REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre :IVA2/M2/2021

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Image et Vie Artificielle (IVA)**

Entrainement d'un réseau de neurones par algorithme génétique

---

Par :

KHODHEIR Thana

Soutenu le ../../…. devant le jury composé de :

Mokhtari Bilal	MCA	Président
Chighoub Rabia	MCB	Rapporteur
Ababsa Tarek	MCB	Examinateur

Année universitaire 2020-2021

Remerciement

Je remercie ALLAH de m'avoir donné le courage ainsi que la force pour pouvoir achever ce modeste travail.

Je tiens tous d'abord à remercier mon encadreur Dr TAREK ABABSA pour sa patience, ses encouragements, ses conseils, et le temps précieux quelle ma accordé pour l'achèvement de ce travaille.

Je tiens à remercier également les membres de jury.

Que tout enseignant nous ayant fait bénéficier de son savoir, trouve ici l'expression de notre profonde gratitude.

Je remercie mes parents pour m'avoir transmit des valeurs telles que la ténacité, grâce auxquelles j'ai pu mener ce travail à son aboutissement.

Aux chères personnes qui m'ont toujours soutenu et accompagné tout au long de mon parcours universitaire.

Dédicace

Je dédie ce modeste travail à:

A mes très chers parents, pour l'incommensurable contribution à mon éducation, à mon instruction et qui m'ont encouragé et guidé vers la réussite tous les instants de ma vie.

Mes sœurs et mes frères.

Toute la promotion.

Tous mes amis.

Toute la famille.

A tous ceux qui

me sont chers.

Table des matières

Liste des figures

Introduction générale

Les techniques avancées de l'automatique issues de l'intelligence artificielle deviennent de plus en plus familières dans divers domaines d'application ces dernières années. L'intelligence artificielle est une discipline scientifique relative au traitement des connaissances et au raisonnement, dans le but de permettre à une machine d'exécuter des fonctions associées à l'intelligence humaine telles que la perception, l'apprentissage, et le raisonnement.

Les outils intelligents sont récemment utilisés dans la conception, la modélisation et la commande de systèmes complexes. On entend par outils intelligents les techniques du soft computing à savoir : les réseaux de neurones, la logique floue et les métaheuristiques.

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. Le développement des réseaux de neurones artificiels est issu d'une volonté des chercheurs de comprendre et d'imiter les capacités du cerveau, mémoire, apprentissage, et traitement parallèle pour la synthèse de systèmes artificiels capables de remplacer l'homme dans la réalisation de tâches complexes.

Grâce aux résultats théoriques et pratiques obtenus au cours des dernières décennies, les réseaux de neurones sont devenus un outil très utilisé dans divers domaines. Ils ont prouvé leur efficacité dans la commande des procédés, l'identification des paramètres, la reconnaissance de formes et le traitement de signal.... Ils demeurent toutefois un sujet d'un grand intérêt pour les chercheurs qui désirent améliorer leurs performances et étendre leur champ d'applications.

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, en y trouve les algorithmes génétiques , quels sont des algorithmes stochastiques adaptés à l'exploration rapide et globale d'un espace de recherche de taille importante en optimisant une fonction quelconque.

Dans ce mémoire on s'intéresse aux Réseaux de Neurones Multicouches , qui sont souvent utilisés, et ceci est dû à leurs simplicités et leurs propriétés d'approximation universelle.

L'objectif de notre travail est d'appliquer les métaheuristiques plus précisément les algorithmes génétiques pour l'optimisation des réseaux de neurones contrôleurs, en créant un mélange entre eux en combinant les avantages de chacun pour augmenter l'efficacité des méthodes développées.

Dans notre travail on utilise un mécanisme évolutif pour surmonter la difficulté de l'entrainement des réseaux de neurones.

Ce mémoire est organisé comme suit :

Le premier chapitre est consacré à la description des réseaux de neurones, les différentes architectures de réseaux et les méthodes d'apprentissages les plus utilisées.

le deuxième chapitre étudie les algorithmes génétiques et leurs caractéristiques.

Le troisième chapitre propose la conception du système et détaille ses composants et quelques résultats expérimentaux.

Chapitre 1

Les réseaux de neurones

1.1 Historique

Les recherches menées dans le domaine du connexionnisme ont démarré avec la présentation en 1943 par W. MCCulloch et W. Pitts d'un modèle simplifié de neurone biologique communément appelé neurone formel . Ils montrèrent également théoriquement que des réseaux de neurones formels simples peuvent réaliser des fonctions logiques, arithmétiques et symboliques complexes.

En 1949, D. Hebb initie, dans son ouvrage "The Organization of Behavior", la notion d'apprentissage. Deux neurones entrant en activité simultanément vont être associés (c'est-à dire que leurs contacts synaptiques vont être renforcés). On parle de loi de Hebb et d'associationnisme.

En 1958, F. Rosenblatt développe le modèle du Perceptron. C'est un réseau de neurones inspiré du système visuel. Il possède deux couches de neurones une couche de perception (sert à recueillir les entrées) et une couche de décision. C'est le premier modèle pour lequel un processus d'apprentissage a pu être défini.

S'inspirant du perceptron , Widrow et Hoff , développent , dans la même période , le modèle de l'Adaline (Adaptive Linear Element). Ce dernier sera, par la suite, le modèle de base des réseaux de neurones multi-couches.

En 1969, Les recherches sur les réseaux de neurones ont été pratiquement abandonnées lorsque M. Minsky et S. Papert ont publié leur livre Perceptrons (1969) et démontré les limites théoriques du perceptron , en particulier , l'impossibilité de traiter les problèmes non linéaires par ce modèle.

En 1982, Hopfield développe un modèle qui utilise des réseaux totalement connectés basés sur la règle de Hebb pour définir les notions d'attracteurs et de mémoire associative. En 1984 c'est la découverte des cartes de Kohonen avec un algorithme non supervisé basé sur l'auto-organisation et suivi une année plus tard par la machine de Boltzman (1985).

Une révolution survient alors dans le domaine des réseaux de neurones artificiels une nouvelle génération de réseaux de neurones, capables de traiter avec succès des phénomènes non-linéaires le perceptron multicouche ne possède pas les défauts mis en évidence par Minsky. Proposé pour la première fois par Werbos, le Perceptron Multi-Couche apparaît en 1986 introduit par Rumelhart, et, simultanément, sous une appellation voisine, chez Le Cun (1985). Ces systèmes reposent sur la rétropropagation du gradient de l'erreur dans des systèmes à plusieurs couches, chacune de type Adaline de Bernard Widrow, proche du Perceptron de Rumelhart (1).

1.2 Neurone biologique

1.2.1 Neurone

Le neurone est l'élément de base du système nerveux. Les neurones récepteurs périphériques du système nerveux centrale transforment les signaux extérieurs du monde physique (la lumière, la chaleur, le son) en impulsions électrique qui vont se propage, en fonction des informations qu'elle reçoit, des signaux électroniques et biologiques. Cette transformation se fait de manière électronique le long du neurone et essentiellement de manière biochimique, par l'intermédiaire de neurotransmetteurs, entre les neurones. La sécrétion de ces neurotransmetteurs se fait de manière très focalisée vers les cellules avec lesquelles le neurone est connecté par des synapses.

1.2.2 Structure de neurone

1.2.2.1 Le corps cellulaire (soma)

Contient le noyau du neurone et effectue les transformations biochimiques nécessaires à la synthèse des enzymes et des autres molécules qui assurent la vie de la cellule. Ce corps cellulaire ayant une forme sphérique ou pyramidale, sa taille est de quelques microns de diamètre.

1.2.2.2 Les dendrites

Sont de fines extensions tubulaires qui se ramifient autour du neurone et forment une sorte de vaste arborescence. Les signaux envoyés au neurone sont captés par les dendrites. Leur taille est de quelques dizaines de microns de longueur.

1.2.2.3 L'axone

Est la fibre nerveuse, sert de moyen de transport pour les signaux émis par le neurone. Il est plus long que les dendrites, et se ramifie à son extrémité où il se connecte aux dendrites des autres neurones. Les connexions entre deux neurones se font en des endroits appelés synapses où ils sont séparés par un peu espace synaptique de l'ordre d'un centième de microns.

1.2.3 Neurone formel

Un "neurone formel" (ou simplement "neurone") est une fonction algébrique non linéaire et bornée, 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 sa "sortie".

Un neurone est donc avant tout un opérateur mathématique, dont on peut calculer la valeur numérique par quelques lignes de logiciel. On a pris l'habitude de représenter graphiquement un neurone comme indiqué sur la Figure 1.1 (2).

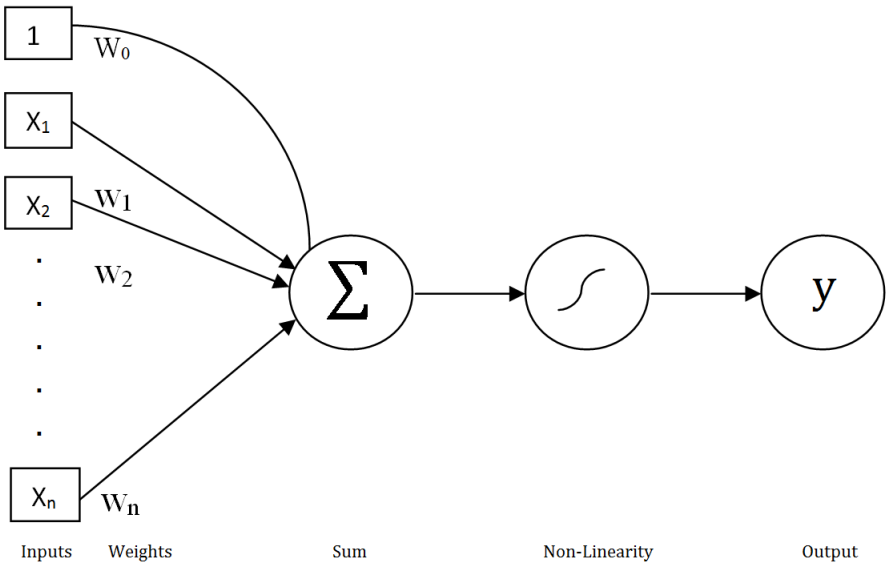


Figure 1.1: Neurone formel

Tels que :
- (x1, x2,. . . , xn) : sont les entrées du neurone.
- (w1, w 2, ..., w n): les poids associés à chaque connexion.
- w 0 : terme de biais.
- Sum : la somme pondérée des entrées

$$\text{Sum} = w_0 + \sum_{i=1}^{n}(w_i x_i)$$

- g : fonction d'activation non linéaire.

$$g(z) = \sigma(z) = \frac{1}{1+e^{-z}}$$

Un exemple courant : fonction sigmoïde

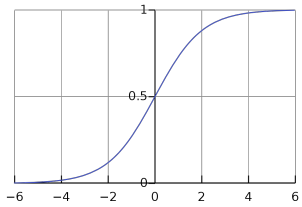


Figure 1.2: fonction sigmoïde

- y : la sortie du neurone :

$$y = g\left(w_0 + \sum_{i=1}^{n}(w_i x_i)\right).$$

1.3 Réseaux de neurones artificiels

1.3.1 Définition

L'origine des réseaux de neurones vient de l'essai de modélisation mathématique du cerveau humain les premiers travaux datent de 1943 et sont l'œuvre de W.M. Culloch et W. Pitts. Ils

supposent que l'impulsion nerveuse est le résultat d'un calcul simple effectué par chaque neurone et que la pensée née grâce à l'effet collectif d'un réseau de neurone interconnecté .

Un réseau de neurones est un assemblage de constituants élémentaires interconnectés (appelés neurones en hommage à leur modèle biologique), qui réalisent chacun un traitement simple mais dont l'ensemble en interaction fait émerger des propriétés globales complexes. Chaque neurone fonctionne indépendamment des autres de telle sorte que l'ensemble forme un système massivement parallèle. L'information est stockée de manière distribuée dans le réseau sous forme de coefficients synaptiques ou de fonctions d'activation, il n'y a donc pas de zone de mémoire et de zone de calcul, l'une et l'autre sont intimement liés.

Un réseau de neurone ne se programme pas, il est entraîné grâce à un mécanisme d'apprentissage. Les tâches particulièrement adaptées au traitement par réseau de neurones sont: l'association, la classification, la discrimination, la prévision ou l'estimation, et la commande de processus complexes.

Les réseaux de neurones artificiels consistent en des modèles plus ou moins inspirés du fonctionnement cérébral de l'être humain en se basant principalement sur le concept de neurone (3).

1.3.2 Domaine d'application des ARN

Les grands domaines d'application des réseaux de neurones découlent naturellement de leur propriété fondamentale :

• la régression non linéaire, ou modélisation de données statiques : il existe une immense variété de phénomènes statiques qui peuvent être caractérisés par une relation déterministe entre des causes et des effets ; les réseaux de neurones sont de bons candidats pour modéliser de telles relations à partir d'observations expérimentales, sous réserve que celles-ci soient suffisamment nombreuses et représentatives .

• la modélisation de processus dynamiques non linéaires : modéliser un processus, c'est trouver un ensemble d'équations mathématiques qui décrivent le comportement dynamique du processus, c'est-à-dire l'évolution de ses sorties en fonction de celle de ses entrées ; c'est donc typiquement un problème qui peut être avantageusement résolu par un réseau de neurones, si le phénomène que l'on désire modéliser est non-linéaire. La prédiction de séries chronologiques (prédictions financières, prédiction de consommation, etc.) entre dans ce cadre.

• la commande de processus : commander un processus, c'est imposer à celui-ci un comportement défini à l'avance en fonction des signaux de commande ; l'ensemble commande + processus peut donc être considéré comme un système qui réalise une fonction (non linéaire) qu'un réseau de neurones peut approcher.

• la classification : supposons que l'on désire classer des formes en deux catégories, A ou B, en fonction de certaines caractéristiques de ces formes ; on peut définir une fonction qui vaut +1 pour toutes les formes de la classe A et -1 pour toutes les formes de la classe B. Les réseaux de neurones sont de bons candidats pour réaliser une approximation de cette fonction , et l'on peut démontrer que cette approximation constitue une estimation de la probabilité d'appartenance de la forme inconnue à la classe A. Les réseaux de neurones fournissent donc une information très riche, qui est loin d'être une simple réponse binaire. Cette propriété remarquable (que les réseaux de neurones partagent avec d'autres classifieurs) n'est malheureusement pas mise à profit dans la plupart des applications(2).

1.3.3 Modélisation générale

On peut modéliser un réseau de neurone par des élémentaires qu'il s'agit de :

• La nature de ses entrées : qu'ils peuvent être binaire (0 ou 1), (-1,1) ou réelles appartenant souvent à intervalle bornée [a, b].

• La fonction des entrées : que sa signifie qu'elle peut définir le pré traitement effectuée sur les entrées.

• Fonction de sortie : Cette fonction calcule la sortie du neurone en fonction de son état d'activation (4).

• Fonction d'activation ou de seuillage : Il se trouve que plusieurs possibilités existent. Différentes fonctions de transfert pouvant être utilisées comme fonction d'activation du neurone sont énumérées au tableau Tab.1. Les trois les plus utilisées sont les fonctions seuil (en anglais hard limit), linéaire et sigmoïde .

La nature de la fonction d'activation de notre modèle, La fonction f. Elle peut être une fonction à seuil, une fonction linéaire ou non linéaire. La fonction sigmoïde se présente comme une approximation continûment dérivable de la fonction d'activation linéaire par morceaux ou de la fonction seuil. Elle présente l'avantage d'être régulière, monotone, continûment dérivable, et bornée entre 0 et 1 (1) :

$$f(x)= \frac{1}{1+e^{-s}}$$

Nom de la fonction	Relation entrée/sortie	Icône
Seuil	$a = 0$ si $n < 0$ $a = 1$ si $n \geq 0$	
seuil symétrique	$a = -1$ si $n < 0$ $a = 1$ si $n \geq 0$	
Linéaire	$a = n$	
linéaire saturée	$a = 0$ si $n < 0$ $a = n$ si $0 \leq n \leq 1$ $a = 1$ si $n > 1$	
Linéaire saturée symétrique	$y = -1$ si $s < -1$ $y = s$ si $-1 \leq s \leq 1$ $y = 1$ si $s > 1$	
Linéaire positive	$y = 0$ si $s \leq 0$ $y = s$ si $s \geq 0$	
Sigmoïde	$y= \frac{1}{1+e^{-s}}$	
Tangente hyperbolique	$y= \frac{e^{s}-e^{-s}}{e^{s}+e^{-s}}$	
Compétitive	$y = 1$ si s maximum $y = 0$ autrement	

Figure 1.3: Différentes fonctions d'activations utilisées dans les RNA

1.3.4 Architecture des réseaux

1.3.4.1 Les réseaux de neurones non bouclés

Un réseau de neurones non bouclé réalise une (ou plusieurs) fonctions algébriques de ses entrées, par composition des fonctions réalisées par chacun de ses neurones. Un réseau de neurones non bouclé est représenté graphiquement par un ensemble de neurones "connectés" entre eux, l'information circulant des entrées vers les sorties sans "retour en arrière"; si l'on représente le réseau comme un graphe dont les nœuds sont les neurones et les arêtes les "connexions" entre ceux-ci, le graphe d'un réseau non bouclé est acyclique. Le terme de "connexions" est une métaphore : dans la très grande majorité des applications, les réseaux de neurones sont des formules algébriques dont les valeurs numériques sont calculées par des programmes d'ordinateurs, non des objets physiques (circuits électroniques spécialisés) ; néanmoins, le terme de connexion, issu des origines biologiques des réseaux de neurones, est passé dans l'usage, car il est commode quoique trompeur. Il a même donné naissance au terme de connexionnisme.

1.3.4.2 Réseaux de neurones monocouches

La structure d'un réseau monocouche est telle que des neurones organisés en entrée soient entièrement connectés à d'autres neurones organisés en sortie par une couche modifiable de poids (figure 1.4).

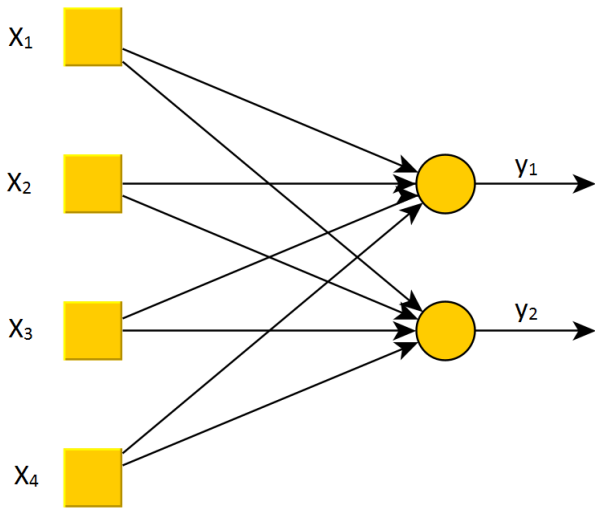


Figure 1.4: Réseaux de neurones monocouches

1.3.4.3 Réseaux de neurones multicouches

Les neurones sont arrangés par couche. Il n'y a pas de connexion entre neurones d'une même couche, et les connexions ne se font qu'avec les neurones de couches avales. Habituellement, chaque neurone d'une couche est connecté à tous les neurones de la couche suivante et celle-ci seulement. Ceci nous permet d'introduire la notion de sens de parcours de l'information (de l'activation) au sein d'un réseau et donc définir les concepts de neurone d'entrée, neurone de sortie.

Par extension, on appelle couche d'entrée l'ensemble des neurones d'entrée, couche de sortie l'ensemble des neurones de sortie. Les couches intermédiaires n'ayant aucun contact avec l'extérieur sont appelées couches cachées.

La figure 1.5 représente un réseau de neurones non bouclé qui a une structure particulière, très fréquemment utilisée : il comprend des entrées, deux couches 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 multicouches.

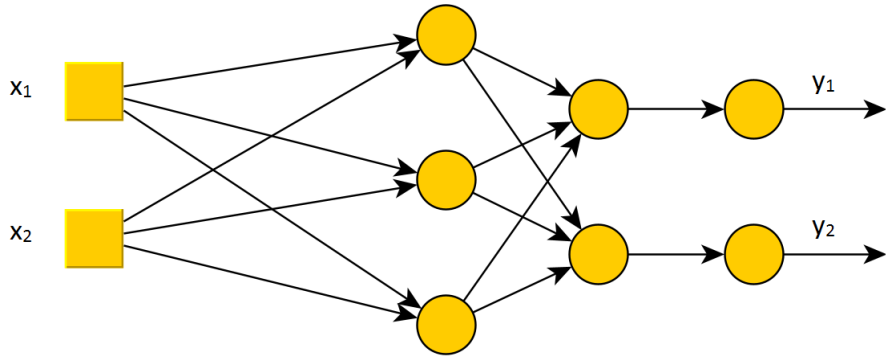


Figure 1.5: Réseaux de neurones multicouches

On note aussi que Les réseaux multicouches sont beaucoup plus puissants que les réseaux simples à une seule couche. En utilisant deux couches (une couche cachée et une couche de sortie), à condition d'employer une fonction d'activation sigmoïde sur la couche cachée, on peut entraîner un réseau à produire une approximation de la plupart des fonctions, avec une précision arbitraire (cela peut cependant requérir un grand nombre de neurones sur la couche cachée). Sauf dans des rares cas, les réseaux de neurones artificiels exploitent deux ou trois couches.

1.3.4.4 Réseaux de neurones à connexions locales

Il s'agit d'une structure multicouche, mais qui à l'image de la rétine conserve une certaine topologie. Chaque neurone entretient des relations avec un nombre réduit et localisé de neurones de la couche avale. Les connexions sont donc moins nombreuses que dans le cas d'un réseau multicouche classique (figure 1.6).

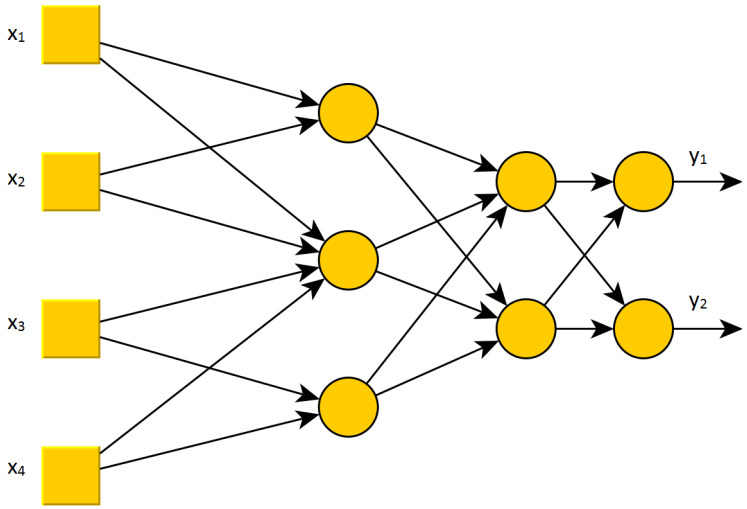


Figure 1.6: Réseaux de neurones à connexions locales

Les réseaux de neurones non bouclés sont des objets statiques : si les entrées sont indépendantes du temps, les sorties le sont également. Ils sont utilisés principalement pour effectuer des tâches d'approximation de fonction non linéaire, de classification ou de modélisation de processus statiques non linéaire.

1.3.4.5 Les réseaux de neurones récurrents

Contrairement aux réseaux de neurones non bouclés dont le graphe de connexions est acyclique, les réseaux de neurones bouclés peuvent avoir une topologie de connexions quelconque, comprenant notamment des boucles qui ramènent aux entrées la valeur d'une ou plusieurs sorties. Pour qu'un tel système soit causal, il faut évidemment qu'à toute boucle soit associé un retard : un réseau de neurones bouclé est donc 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.

Il s'agit donc de réseaux de neurones avec retour en arrière (feedback network or recurrent network), (Figure 1.7).

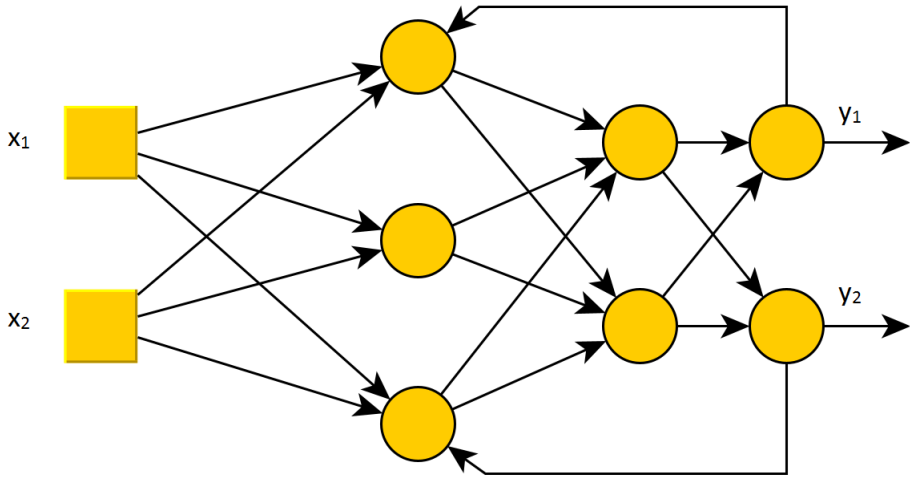


Figure 1.7: réseaux de neurones récurrentsts

Les réseaux de neurones bouclés sont utilisés pour effectuer des tâches de modélisation de systèmes dynamiques, de commande de processus, ou de filtrage (3)

1.4 Algorithmes d'apprentissage

L'une des caractéristiques les plus pertinentes des réseaux de neurones artificiels est leur capacité à apprendre à partir de la présentation d'échantillons (motifs), qui exprime le comportement du système. Par conséquent, une fois que le réseau a appris la relation entre les entrées et les sorties, il peut généraliser les solutions, ce qui signifie que le réseau peut produire une sortie qui est proche de la sortie attendue (ou souhaitée) de toute valeur d'entrée donnée.

Par conséquent, le processus d'apprentissage d'un réseau de neurones consiste à appliquer les étapes ordonnées requises pour régler les poids et seuils synaptiques de ses neurones, afin de généraliser les solutions produites par ses sorties.

L'ensemble des étapes ordonnées utilisées pour entraîner le réseau est appelé algorithme d'apprentissage. Au cours de son exécution, le réseau pourra ainsi extraire des caractéristiques discriminantes du système en cours de cartographie à partir d'échantillons acquis du système.

Au cours du processus d'apprentissage des réseaux de neurones artificiels, chaque présentation complète de tous les échantillons appartenant à l'ensemble d'apprentissage, afin d'ajuster les poids et les seuils synaptiques, sera appelée époque d'apprentissage (5)

On peut distinguer deux types d'apprentissage :

1.4.1 Apprentissage supervisé

La stratégie d'apprentissage supervisé consiste à disposer des sorties souhaitées pour un ensemble donné de signaux d'entrée; en d'autres termes, chaque échantillon d'apprentissage est composé des signaux d'entrée et de leurs sorties correspondantes. Désormais, il nécessite une table avec des données d'entrée / sortie, également appelée table attribut / valeur, qui représente le processus et son comportement. C'est à partir de ces informations que les structures neurales formuleront une hypothèse sur le système en cours d'apprentissage.

Dans ce cas, l'application de l'apprentissage supervisé ne dépend que de la disponibilité de cette table attribut / valeur, et se comporte comme si un coach apprenait au réseau quelle est la réponse correcte pour chaque échantillon présenté pour son entrée.

Les poids et seuils synaptiques du réseau sont ajustés en permanence grâce à l'application d'actions comparatives, exécutées par l'algorithme d'apprentissage lui-même, qui supervisent l'écart entre les sorties produites par rapport aux sorties souhaitées, en utilisant cette différence sur la procédure d'ajustement. Le réseau est considéré comme formé lorsque cet écart se situe dans une plage de valeurs acceptables, compte tenu des objectifs de généralisation des solutions.

En fait, l'apprentissage supervisé est un cas typique d'inférence inductive pure, où les variables libres du réseau sont ajustées en connaissant a priori les sorties souhaitées pour le système étudié.

Donald Hebb a proposé la première stratégie d'apprentissage supervisé en 1949, inspirée des observations neurophysiologiques (Hebb 1949) (5).

1.4.2 Apprentissage non supervisé

Contrairement à l'apprentissage supervisé, l'application d'un algorithme basé sur un apprentissage non supervisé ne nécessite aucune connaissance des sorties respectives souhaitées.

Ainsi, le réseau doit s'organiser lorsqu'il existe des particularités existantes entre les éléments qui composent l'ensemble de l'échantillon, en identifiant les sous-ensembles (ou clusters) présentant des similitudes. L'algorithme d'apprentissage ajuste les poids et les seuils synaptiques du réseau afin de refléter ces clusters au sein du réseau lui-même.

Alternativement, le concepteur de réseau peut spécifier (a priori) la quantité maximale de ces clusters possibles, en utilisant ses connaissances sur le problème (5).

1.5 Conclusion

Dans ce chapitre, nous avons fourni des définitions de base des réseaux de neurones. Nous avons distingué les réseaux de neurones statiques non linéaires, qui exécutent des fonctions non linéaires, et les réseaux en anneau dynamiques, qui exécutent des équations de différence non linéaires.

Nous avons également mis l'accent sur l'utilisation des réseaux de neurones comme outils d'apprentissage de modélisation. Celles-ci permettent de superposer des fonctions non linéaires très générales à des groupes de points. Comme toute méthode reposant sur des techniques statistiques, l'utilisation de réseaux de neurones nécessite des données représentatives suffisantes.

Chapitre 2

Les algorithmes génétiques

2.1 Introduction

Les algorithmes génétiques ont été créés par J.H. Holland pour mimer les processus observés dans l'évolution des espèces. De nos jours ces processus ne sont pas encore complètement connus mais les biologistes s'accordent sur certains points:

- L'évolution des espèces est un processus qui opère sur des structures appelées chromosomes. Ces structures représentent sous une forme codée les caractéristiques d'un individu (c'est-à-dire un être vivant).

- La sélection naturelle est un processus par lequel les individus qui se sont le mieux adaptés à l'environnement qui les entoure se voient donner une plus grande chance de survie. Les chromosomes de ces individus auront donc une probabilité plus grande de figurer dans les descendances futures.

- Le matériel chromosomique des descendants n'est pas identique à ceux des deux parents. Les chromosomes créés dans les descendances proviennent de la recombinaison ou de la modification des chromosomes hérités des deux parents. Les opérateurs biologiques assurant ce mélange chromosomique se nomment respectivement Croisement (Crossover en anglais) et Mutation.

- Le seul arbitre de la sélection est l'adaptation à l'environnement. L'évolution des espèces ne possède pas de mémoire: la seule connaissance sur la manière de produire les individus les mieux adaptés est contenue dans l'ensemble des gènes formant le chromosome.

Ces principes ont intrigué des chercheurs dans les années 60 et les premiers ouvrages sur des algorithmes génétiques sont dus à [Bagley67] ou [Rosenberg67]. Mais ce n'est qu'en 1975 que J.H. Holland [Holland75] jette les bases théoriques d'un algorithme d'optimisation s'appuyant sur des techniques dérivées de la génétique et des mécanismes de la sélection naturelle (6)

2.2 Le principe des algorithmes génétiques

Les algorithmes génétiques présentent des qualités intéressantes pour la résolution de problèmes d'optimisation non linéaire. Leurs fondements théoriques tentent de simuler le processus d'évolution des espèces dans leur milieu naturel. La génétique représente un individu par un code, c'est-à-dire un ensemble de données (appelés chromosomes), identifiant complètement l'individu. La reproduction est, dans ce domaine, un mixage aléatoire de chromosomes de deux individus, donnant naissance à des individus enfants ayant une empreinte génétique nouvelle, héritée des parents. La mutation génétique est caractérisée dans le code génétique de l'enfant par l'apparition d'un chromosome nouveau, inexistant chez les individus parents. Ce phénomène génétique d'apparition de

" mutants" est rare, dans le sens d'une meilleure adaptation au milieu naturel. La disparition de certaines espèces est expliquée par " les lois de survie " selon lesquels seuls les individus les mieux adaptés auront une longévité suffisante pour générer une descendance. Les individus peu adaptés auront une tendance à disparaître. C'est une sélection naturelle qui conduit de génération en génération à une population composée d'individus de plus en plus adaptés.

Un algorithme génétique est construit de manière tout à fait analogue. Dans l'ensemble des solutions d'un problème d'optimisation, une population de taille N est constituée de N solutions (les individus de la population) convenablement marquées par un codage identifie complètement. Une procédure d'évaluation est nécessaire à la détermination de la force de chaque individu de la population. Viennent ensuite une phase de sélection et une phase de recombinaison (opérateurs de croisement et de mutation) qui génèrent une nouvelle population d'individus, qui ont de bonnes chances d'être plus fortes que ceux de la génération précédente. De génération en génération, la force des individus de la population augmente et après un certain nombre d'itérations, la population est entièrement constituée d'individus tous forts, soit de solutions quasi-optimales du problème posé.

Le fonctionnement d'un AG est alors passe par les phases suivantes :

1. Initialisation : une population initiale de taille N chromosomes est tirée aléatoirement.

2. Évaluation : chaque chromosome est décodé puis évalué.

3. Reproduction: création d'une nouvelle population de N chromosomes par l'utilisation d'une méthode de sélection appropriée.

4. Opérateurs génétiques: croisement et mutation de certains chromosomes au sein de la nouvelle population.

5. Retour à la phase 2 tant que la condition d'arrêt du problème n'est pas satisfaite.

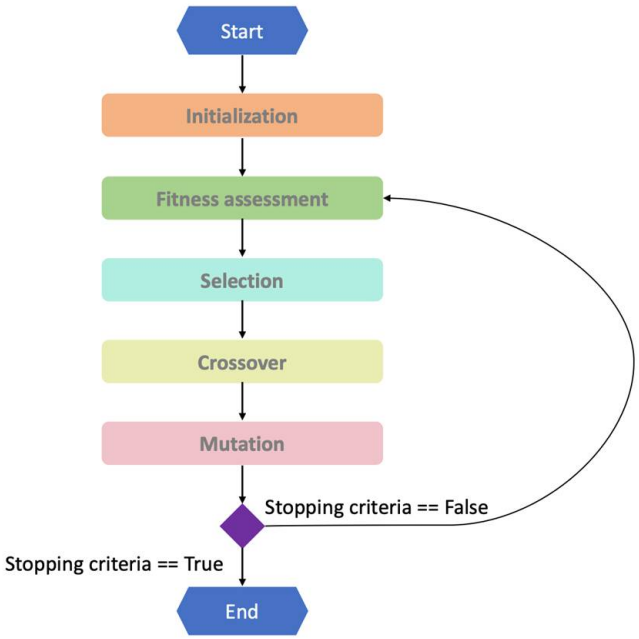


Figure 2.1: Principe de fonctionnement d'un algorithme génétique

Un algorithme génétique a pour but de rechercher un optimum d'une fonction définie sur un espace borné, et repose sur les points suivants :

- un principe de codage de l'élément de population : la qualité du codage des données conditionne le succès de la recherche de l'optimum. On retrouve deux formes de codage : le codage binaire (parfois GRAY) et le codage réel.

- un mécanisme de génération de la population initiale non homogène. Le choix de la population conditionne la rapidité de la convergence vers l'optimum.

- une fonction à optimiser qui retourne une fonction d'adaptation (fitness) de l'individu.

- un mécanisme de sélection des individus permettant d'identifier statistiquement les meilleurs individus d'une population et d'éliminer les moins bons.

- des opérateurs permettant de diversifier la population au cours des générations et d'explorer l'espace des autres solutions possibles. L'opérateur de croisement (crossing-over) recompose les gènes d'individus existant dans la population.

L'opérateur de mutation a pour but de garantir l'exploitation de l'espace de solutions.

- des paramètres de dimensionnement : la taille de la population, critères d'arrêt, probabilité d'application des opérateurs génétiques (7) .

2.3 Termes et définitions

2.3.1 Population initiale

Ce mécanisme doit être capable de produire une population d'individus uniformément répartis sur l'espace de recherche, qui servira de base pour les générations futures. Suivant la position de la population initiale dans l'espace d'état, on favorise la recherche de l'optimum. Si l'on n'a aucune connaissance de la position de l'optimum, on génère aléatoirement des individus en faisant des tirages uniformes dans l'espace de recherche. Si par contre, on dispose d'informations a priori sur le problème, on indique un sous-espace où l'on est sûr de trouver l'optimum, il faut générer les individus dans ce sous-espace afin d'accélérer la convergence. A partir de cette population initiale, il faut être capable de maintenir la diversité de la population au cours des générations afin d'entretenir le processus d'exploration de l'espace d'état : c'est le rôle des opérateurs génétiques. Mais auparavant, la performance de chacun des individus doit d'abord être évaluée. Le nombre d'individus d'une population ou la taille de la population constitue un paramètre important pour l'AG qu'il faudra déterminer. La représentation de la population P est: P = (C1, C2,. . . , Ci,. . . , C taille-pop)

Où C1 représente le ième chromosome dans la population et taille-pop représente le nombre de chromosomes dans la population (8) .

Généralement, la taille de la population reste constante tout au long de l'algorithme génétique.

2.3.2 Individu

Les individus correspondent aux solutions possible de la fonction à optimiser f(x). Ces solutions doivent être codées pour que le traitement puisse être effectué par l'algorithme génétique. Cette représentation codée d'une solution est appelée chromosome, et est composée de gènes. Chaque gène peut représenter une variable, un élément de la solution, ou encore une partie plus abstraite. La manière la plus utilisée de codage par algorithme génétique est le codage en vecteurs. Chaque solution est représentée par un vecteur. Ce vecteur peut être binaire ou encore de n'importe quel type discret dénombrable (entier, caractères, etc.). On pourrait également utiliser nombres réels, mais dans ce cas, il faut également revoir les opérations qui modifient le contenu des chromosomes (la fonction qui crée aléatoirement les chromosomes et les opérateurs génétiques). La simplicité veut que les chromosomes soient uniformes, c'est-à-dire que tous les gènes sont du même type. Cependant, si on tient, compte encore une fois des opérations qui modifient le contenu des chromosomes, on peut assez aisément construire des vecteurs d'éléments de type différents (7) .

2.3.3 Le codage

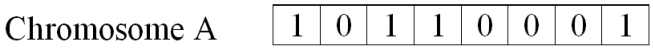
Le premier problème rencontré lors de l'utilisation des algorithmes génétiques est le codage des individus (i.e. élément de l'espace de recherche). Cette représentation est l'un des points fondamentaux des algorithmes génétiques. Le chromosome peut selon différentes façons, contenir l'information concernant la solution qu'il représente.

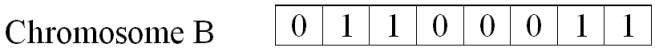
2.3.3.1 Le codage binaire

Goldberg et Holland ont démontré qu'il est idéal de représenter le chromosome en une chaîne binaire. C'est pourquoi les AG utilisent généralement cette représentation Les individus sont représentés sous forme de chaînes de bits contenant toute l'information nécessaire à la description d'un point dans l'espace. Ce type de codage a pour intérêt de permettre la création d'opérateurs de croisement et de mutation simples.

Dans le codage binaire le gène est codé par un caractère binaire, 0 ou 1. C'est le plus courant et celui qui a été employé lors de la première application des algorithmes génétiques. Un des avantages du codage binaire est que l'on peut ainsi facilement coder toutes sortes d'objets : des réels, des entiers, des valeurs booléennes, des chaînes de caractères... Cela nécessite simplement l'usage de fonctions de codage et décodage pour passer d'une représentation à l'autre (9) .

Exemples :





2.3.3.2 Le codage réel

Davis, Janikow et Michalewicz ont effectué une comparaison entre la représentation binaire et la représentation réelle. Ces auteurs ont trouvé que la représentation réelle donne de meilleurs résultats d'après leur problème à résoudre. Dans ce codage le génome est un vecteur réel et l'espace de recherche est un sous ensemble de IR. Cette représentation est aujourd'hui très

utilisée dans les problèmes d'optimisation car dans de nombreuses applications du monde réel, ces problèmes sont naturellement formulés sous forme paramétrique i.e. Les premiers travaux qui ont utilisé ce type de représentation ont été ceux de Rechenberg et de Schwefel quand ils ont introduit les stratégies d'évolution. Le codage réel peut-être utile notamment dans le cas où l'on recherche le maximum d'une fonction réelle (9) .

Exemple :

Chromosome

5	9	8	0	1	3	1	0

2.3.4 L'opérateur de mutation

La mutation est exécutée seulement sur une seule chaîne. Elle représente la modification aléatoire et occasionnelle de faible probabilité de la valeur d'un caractère de la chaîne, pour un codage binaire cela revient à changer un 1 en 0 et vice versa (figure 2.2). Cet opérateur introduit de la diversité dans le processus de recherche des solutions et peut aider l'AG à ne pas stagner dans un optimum local (10) .

Figure 2.2: Représentation d'une mutation de bits dans une chaîne

2.3.5 L'opération de croisement ou crossover

Cette technique nécessite deux parents afin d'effectuer l'échange des gènes entre eux. Cet échange permet de former deux descendants possédant des caractéristiques issues des deux parents. Chaque individu se voit attribuer une même probabilité Pcross de participer à un croisement. L'opérateur de croisement favorise l'exploration de l'espace de recherche. Il assure le brassage du matériel génétique et l'accumulation des mutations favorables. En d'autres termes, cet opérateur permet de créer de nouvelles combinaisons ayant des caractéristiques communes avec leurs parents. Souvent les meilleures caractéristiques sont transmises aux descendants. Cette transmission est appelée héritage. Seuls les individus les mieux adaptés vivent suffisamment longtemps pour se reproduire. Ceci va conduire à des progénitures encore mieux adaptées.

2.3.5.1 Croisement à un point

Le croisement à un point est le croisement le plus simple. Pour effectuer ce type de croisement, on sélectionne aléatoirement un point de coupure K qui soit compris entre1 et L-1 (L est la longueur du chromosome) puis on subdivise le génotype de chacun des parents en deux parties de part et de l'autre part de ce point. On échange ensuite les deux sous chaînes terminales de chacun des chromosomes, ce qui produit deux enfants.

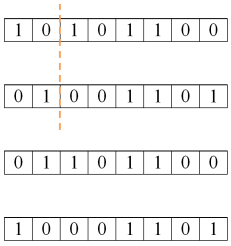


Figure 2.3: Représentation d'un croisement a un point

2.3.5.2 Croisement multipoints

Ce type de croisement peut être vu comme une généralisation du croisement à un point, en découpant le chromosome non seulement en 2 sous chaînes mais en K sous chaînes.

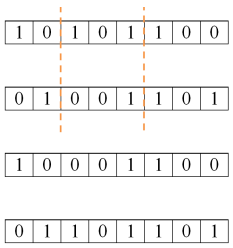


Figure 2.4: Représentation d'un croisement a deux points

2.3.5.3 Croisement uniforme

Le croisement uniforme peut être vu comme un croisement multipoint dont le nombre de coupures est indéterminé a priori. Pratiquement on utilise un masque de croisement , engendré aléatoirement pour chaque couple d'individus, qui est un mot binaire de même longueur que les chromosomes. S'il est égal à 1, l'enfant 1 reçoit l'allèle correspondant du parent 1 et l'enfant 2 reçoit celui de parent 2. Sinon l'échange se fait dans l'autre sens (11) .

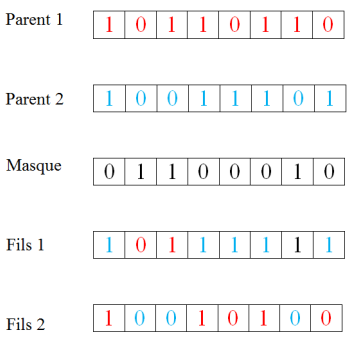


Figure 2.5: Représentation d'un croisement uniforme

2.3.6 L'opérateur de sélection

Cet opérateur est chargé de définir quels seront les individus de P qui vont être dupliqués dans la nouvelle population P' et vont servir de parents (application de l'opérateur de croisement).Soit n le nombre d'individus de P, on doit sélectionner P/2 (l'opérateur de croisement nous permet de repasser à n individus).

Cet opérateur est peut-être le plus important puisqu'il permet aux individus d'une population de survivre, de se reproduire ou de mourir. En règle générale, la probabilité de survie d'un individu sera directement reliée à son efficacité relative au sein de la population (8) .

2.3.6.1 La sélection par tournois

Consiste à tirer aléatoirement (k) individus de la population, sans tenir compte de la valeur de leur fonction d'adaptation, et de choisir le meilleur individu parmi les k individus. Le nombre d'individus sélectionnés a une influence sur la pression de sélection, lorsque k = 2, la sélection est dite par tournoi binaire (10) .

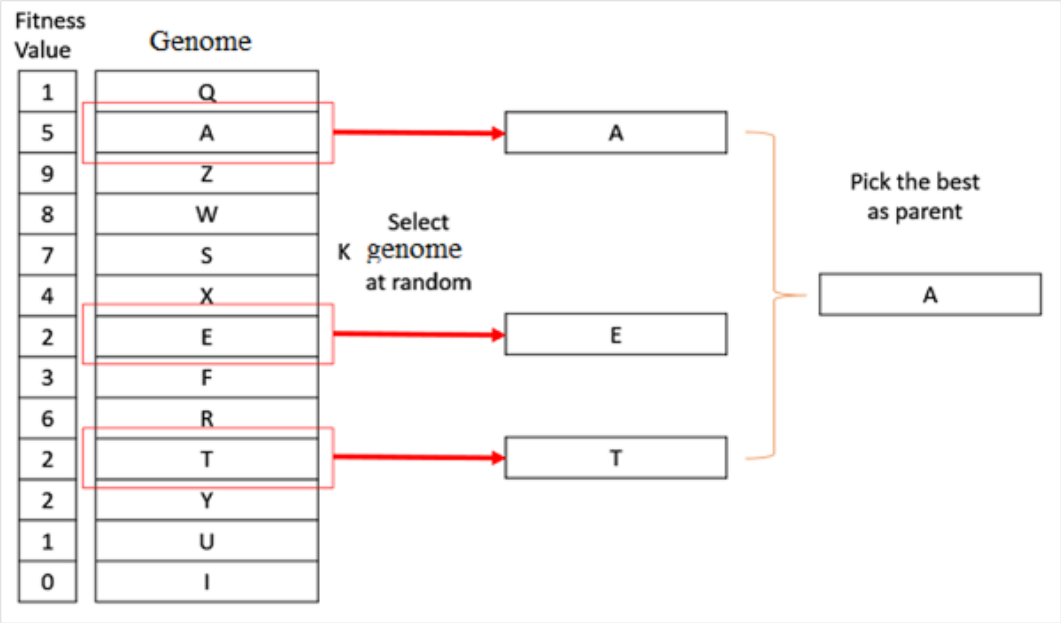


Figure 2.6: Représentation d'une sélection par tournoi

2.3.6.2 La sélection par roulette

C'est la méthode la plus connue des sélections stochastiques, proposé par J. Holland. Elle consiste à sélectionner les individus proportionnellement à leur performance, donc plus les individus sont adaptés au problème, plus ils ont de chances d'être sélectionnés. La probabilité de sélection est calculée à partir de la valeur de "fitness" du chromosome dans la population (12) .

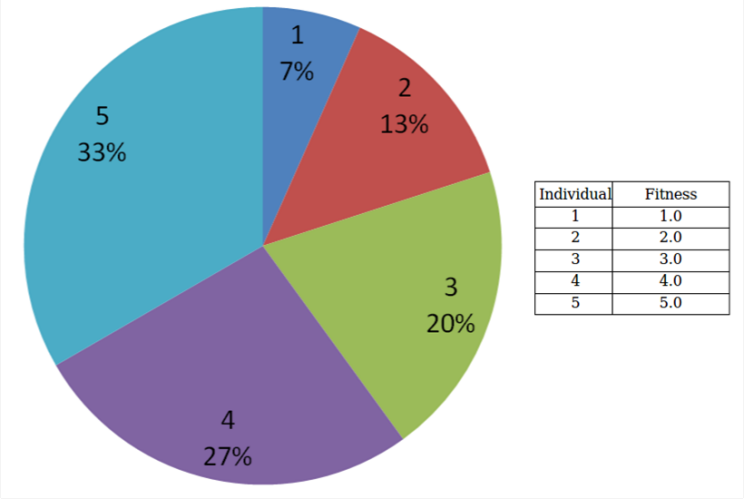


Figure 2.7: Représentation d'une sélection par roulette

2.3.6.3 La méthode élitiste

Cette méthode consiste à sélectionner les M meilleurs individus de la population P dont on a besoin pour la nouvelle génération P0; après les avoir trié de manière décroissante selon leurs fonctions de performance. Il est inutile de préciser que cette méthode est encore pire que celle de la roulette dans le sens où elle amène à une convergence prématurée encore plus rapidement et surtout de manière encore plus sûre que la méthode de sélection de roulette ; en effet, la pression de la sélection est trop forte, la variance nulle et la diversité inexistante, du moins le peu de diversité qu'il pourrait y avoir ne résultera pas de la sélection mais plutôt du croisement et des mutations (12) .

2.3.7 La fonction fitness (la fonction d'évaluation)

Chaque chromosome apporte une solution potentielle au problème à résoudre. Néanmoins, ces solutions n'ont pas toutes le même degré de pertinence. C'est à la fonction de performance, dite aussi fonction d'adaptation ou fitness, de mesurer cette efficacité pour permettre à l'algorithme génétique de faire évoluer la population dans un sens bénéfique pour la recherche de la meilleure solution.

Autrement dit, la fonction d'adaptation doit pouvoir attribuer à chaque individu un indicateur positif représentant sa pertinence pour le problème qu'on cherche à résoudre. La fonction d'évaluation ou d'adaptation (fitness) associe donc un coût à chaque chromosome. Et c'est grâce à la fitness que l'algorithme évalue l'adaptation des individus à leur environnement et calcule leurs chances de survivre et de se reproduire en favorisant la sélection d'individus dans la direction de l'optimum qui est, a priori, inconnue.

Elle constitue donc le critère à base duquel l'individu serait ou pas sélectionné pour être reproduit dans la génération suivante. La qualité de cette fonction conditionne, pour une grande part, l'efficacité d'un algorithme génétique. Il est, par conséquent, important de tenir compte de sa complexité. En effet, dans le cas où la fonction d'adaptation apparaît excessivement complexe, consommant une importance puissance de calcul, il est souhaitable de lui rechercher une approximation plus simple (8) .

2.4 Le critère d'arrêt de l'algorithme:

Les conditions d'arrêt de l'algorithme sont variables, temps limité, optimum atteint...

1. L'arrêt de la convergence au bout d'un nombre fixé de générations.

2. L'arrêt lorsque l'évaluation d'un individu atteint une note fixée.

3. L'arrêt lorsque la note du meilleur individu ne progresse plus au de la d'un certain nombre d'évaluations.

4. L'arrêt lorsqu'un temps ou un nombre fixé d'évaluations a été atteint.

2.5 Conclusion

Nous avons présenté dans ce chapitre les principes de base d'un outil de résolution qui a fait ses preuves en intelligence artificielle (IA), à savoir l'algorithme génétique. Nous retenons principalement qu'un algorithme génétique est une reproduction artificielle de mécanismes naturels liés à la génétique. Son principe, basé sur une description fidèle d'une évaluation naturelle, est un processus cyclique manipulant une population et assurant une recherche efficace dans un espace de solutions candidates d'un problème donné, par application des trois opérateurs qui sont la sélection, le croisement et la mutation, dans le but d'optimiser la fonction d'adaptation associée, définie sur un espace éventuellement complexe.

Chapitre 3

Conception et résultat

3.1 Introduction

Le chapitre précédent nous a permis d'avoir une vue générale sur les concepts des algorithmes génétiques.

Nous pouvons conclure que les algorithmes génétiques sont des algorithmes simples de conception et peuvent résoudre des problèmes assez complexes.

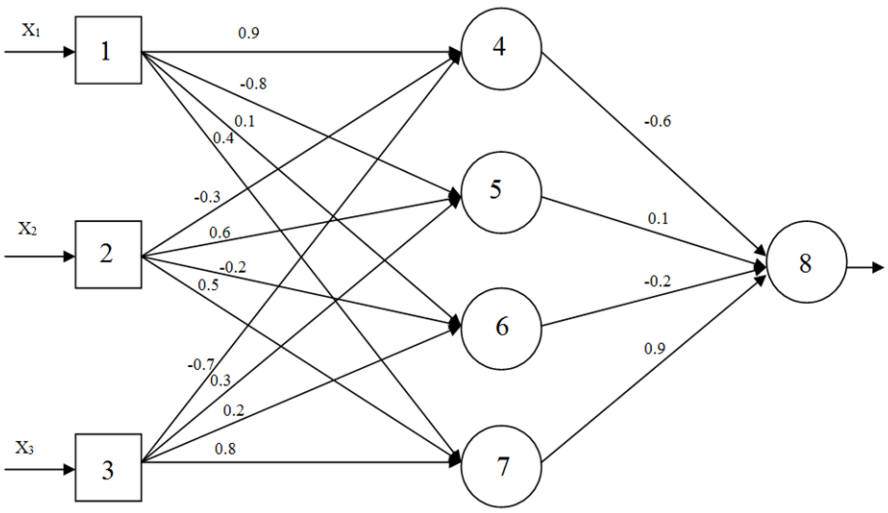
La résolution de ces problèmes est obtenue grâce aux opérateurs de reproduction. Ces AG sont des procédures assez robustes pour résoudre un problème d'optimisation pour la sélection des primitives. Néanmoins elles présentent certaines limites et des difficultés. Ces difficultés reposent sur le choix des bons paramètres tels que : la taille de la population, le nombre de génération, les probabilités de croisement et de mutation et les méthodes des opérateurs de reproduction. Ces paramètres dépendent du problème à résoudre et d'une codification appropriée au problème à solutionner.

Dans notre projet, nous utilisons les algorithmes génétiques pour l'entrainement des réseaux de neurones.

Dans ce chapitre, nous expliquons comment nous procédons.

3.2 Utilisation des algorithmes génétiques pour une optimisation des poids

La partie suivante présentent le concept de base d'une technique d'optimisation de poids génétique (le Montana et David, 1989; blanchement et Hanson, 1989; Ichikawa et sawa, 1992). Pour une utilisation des algorithmes génétiques, il faut d'abord représenter le domaine de problème comme un chromosome. Par exemple, nous voulons optimiser les poids d'un perceptron multicouche présenté dans la figure 3.1 .



0.9	-0.3	-0.7	-0.8	0.6	0.3	0.1	-0.2	0.2	0.4	0.5	0.8	-0.6	0.1	-0.2	0.9

Figure 3.1: Présentation d'un système neuro-génétique.

Dans la première étape on va générer Des poids initiaux dans le réseau choisi aléatoirement dans le petit intervalle [-1,1]. Dans ce perceptron, il y a 16 liaisons pondérées entre les neurones. Puisqu'un individu est un ensemble de gènes, l'ensemble des poids peut être représenté par un individu à 16 gènes, où chaque gène correspond à une liaison simple pondérée dans le réseau. Ce chromosome présente un individu d'une population c.ad une solution proposé à partir d'un ensemble des solutions.

Dans La deuxième étape on doit définir une fonction d'évaluation (fitness) pour évaluer la performance des individus. Cette fonction doit estimer la performance d'un réseau neuronal donné. Nous pouvons appliquer ici une fonction assez simple définie par la réciproque de l'erreur telle que l'algorithme génétique essaye de trouver un ensemble des poids (individu) qui réduisent au minimum la somme d'erreurs quadratique. On peut utiliser aussi comme une fonction le taux de classification non correcte et l'algorithme génétique essaye de trouvé l'individu qui réduise aux minimum ce taux.

La troisième étape on doit appliquer les deux opérateurs des algorithmes génétiques croissement et mutation. Un opérateur de croisement prend deux individus parentaux et crée un enfant simple avec le matériel génétique des deux parents. Chaque gène de l'individu d'un enfant est une information génétique transmise par les parents. Figure 3.2 montre une application de l'opérateur de croisement.

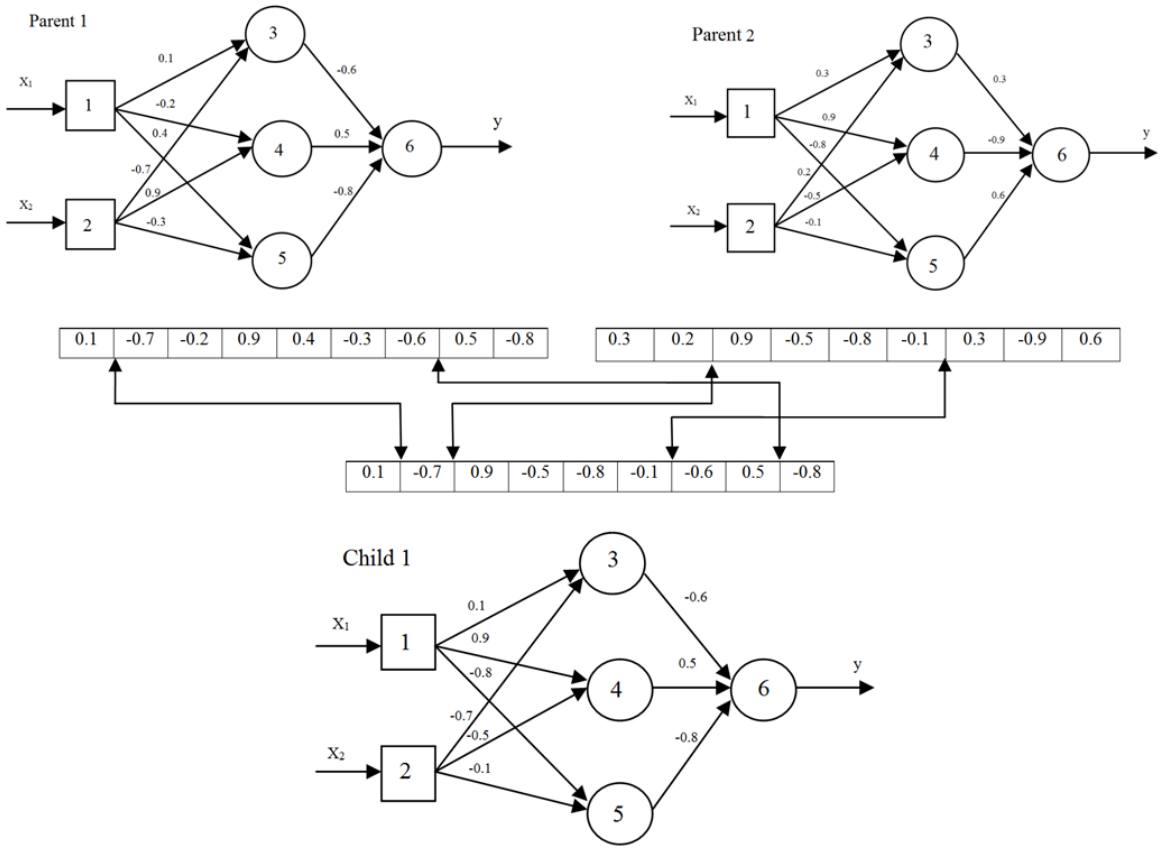


Figure 3.2: Opérateur de croisement dans un système neuro-génétique

Un opérateur de mutation choisit aléatoirement un gène dans un individu et ajoute une petite valeur aléatoire à chaque poids dans ce gène. Figure 3.3 montre un exemple de mutation.

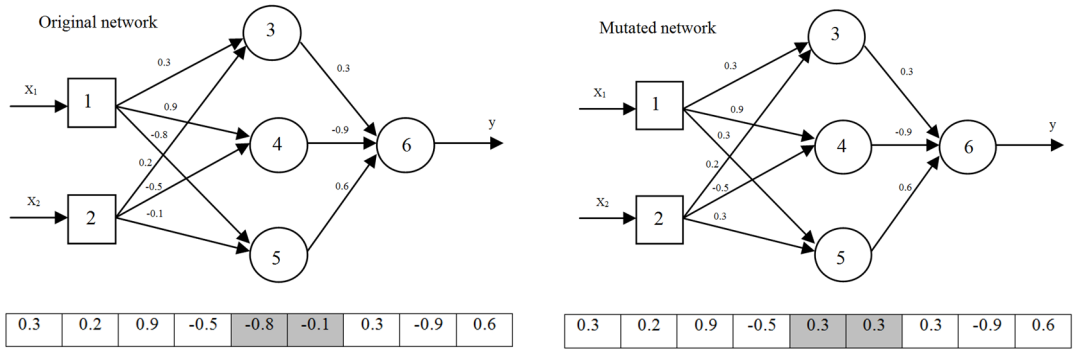


Figure 3.3: opérateur de mutation dans le système neuro-génétique.

Maintenant nous sommes prêts à appliquer l'algorithme génétique. Bien sûr, nous devons toujours définir la taille de population, c'est-à-dire le nombre de réseaux avec des poids différents, la probabilité de croisement et de mutation et le nombre de générations.

Jusqu'ici nous avons assumé que la structure du réseau est fixée et l'apprentissage génétique est employé seulement pour optimiser les poids dans le réseau donné. Cependant l'architecture du réseau c'est-à-dire le nombre de neurones et les connections entre les neurones détermine souvent l'échec de l'application.

D'habitude l'architecture de réseau est décidée par l'essai et l'erreur; il y a un grand besoin d'une méthode pour concevoir automatiquement l'architecture pour une application particulière. Des algorithmes génétiques peuvent bien nous aider dans la sélection de l'architecture de réseau.

3.3 Conception détaille

3.3.1 Les bases de données

Une base de données est un ensemble structuré et organisé permettant le stockage de grandes quantités d'informations afin d'en faciliter l'exploitation. Dans notre cas, on traite des tables des données avec des attributs de type numérique pour simplifier les calculs, et faire des expériences pour s'assurer de l'exactitude du travail.

```
{4.375013190394657   ,-3.419603298472216
{-1.4013042863851597 ,4.284759964524524
{4.653653035344373   ,4.874780337369209
{0.8541570488311878  ,1.6084581070786736
{4.846509614332518   ,-1.4410091626106993
{0.39937836558904394 ,-4.691477707148027
{4.87371064745496    ,-3.6298671375394598
{0.8811169793854248  ,-2.658222799247243
{1.7953101844357624  ,-1.8143777884150127
{-2.4512521990395317 ,-4.510060087709577
{1.3963733419104516  ,-0.3939248954346173
{2.8467316364087836  ,4.961330424673834
{1.0904663010310855  ,-3.9754495373523566
{-0.3354596636890559 ,-4.445046811552087
{-3.358098796752885  ,4.616164273441841
{-3.4664214333535726 ,1.008017797060523
{0.13601826053159094 ,-1.7136203003994108
{0.5737403964854781  ,-0.44141855135356156
{3.9257173323576     ,-2.7424550206014375
{-2.3346252483478667 ,3.656708590466014
{-3.7888654618704654 ,-1.073087108551042
        ⋮                    ⋮
```

Figure 3.4: une base de données.

3.3.2 Module d'évolutif avec les algorithmes génétiques

3.3.2.1 Génération de population initiale

La taille de la population initiale est importante et est déterminée au début, elle reste constante et ne change pas tout au long de l'étude.

```
Initi_population ()
début
    entier i, popsize;
    liste génome;
    pour ( i=0 à popsize ) faire
        add2pop( génome(i));
    Fin pour
fin
```

tel que:

popsize: est la taille de population.

génome :la liste des individus qui composent la population.

add2pop :une fonction qui ajoute les individu à la population.

3.3.2.2 Fonction de fitness

La fonction de fitness se change d'un algorithme génétique à un autre selon le problème traité. Dans notre cas la fonction de fitness sera assez simple définie par l'erreur telle que l'algorithme génétique essaye de trouver un ensemble des poids (individu) qui réduisent au minimum la somme d'erreurs de la forme :

$$\text{Erroro} = \sum_{i=0}^{n} (/\mathrm{d_i} - \mathrm{y_i}/)$$

Avec d: est la sortie désirée.

y: est la sortie obtenu.

Le but est d'obtenir y le plus proche possible de d, pour tout i = 1, ..., n. Pour savoir si l'objectif est atteint, nous mesurons la différence entre ces valeurs.

3.3.2.3 La sélection

Nous avons utilisé la sélection par tournois.

```
Sélection ()
début
  entier i, j;
  liste selected_liste;  //  c'est une liste qui a k individus choisis au hasard dans la population
  j=1;
  Best = selected_liste (j);
  pour ( i = 2  à  selected_liste_size ) faire
    si ( selected_liste (i)  <  Best );
        Best =  selected_liste (i)  ;
        j= i;
    Finsi
  Fin pour
  return j;
fin
```

3.3.2.4 Coisement

```
Crossover ()
début
  Par le processus de sélection, nous sélectionnons des individus à croiser pour créer de
  nouveaux individus.
  Switch liste do
    case 0
    single_point_crossover (génome1, génome2);
    Break;
    case 1
    double_point_crossover (génome1, génome2);
    Break;
    case 2
    uniform_crossover (génome1, génome2);
    Break;
    case 3
    miror_ double_point_crossover (génome1, génome2);
    Break;
  end
Fin
```

```
single_point_crossover (genome1, genome2)
Début
    entier genomesize, i;
    choisir un point de découpe de manier aléatoire X de [1, genomesize - 1 ] // X≠0 et
                                                                                // X≠ genomesize
    Pour ( i = 0 à X ) faire
        Fils1.add (genome1.get (i));
        Fils2.add (genome2.get (i));
    Fin Pour
    Pour (i = X à genomesize ) faire
        Fils1.add (genome2.get (i));
        Fils2.add (genome1.get (i));
    Fin Pour
    Deux fils sont créés pour la mutation
Fin
```

```
double_point_crossover (genome1, genome2)
Début
    entier genomesize, i;
    choisir deux points de découpe de manier aléatoire X1 et X2 tel que X1 < X2 et X1 ≠ 0 et
                                                                    X2 ≠ genomesize.
    Pour ( i = 0 à X1 ) faire
        Fils1.add (genome1.get (i));
        Fils2.add (genome2.get (i));
    Fin Pour
    Pour (i = X1 à X2 ) faire
        Fils1.add (genome2.get (i));
        Fils2.add (genome1.get (i));
    Fin Pour
    Pour (i = X2 à genomesize ) faire
        Fils1.add (genome1.get (i));
        Fils2.add (genome2.get (i));
    Fin Pour
    Deux fils sont créés pour la mutation
Fin
```

```
miror_ double_point_crossover (genome1, genome2)
Début
    entier genomesize, i;
    choisir deux points de découpe de manier aléatoire X1 et X2 tel que  X1 < X2 et  X1 ≠ 0 et
                                                                          X2 ≠ genomesize.
    Pour ( i = 0 à X1 ) faire
        Fils1.add (genome2.get (i));
        Fils2.add (genome1.get (i));
    Fin Pour
    Pour (i = X1 à X2 ) faire
        Fils1.add (genome1.get (i));
        Fils2.add (genome2.get (i));
    Fin Pour
    Pour (i = X2 à genomesize  ) faire
        Fils1.add (genome2.get (i));
        Fils2.add (genome1.get (i));
    Fin Pour
    Deux fils sont créés pour la mutation
Fin
```

```
uniform_crossover (genome1, genome2)
Début
    entier genomesize, i;
    Pour ( i = 0 à  genomesize ) faire
        définir R par hasard vrai ou faux
        Si (R)
            Fils1.add (genome2.get (i));
            Fils2.add (genome1.get (i));
        Sinon
            Fils1.add (genome1.get (i));
            Fils2.add (genome2.get (i));
        Fin si
    Fin Pour
    Deux fils sont créés pour la mutation
Fin
```

3.3.2.5 Mutation

```
Mutation ()
début
    Random rnd, rnd1;          // rnd from [0 , 1]  and rnd1 from [-5 , 5]
    sélectionner les bits a muté x aléatoirement dans chaque individu enfant
    pour chaque x :
    Switch liste do
        case 0
        génome (x) = génome (x) + rnd * génome (x);
        Break;
        case 1
        génome (x) = génome (x) - rnd * génome (x);
        Break;
        case 2
        génome (x) = rnd1;
        Break;
    end
Fin
```

Nous avons trois cas,

case 0 : changer sa valeur en l'ajoutant à une partie de sa valeur.

case 1 : changer sa valeur en la soustrayant avec une partie de sa valeur.

case 2 : changer complètement sa valeur.

3.3.2.6 Générer de nouvelles populations

Si la condition n'est pas remplie, nous revenons pour former une nouvelle population et appliquons les opérations d'algorithme génétique.

3.4 Structure du réseau de neurone

Le réseau neurone proposé est une architecture à trois couches, nous considérons cette fois ci un cas plus général avec un réseau à deux entrées [x1, x2] et à une sortie y. En considérant la figure suivante, explicitons le fonctionnement du réseau, couche par couche.

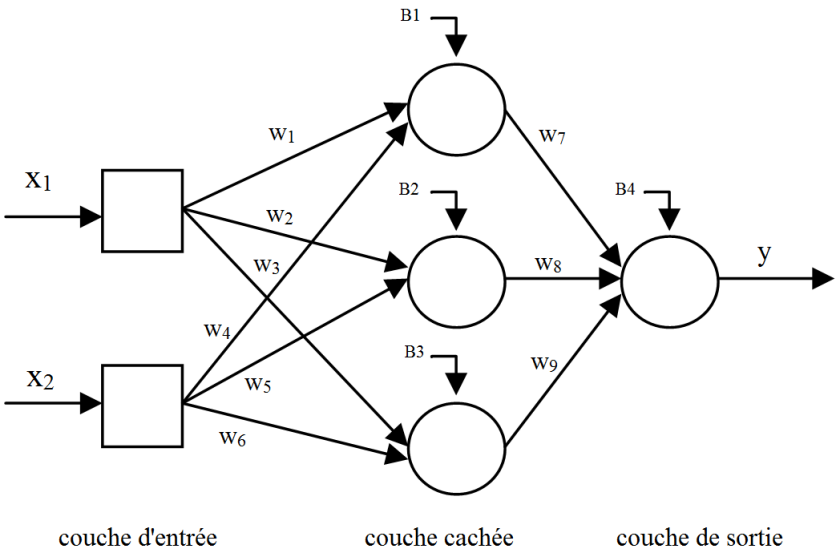


Figure 3.5: Exemple d'un réseau à une seule couche cachée.

Couche d'entrée : Aucune fonction n'est réalisée avec les neurones de cette couche. Chaque neurone transmet le signal d'entrée vers la deuxième couche. Avec 2 entrées noté x1, x2.

Couche cachée : une seule couche cachée est composé de 3 neurones. Les entrées sont :

$$c_j = b_j + \sum_{i=j=1}^{n} x_i * w_{ij}.$$

Où

w est le poids de connexion entre le neurone i de la couche d'entrée et le neurone j de la couche cachée.

b est le biais.

les sorties sont : f(cj)

tel que f est la fonction d'activation dans notre projet est :

$$f(x) = \frac{1}{2 + e^{-x}}$$

La couche de sortie : la sortie est composée d'un seul neurone. la sortie est

$$y = c_j = b_j + \sum_{i=j=1}^{n} x_i * w_{ij}.$$

w est le poids de connexion entre le neurone i de la couche cachée et le neurone j de la couche de sortie.

L'évaluation de RN :

C'est le calcul la fonction de fitness (la fonction d'erreur).

$$\text{Erroro} = \sum_{i=0}^{n} (/d_i - y_i/)$$

Avec d: est la sortie désirée.

y: est la sortie obtenu.

Dans notre exemple ici, nous avons une sortie y.

On calcule d par une fonction donnée selon l'exemple étudié, dans notre projet, nous avons utilisé la fonction suivante :

$$d(x_1, x_2) = \cos(x_1) * \sin(x_2).$$

Codage de l'individu:

Nous représentons le réseau de neurones en tant qu'individu de la manière suivante:

W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	B_1	B_2	B_3	B_4

3.5 Implémentation de l'algorithme

Après initialisation des poids et des biais :

1) Appliquer le vecteur d'entrée X = (x1,...,xn) aux neurones d'entrée.

2) Calculer les valeurs d'entrée et de sortie des neurones des couches cachées.

3) Calculer toutes les sorties finales.

4) Calculer la fonction fitness de réseau de neurones.

5) Transformer le réseau de neurones en un individu .

Maintenant si nous voulons former une population, nous préparons de nombreux réseaux de neurones et les convertissons en individus.

Maintenant pour former la population, nous utilisons les opérations d'algorithme génétique pour l'optimisation.

3.6 Les structures des données utilisées

Pour la manipulation des données, nous avons choisi une structure qui convient mieux au langage de programmation et la nature des données utilisées.

Le génome (individu) est représenté par différents types de champs:

– rnd de type Random pour générer des valeurs aléatoires.

– genotype un vecteur de double qui contient des valeurs généré aléatoirement par rnd.

– phenotype de type NeuralNetwork se compose de deux listes, la première étant les couches (layers)et la seconde les liens(links).

– error c'est l'erreur du génome.

– cases est une matrice qui contient des cas d'entrées.

– netconfig un vecteur qui représente la structure du réseau de neurones.

population des génomes

– genomes une liste de type genome.

– rnd de type Random pour générer des valeurs aléatoires.

– popsize c'est la taille de la population.

3.7 Résultats

Après avoir vérifié l'exactitude de notre travail en appliquant de nombreux exemples, nous pouvons maintenant en donner un et voir ses résultats dans un exemple.

Ici, nous avons utilisé un réseau de neurones avec une configuration [2, 30, 1], cela signifie que nous avons trois couches:

La couche d'entrée est constituée de deux neurones.

La couche cachée se compose de 30 neurones.

Et la couche de sortie provient d'un seul neurone.

Et pour faire la formation, nous avons formé des individus (500 individus) pour former la population initial par des poids généré aléatoirement, ensuite, nous avons appliqué les opérations de l'algorithme génétique pour 2000 générations on peut faire plus de générations, mais on peut obtenir de bons résultats avec ce nombre, cela suffit pour montrer à quel point le réseau de neurones est avancé .

À chaque génération, nous avons conservé le meilleur génome.

Et pour voir à quel point le réseau de neurones s'est amélioré, nous avons représenté les individués conservés à chaque génération. Nous avons donné aux individués une base de données de 300 états d'entrée(chaque individu est un réseau de neurones) et calculé les sorties et les sorties souhaitées, décrites comme suit:

```
double cases[][] =
  {
      {4.375013190394657,-3.419603298472216, 0.0, 0.0, 0.0},
      {-1.4013042863851597,4.284759964524524,0.0, 0.0, 0.0},
      {4.653653035344373,4.874780337369209,  0.0, 0.0, 0.0},
      {0.8541570488311878,1.6084581070786736,0.0, 0.0, 0.0},
      {4.846509614332518,-1.4410091626106993,0.0, 0.0, 0.0},
      {0.39937836558904394,-4.691477707148027,0.0, 0.0, 0.0},
            ⋮                  ⋮          ⋮   ⋮   ⋮
```

Pour plus de précision, il est représenté par :

x_1	x_2	$F(x_1,x_2)$	$D(x_1,x_2)$	Error
4.375013190394657	-3.419603298472216	0.0	0.0	0.0
-1.4013042863851597	4.284759964524524	0.0	0.0	0.0

donc cela représente:

x_1, x_2 les entrées généré aléatoirement.

$F(x_1, x_2)$ c'est la valeur obtenue à partir du réseau de neurones.

$D(x_1, x_2)$ la valeur désirée est calculée par : $D(x_1, x_2) = \cos(x_1) * \sin(x_2)$.

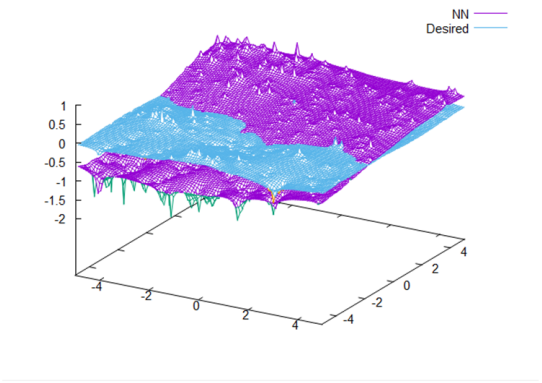
Error est $/ D(x_1, x_2) - F(x_1, x_2) /$.

Ensuite, nous calculons l'erreur pour le RN par :

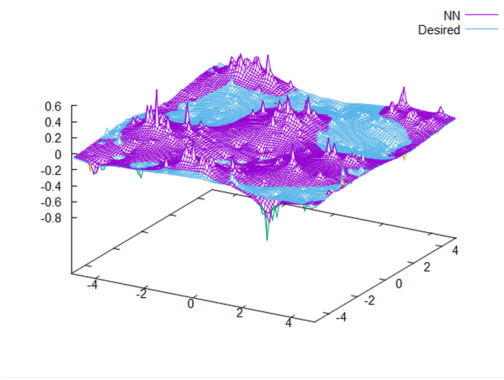
$$Err = \sum_{i=0}^{n} Error$$

n : Le nombre de cas ici est de 300.

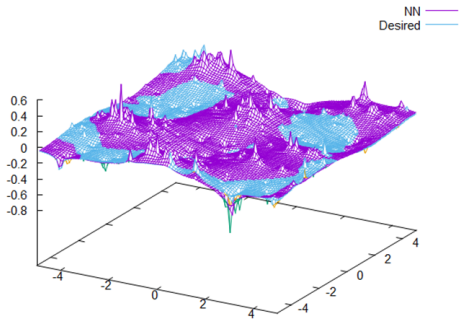
Nous représentons ces cas avec un programme appelé gnuplot ce sont les résultats:



Génération 1 , Err = 282.1356013307033	Génération 100 , Err = 91.84032878101098



Génération 500 , Err = 74.38473396316667	Génération 1000 , Err = 51.70122094735517



Génération 1500 , Err = 43.72978487296201	Génération 2000 , Err = 39.08986906708626

Où la courbe en bleu représente : x_1, x_2 de [-5,5] ➔ $D(x_1, x_2)$.

La courbe en violet représente : x_1, x_2 de [-5,5] ➔ $F(x_1, x_2)$.

La formation de cet exemple pourrait être améliorée si nous produisions plus de générations que nous n'en avons fait.

3.8 Conclusion

Après avoir appliqué l'algorithme pour optimiser l'entrainement des réseaux de neurone pour simuler et contrôler des systèmes non linéaires, les résultats de simulation ont été performants très satisfaisants pour la simulation et le contrôle.

Conclusion général

Les réseaux de neurones sont largement utilisé pour contrôler des systèmes complexes (notamment dans la robotique et les automates industrielles). Dans ce cas la, l'entrainement du réseau de neurones devient une tache difficile.

Dans notre travail on utilise un mécanisme évolutif pour surmonter la difficulté de l'entrainement des réseaux de neurones.

Nous avons résolu ce problème par une évolution de réseau neurone avec l'algorithme génétique (AG) pour augmenter la robustesse du processus de recherche des bonnes solutions et améliorer le processus d'entrainement.

Pour traiter ce thème, nous avons fait une étude détaillée sur les réseaux de neurones et l'algorithme génétique qui ont été l'objet de deux premiers chapitres. Nous avons présenté la conception et les résultats de l'application dans le troisième chapitre, où nous les avons combinés en transformant le réseau neuronal en un chromosome qui transporte toutes ses informations afin que l'application de l'algorithme génétique lui soit correcte et réussie.

L'étude expérimentale que nous avons présentée a montré que le développement du réseau de neurones utilisant l'algorithme génétique a donné des résultats très bons et satisfaisants.

Bibliographies

[1] M. Y. Ammar, "Mise en œuvre de réseaux de neurones pour la modélisation de cinétiques réactionnelles en vue de la transposition batch/continu," Ph.D. dissertation, 2007.

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

[3] Y. Djeriri, "Les réseaux de neurones artificiels," *mémoire fin d'etude, Sidi Bel Abbès, Algerie*, 2017.

[4] L. SOUISSI, "Entrainement d'un réseau de neurones par algorithme génétique."

[5] I. N. Da Silva, D. H. Spatti, R. A. Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves, "Artificial neural network architectures and training processes," in *Artificial neural networks*. Springer, 2017, pp. 21–28.

[6] L. Saludjian, "Optimisations en electrotechnique par algorithmes génétiques," Ph.D. dissertation, Institut National Polytechnique de Grenoble-INPG, 1997.

[7] A. Bezzini, "Commande prédictive non linéaire en utilisant les systèmes neuro-flous et les algorithmes génétiques," Ph.D. dissertation, Faculté des sciences et de la technologie UMKBiskra, 2013.

[8] L. Amiar, "Un système hybride ag/pmc pour la reconnaissance de la parole arabe," Ph.D. dissertation, Annaba, 2005.

[9] A. Terki, "Analyse des pérformances des algorithmes génétiques utilisant différentes techniques d'évolution de la population," 2009.

[10] A. BOUABDALLAH, "Application des algorithmes génétiques au dispatching économique et environnemental," Master's thesis, 2012.

[11] Z. Djessas, "Extraction des paramètres physiques d'une cellule solaire à deux exponentiels par la méthode des algorithmes génétiques," Ph.D. dissertation, 2018.

[12] F. MEHIDID, "Algorithme génétique."