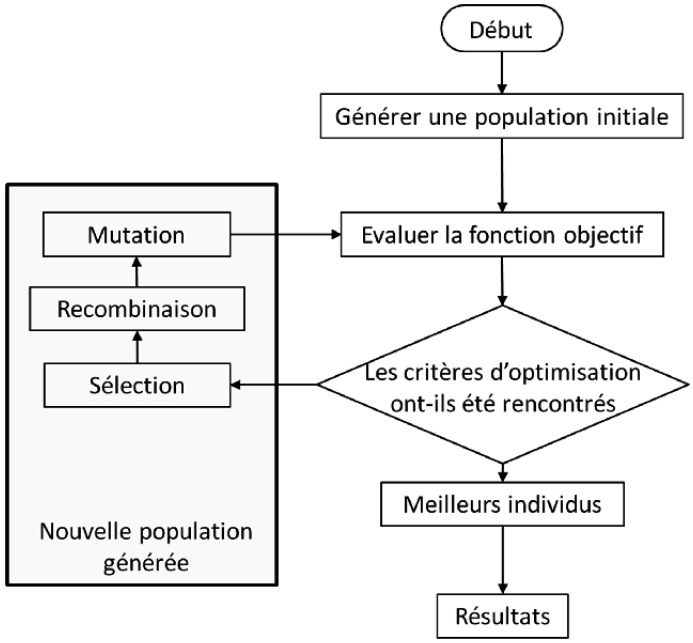


FIGURE 4.12 – Organigramme générale d'un algorithme génétique

4.4.3 Algorithme de recuit simulé

L'algorithme de recuit simulé est développé par le comportement des systèmes physiques en faisant fondre une substance et en abaissant lentement sa température jusqu'à ce qu'elle atteigne le point de congélation (recuit physique).
RS diminue lentement la température et à chaque température considère un voisin de l'état actuel. Si l'énergie du voisin est meilleure que l'état actuel, RS déplace le système vers l'état voisin, mais si l'énergie du voisin était inférieure à l'état actuel, RS utilise une fonction d'acceptation pour décider de passer ou non à l'état voisin. La procédure est contrôlée par un groupe de paramètres appelés programme de refroidissement, de sorte qu'elle peut donner un résultat presque optimal au problème dans un délai raisonnable [26]. L'algorithme accepte une nouvelle solution avec la probabilité p_t :

$$p_t(a \Rightarrow a') = \begin{cases} 1 \quad si \quad f(a') \leq f(a) \\ exp(\frac{f(\alpha)-f(\alpha')}{t}), \quad sinon \end{cases} \tag{4.2}$$

Dans la formule (4.2), f est la fonction objectif, α est la solution courante, α' est une nouvelle solution, t est la température courante [45].

les étapes de l'algorithme RS montre comme suit [46] :

Algorithm 4.8 *Algorithme récuit simulé*

— Sélectionnez le meilleur vecteur de solution x_0 à optimiser
— Initialiser les paramètres : température T, constante de Boltzmann k, facteur de réduction c.

1. **Tant que** le critère de résiliation n'est pas satisfait **Faire**
2. **Pour** nombre de nouvelle solution
   Sélectionnez une nouvelle solution : $x_0 + \Delta x$
3. **Si** $f(x_0 + \Delta x) > f(x_0)$ **Alors**
   $f_{nouvel} = f(x_0 + \Delta x); x_0 = x_0 + \Delta x$
4. **Sinon**
   $\Delta f = f(x_0 + \Delta x) - f(x_0)$
   aléatoire $r(0, 1)$
5. **Si** $r > exp(-\Delta f/KT)$ **Alors**
   $f_{nouveau} = f(x_0 + \Delta x), \quad x_0 = x_0 + \Delta x$
6. **Sinon**
   $f_{nouveau} = f(x_0)$
7. **Fin Si**
8. **Fin Si**
   $f = f_{nouveau}$ Baisser la température périodiquement : $T = c \times T$
9. **Fin Pour**
10. **Fin Tant que**

4.5 Processus d'optimisation

Le premier ensemble de données des deux utilisé dans l'ensemble de données de recherche sur la reconnaissance des activités humaines et des transitions posturales par smartphone de l'ensemble de données du référentiel d'apprentissage automatique de l'UC Irvine contient 10929 échantillons avec 561 fonctionnalités. Ensemble de données de reconnaissance d'activité construit à partir des enregistrements de 30 sujets effectuant des activités de base et des transitions posturales tout en portant un smartphone monté à la taille avec des capteurs inertiels intégrés. Dans la recherche, 6 classes, marcher, monter les escaliers, descendre les escaliers, s'asseoir, se tenir debout et s'allonger, sont utilisées. Les 6 classes contiennent environ 95 de l'ensemble de données. 65 de l'ensemble de données est utilisé pour la formation et le reste est utilisé pour les tests.

Le deuxième jeu de données contient 1500 échantillons avec 45 caractéristiques. Ces caractéristiques sont les valeurs de coordonnées xyz pour la main, la taille et le genou dans une période d'intervalle de 1 seconde dans une activité de cinq secondes. Le nombre de classes de sortie est de 5 : chute, debout, saut, marche et assise. Cet ensemble de données a été collecté pour la recherche menée à l'Université technique de Yildiz d'ici l'année 2014 [29]. 1250 de ces 1500 sont utilisés pour la formation et le reste est utilisé pour les tests.

Pour la topologie du réseau, un perceptron multicouche avec une couche cachée est conçu. Le réseau pour s'entraîner à l'aide de l'algorithme de propagation arrière est la cible. Les poids initiaux sont sélectionnés au hasard. 100 itérations sont implémentées pour la formation. Lorsque la précision du test est calculée, pour le même taux d'apprentissage, le même coefficient d'impulsion, le nombre de neurones cachés et le type de fonction d'activation, la précision du test est calculée 5 fois avec des poids initiaux choisis au hasard, et la moyenne arithmétique

de ces 5 valeurs est calculée. Le seul réseau PMC en couches caché formé pour classer ces ensembles de données est optimisé respectivement avec ABC, AG et RS. Les paramètres à optimiser sont le nombre de neurones cachés, le taux d'apprentissage, le coefficient d'impulsion et le type de fonction d'activation.

Les options disponibles pour la fonction d'activation sont la fonction sigmoïde, sigmoïde bipolaire, linéaire et échelonnée. Pour l'algorithme ABC, l'optimisation est appliquée par les colonies avec des volumes différents. Le facteur limite est choisi comme 5. La quantité de nectar collecté correspond à la performance du test. Pour AG, l'optimisation est appliquée par différentes tailles et générations de population.

4.6 Résultats

Les résultats classifient l'ensemble de données HAPT sont présentés dans les tableaux (4.1), (4.2) et (4.3) respectivement en utilisant ABC, AG et RS.

Les résultats classent le deuxième ensemble de données sont présentés dans les tableaux (4.4), (4.5) et (4.6) respectivement en utilisant ABC, AG et RS.

Des abeilles	Itération	Taux d'apprentissage	Coeifficient de quantité	Nombre de Neurone caché	Fonction d'activation	Tester les Performance
6	10	0.4474	0.6329	111	Linéare	95.5%
8	8	0.2909	0.0316	596	sigmoid	93.67%
10	5	0.4111	0.4966	309	sigmoid	93.5%
12	3	0.4004	0.7487	71	linéar	95.33%
14	2	0.7241	0.312	46	linéar	95.33%

TABLE 4.1 – Ensemble de données HAPT avec algorithme ABC.

Population	Génération	Taux d'apprentissage	Coeifficient de quantité de mouvement	Nombre de Neurone caché	Fonction d'activation	Tester les Performance
5	10	0.4295	0.5805	12	sigmoid	88%
10	15	0.2199	0.6598	112	linear	90.5%
10	15	0.4268	0.5321	151	sigmoid	91.5%
15	15	0.1012	0.6410	458	sigmoid	92.16%
15	20	0.411	0.4966	309	sigmoid	93.5%

TABLE 4.2 – Ensemble de données HAPT avec algorithme génétique AG.

Itération	Taux d'apprentissage	Coeifficient de quantité	Nombre de Neurone caché	Fonction d'activation	Tester les Performance
5	0.5	0.5	200	linear	90.67%
10	0.515	0.417	265	linear	92%
15	0.533	0.429	252	linear	92.16%
20	0.889	0.026	22	sigmoid	93.33%
40	0.646	0.354	208	sigmoid	93.66%

TABLE 4.3 – Ensemble de données HAPT avec l'algorithme RS.

Des abeilles	Itération	Taux d'apprentissage	Coeifficient de quantité	Nombre de Neurone Number	Fonction d'activation caché	Tester les Performance
6	12	0.441	0.2932	101	step	86.6%
8	9	0.0117	0.4553	390	step	87.4%
10	8	0.0074	0.2482	23	step	87.2%
12	6	0.0052	0.6278	5	sigmoid	86.4%
14	5	0.0074	0.2482	237	step	87.2%

TABLE 4.4 – Deusième ensemble de données avec l'algorithme ABC.

Population	Génération	Taux d'apprentissage	Coeifficient de quantité de mouvement	Nombre de Neurone caché	Fonction d'activation	Tester les Performance
5	10	0.4604	0.7091	187	linéar	74.6%
10	10	0.6280	0.4624	351	Sigmoid	75.4%
10	15	0.0011	0.7892	320	Sigmoid	78.2%
15	15	0.0019	0.6476	265	Sigmoid	81.8%
15	20	0.0052	0.6278	5	Sigmoid	81.4%

TABLE 4.5 – Deusième ensemble de données avec l'algorithme génétique AG.

Itération	Taux d'apprentissage	Coeifficient de quantité	Nombre de Neurone caché	Fonction d'activation	Tester les Performance
5	0.549	0.549	58	linéar	73.04%
10	0.164	0.847	223	Sigmoid	75.18%
15	0.011	0.043	32	linéar	77.2%
20	0.974	0.06	4	Sigmoid	78.6%
40	0.909	0.081	324	Sigmoid	78.8%

TABLE 4.6 – Deuxième ensemble de données avec l'algorithme RS.

Pour l'algorithme ABC, l'itération s'est poursuivie jusqu'à ce que le taux de réussite le plus élevé soit atteint dans les colonies contenant un nombre différent d'abeilles. A la fin, on ne voit que dans les colonies avec un plus grand nombre d'abeilles ; le taux de réussite le plus élevé est atteint avec moins d'itérations.

Le nombre maximal d'itérations a été sélectionné pour les critères d'arrêt dans l'AG. Des processus d'optimisation ont été lancés pour différentes valeurs de population. Dans l'algorithme RS, la température initiale a pris 100. Ici encore, le critère d'arrêt est sélectionné comme le nombre d'itérations. Au final, l'AG est le plus lent en terme de temps d'exécution. L'algorithme RS, en revanche, a bien fonctionné en termes de vitesse mais n'a pas pu obtenir des résultats bonnes par rapport aux autres algorithmes. Par conséquent, l'algorithme ABC a été le plus réussi en termes d'obtention des meilleurs résultats.

4.7 Conclusion

Dans ce chapitre, on a donné une vision générale sur les méthodes ou approches de la classification supervisées.
Nous avons présenté aussi quelques algorithmes ou méthodes de classification (KNN, les réseaux de neurones, machine à vecteur support).

Puis nous avons utilisé une méthode efficace pour optimiser le perceptron multicouche par des algorithmes d'optimisation. Les trois algorithmes sont comparés et des résultats de test détaillés sont fournis. Il est observé que, bien que RS soit l'algorithme le plus rapide parmi les algorithmes choisis, l'algorithme ABC affiche les performances les plus élevées pour la précision des tests du classificateur PMC.

Conclusion générale

Depuis une dizaine d'années, l'utilisation des réseaux de neurones artificiels (RNA) s'est développée dans de nombreuses disciplines (sciences économiques, écologie et environnement, biologie et médecine...). Ils sont notamment appliqués pour résoudre des problèmes de classification, de prédiction, de catégorisation, d'optimisation, de reconnaissance des formes et de mémoire associative .

Dans le cadre du traitement des données, les RNA constituent une méthode d'approximation de systèmes complexes, particulièrement utile lorsque ces systèmes sont difficiles à modéliser à l'aide des méthodes statistiques classiques. Les RNA sont également applicables dans toutes les situations où il existe une relation non linéaire entre une variable prédictive et une variable prédite. Par leur nature et leur fonctionnement, les RNA peuvent détecter les interactions multiples non linéaires parmi une série de variables d'entrée, ils peuvent donc gérer des relations complexes entre les variables indépendantes et les variables dépendantes .

les réseaux de neurones sont les modèles et les méthodes d'apprentissage automatique les plus connus visant à fournir des algorithmes et des logiciels capables d'apprendre par l'expérience, car ils entrent dans le domaine de l'apprentissage automatique et de l'intelligence artificielle, qui est inclus dans tous les domaines qui nous entourent, de la communication sociale à la pathologie des systèmes de diagnostic dans les hôpitaux. L'intelligence artificielle a appliqué des réseaux de neurones biologiques qui fonctionnent avec succès sur la reconnaissance vocale, l'analyse d'image et le contrôle machine et robotique...

Le but principal de ce mémoire est l'étude des réseaux de neurones artificielles dans le cadrel de classification supervisé des données par le réseau PMC qui est optimisé par des algorithmes mathématique(recuit simulé, algorithme génétique, colonies d'abeilles artificielles).

Bibliographie

[1] B. H. Ateme-Nguema. *Conception optimale des cellules de fabrication flexibles basée sur l'approche par réseaux de neurones*. PhD thesis, École de technologie supérieure, 2007.

[2] C.-A. Azencott. *Introduction au machine learning*. Dunod, 2019.

[3] N. Benahmed. *Optimisation de réseaux de neurones pour la reconnaissance de chiffres manuscrits isolés : Sélection et pondération des primitives par algorithmes génétiques*. École de technologie supérieure, 2002.

[4] M. Benrahmoune. Diagnostic des défaillances d'une turbine à gaz à base des réseaux de neurones artificiels pour l'amélioration de leur système de détection des vibrations. *Djelfa : Université Ziane Achour*, 2017.

[5] P. J. Braspenning, F. Thuijsman, and A. J. M. M. Weijters. *Artificial neural networks : an introduction to ANN theory and practice*, volume 931. Springer Science & Business Media, 1995.

[6] A. Ciampi and Y. Lechevallier. Réseaux de neurones et modèles statistiques. *Monde des Util. Anal. Données*, 15 :27–46, 1995.

[7] P. Comon. Classification supervisée par réseaux multicouches. *Traitement du signal*, 8(6) :387–407, 1991.

[8] M. Cottrell. Les réseaux de neurones : historique, méthodes et applications. *Université Paris*, 2003.

[9] B. Derras. *Contribution des données accélérométriques de KiKNet à la prédiction du mouvement sismique par l'approche neuronale avec la prise en compte des effets de site*. PhD thesis, Université de Tlemcen, 2011.

[10] S. Djaaoui. Optimisation des réseaux de neurones par pso. 2015.

[11] G. Dreyfur. Réseaux de neurones : Méthodologie et application, 2004.

[12] G. Dreyfus, J. Martinez, M. Samuelides, M. Gordon, F. Badran, S. Thiria, and L. Hérault. *Réseaux de neurones*, volume 39. Eyrolles Paris, 2002.

[13] M. Fatima and S. Hamid. Comparaison de méthodes de classification réseau rbf, mlp et rvflnn1. *Damascus University Journal Vol,(25)-No.(2)*, 2009.

[14] M. Friedman and A. Kandel. *Introduction to pattern recognition : statistical, structural, neural and fuzzy logic approaches*, volume 32. World Scientific Publishing Company, 1999.

[15] S. E. GACEM. *IDENTIFICATION DES SYSTEMES NON LINEAIRES PAR RESEAUX DE NEURONES*. PhD thesis, Université Mohamed Khider–Biskra, 2015.

[16] R. Gilleron. Apprentissage automatique, les réseaux de neurones, groupe de recherche sur l'apprentissage automatique, université de lille, 2007.

[17] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[18] D. E. Goldberg. Genetic algorithms in search, optimization, and machine learning. addison. *Reading*, 1989.

[19] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[20] A. Hafaifa, M. Guemana, and A. Daoudi. Fault detection and isolation in industrial systems based on spectral analysis diagnosis. 2013.

[21] S. S. Haykin et al. Neural networks and learning machines/simon haykin., 2009.

[22] S. HEDDAM. *Contribution à la modélisation de la qualité des eaux par les réseaux des neurones*. PhD thesis, INA, 2006.

[23] N. M. Héritier and I. B. Nephtali. L'algorithme de rétro-propagation de gradient dans le perceptron multicouche : Bases et étude de cas. *International Journal of Innovation and Applied Studies*, 32(2) :271–290, 2021.

[24] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8) :2554–2558, 1982.

[25] M. Idir. *Implémentation d'un réseau de neurones d'un micro capteur sur un FPGA*. PhD thesis, Université Mouloud Mammeri, 2010.

[26] M. M. Keikha. Improved simulated annealing using momentum terms. In *2011 Second International Conference on Intelligent Systems, Modelling and Simulation*, pages 44–48. IEEE, 2011.

[27] P. Kim. Matlab deep learning. *With machine learning, neural networks and artificial intelligence*, 130(21), 2017.

[28] M. Kordos, W. Duch, et al. Variable step search mlp training method. *International Journal of Information Technology and Intelligent Computing*, 1 :45–56, 2006.

[29] O. C. Kurban. İnsan aktivitelerinin giyilebilir sensörler ile özellik çıkarımı yapılmadan sınıflandırılması. 2014.

[30] Z. Liao. *Théorie des matrices aléatoires pour l'apprentissage automatique en grande dimension et les réseaux de neurones*. PhD thesis, Université Paris-Saclay (ComUE), 2019.

[31] T. B. Ludermir, A. Yamazaki, and C. Zanchettin. An optimization methodology for neural network weights and architectures. *IEEE Transactions on Neural Networks*, 17(6) :1452–1459, 2006.

[32] P. Martin. *Réseaux de neurones artificiels : Application à la reconnaissance optique de partitions musicales*. PhD thesis, Université Joseph-Fourier-Grenoble I, 1992.

[33] R. Melki. Apprentissage des réseaux de neurones mlp par une méthode hybride à base d'une métaheuristique. 2019.

[34] Y. Merabti. Optimisation des réseaux de neurones mlp par l'algorithme hybride ag-rt pour le contrôle d'un système non linéaire. 2015.

[35] B. Mohammed and B. Brahim. Lapprentissage profond (deep learning) pour la classification et la recherche d'images par le contenu. *UNIVERSITE KASDI MERBAH OUARGLA Faculté des Nouvelles Technologies de l'Information et de la Communication*, 2017.

[36] M. Msaaf and F. Belmajdoub. L'application des réseaux de neurone de type «feedforward» dans le diagnostic statique. In *Xème Conférence Internationale : Conception et Production Intégrées*, 2015.

[37] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions : Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv :1811.03378*, 2018.

[38] M. Parizeau. Le perceptron multicouche et son algorithme de rétropropagation des erreurs. *département de génie électrique et de génie informatique, Université de laval*, 2004.

[39] J.-W. Park, R. G. Harley, and G. K. Venayagamoorthy. Comparison of mlp and rbf neural networks using deviation signals for on-line identification of a synchronous generator. In *2002 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No. 02CH37309)*, volume 1, pages 274–279. IEEE, 2002.

[40] L. Personnaz and I. Rivals. *Réseaux de neurones formels pour la modélisation, la commande et la classification*. CNRS, 2003.

[41] G. PETITJEAN. Dossier : l'intelligence artificielle et l'aide à la décision dans les entreprises.

[42] L. Pibre, M. Chaumont, D. Ienco, and J. Pasquet. Étude des réseaux de neurones sur la stéganalyse. In *CORESA : COmpression et REprésentation des Signaux Audiovisuels*, 2016.

[43] E. Poisson, C. Viard-Gaudin, and P. Lallican. Réseaux de neurones à convolution : reconnaissance de l'écriture manuscrite non contrainte. *Revue VALGO, In Valgo*, (01-02) :726–730, 2001.

[44] P. Potocnik. Neural networks : Matlab examples. *Neural Network course (practical Examples)*, 2012.

[45] J.-Y. Qi. Application of improved simulated annealing algorithm in facility layout design. In *Proceedings of the 29th Chinese Control Conference*, pages 5224–5227. IEEE, 2010.

[46] L. R. Rere, M. I. Fanany, and A. M. Arymurthy. Simulated annealing algorithm for deep learning. *Procedia Computer Science*, 72 :137–144, 2015.

[47] M. Taşkıran, Z. G. Çam, and N. Kahraman. An efficient method to optimize multi-layer perceptron for classification of human activities. In *2nd International Conference on Computer, Control and Communication Technologies (CCCT'15), Antalya, Turkey*, pages 3–4, 2015.

[48] L. Thiaw. *Identification de systemes dynamiques non linéaires par réseaux de neurones et multimodeles*. PhD thesis, 2008.

[49] C. Touzet. *les réseaux de neurones artificiels, introduction au connexionnisme*. Ec2, 1992.

[50] M. Watts, L. Major, and W. Tate. Evolutionary optimisation of mlp for modelling protein synthesis termination signal efficiency. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 1, pages 193–198. IEEE, 2002.

[51] B. Widrow and M. A. Lehr. 30 years of adaptive neural networks : perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9) :1415–1442, 1990.

[52] A. ZAHIR. *Identification des systemes non lineaires par les reseaux de neurones*. PhD thesis, Université de Béjaia-Abderrahmane Mira, 2009.

[53] S. F. P. Zaki. *Classification par réseaux de neurones dans le cadre de la scattérométrie ellipsométrique*. PhD thesis, Lyon, 2016.

[54] W. Zhu, C. Guicheney, J.-L. Berdagué, and J. Jousset. Application des réseaux perceptron multicouches au contrôle de la qualité des aliments par analyse sensorielle. comparaison des résultats avec différentes méthodes d'analyse discriminante. *Revue de statistique appliquée*, 45(2) :39–57, 1997.