République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie d'Oran, Mohamed Boudiaf

Faculté des Sciences
Département d'informatique

Spécialité : Informatique
Option : Reconnaissance des formes et Intelligence artificielle.

Mémoire en vue de l'obtention du diplôme de
MAGISTER

Présenté par:

M^r^ MEDJAHED Chahreddine

Thème

MISE EN POINT D'UN PROGRAMME D'OPTIMISATION POUR LE STOCKAGE D'ENERGIE THERMIQUE

Soutenu publiquement le :

Devant le jury :

Président	**Mr BENYETTOU Abdelkader.**	Professeur	USTO(MB)
Rapporteur	**Mr BENYETTOU Mohammed.**	Professeur	USTO(MB)
Examinateur	**Mr DJEBBAR Bachir.**	Professeur	USTO(MB)
Examinateur	**Mme FIZAZI Hadria.**	Maitre de conférences	USTO(MB)
Examinateur	**Mr BELKADI Khaled.**	Maitre de conférences	USTO(MB)

2012/2013

DEDICACES

Si je suis arrivé là c'est grâce à mes très chers parents que j'aime énormément et à qui je dois tout.

A ce propos, je dédie ce travail :

À la personne qui me manque le plus au monde mais qui est toujours présent dans mon cœur et mes pensées, à toi ***mon père*** *et j'espère que tu es fièr de moi .Que DIEU t'accorde sa sainte miséricorde et t'accueille dans son vaste paradis.*

À la personne que j'aime le plus et qui compte le plus dans ma vie, à la meilleure maman dans l'univers, à toi ***ma mère*** *en signe de ton amour, tes sacrifices que tu ne cesse pas d'en faire, ton affection, ta générosité et ta patience. Tu es la mère, le père, la grande sœur, l'amie et la confidente. J'espère qu'en réussissant ce travail je réalise une partie de tes rêves. Que DIEU te protège et te garde pour nous.*

À mes frères et mes sœurs, merci pour leurs soutiens et leurs encouragements tout au long de ma vie.

Enfin à toute personne qui m'a soutenu et aidé dans les moments les plus difficiles.

Je m'excuse auprès des personnes dont je n'ai pas cité leur nom, mais cette liste ne sera jamais exhaustive. En tout cas, j'adresse un grand salut à toutes les personnes qui m'aiment.

CHAHREDDINE

REMERCIEMENTS

*Je remercie **Dieu** tout puissant de m'avoir donné le courage et la patience pour bien mener ce travail, qu'il soit béni et glorifié.*

*Je tiens expressément et chaleureusement à remercier **mes parents** ainsi que ma petite famille pour leur soutien et leurs encouragements tout au long de ma vie.*

*Je tiens à remercier particulièrement mon encadreur Monsieur **BENYETTOU Mohamed**, professeur et responsable du laboratoire modélisation et simulation à USTO, pour le suivi et l'attention qu'il à apporté à mon travail.*

*Je souhaite exprimer mes gratitudes au président du jury, le professeur **BENYETTOU Abdelkader**, responsable du laboratoire signal image parole à USTO.*

*Je souhaite remercier également le professeur **DJEBBAR Bachir** professeur à USTO, pour l'honneur qu'il m'a fait en acceptant d'examiner ce travail.*

*Mes remerciements vont également à Madame **FIZAZI Hadria**, maitre de conférences à USTO pour l'intérêt qu'elle a apporté à ce travail.*

*Mes remerciements vont également à Monsieur **BELKADI Khaled**, maitre de conférences à USTO, d'avoir bien voulu consacrer du temps et de l'attention à mon travail.*

Un grand merci à toute personne ayant participé d'une façon ou d'une autre à la réalisation de ce travail.

TABLE DES MATIERES

Introduction générale

Chapitre I : *Les Méthodes D'optimisation*

I.1 Introduction…… 1
I.2 Notions de base ……. 2
I.2.1 Quelques définitions …… 2
I.2.2 optimisation combinatoire …… 2
I.2.3 Les heuristiques …… 3
I.2.4 Les métaheuristiques …… 3
I.2.4.a) Étymologie …… 3
I.2.4.b. Définition…… 4
I.2.4.c. Propriétés fondamentales et fonctionnement……. …… 4
I.2.4.d. classement…… 5
I.3 Les méthodes exactes…… 6
I.3.1 Programmation dynamique……. …… 6
I.3.2 La méthode Branch and Bound …… 7
I.4 Les méthodes approchées (heuristiques)…… 9
I.4.1 Le principe de recherche locale…… 9
I.4.2 Structure de voisinage et minimum locale …… 10
I.4.3 Les méthodes de recherche locale …… 11
I.4.3.1. La méthode de descente (hill climbing)…… 11
I.4.3.1.a. Inconvénients…… 12
I.4.3.1.b. Avantages …… 12
I.5. Les métaheuristique…… 12
I.5.1 La méthode du recuit simulé : (simulated annealing) …… 13
I.5.1.1 origines …… 13
I.5.1.2 Définition…… 13
I.5.1.3 Algorithme …… 14
I.5.1.3. a. Algorithme de Metropolis…… 14
I.5.1.3. b Algorithme du recuit simulé …… 14
I.5.1.4 Etat initial de l'algorithme …… 15
I.5.1.5 Variation de la température…… 15
I.5.1.5.a. décroissance par paliers …… 15
I.5.1.5.b Décroissance continue …… 15
I.5.1.6 Amélioration …… 16
I.5.1.7 Avantages et Inconvénients…… 16
I.5.1.7.a Avantages …… 16

I.5.1.7.b Inconvénients…… 16
I.5.1.c. Applications …… 16
I.5.2 La recherche Tabou …… 17
I.5.2.1 Définition de la recherche Tabou (RT) …… 17
I.5.2.2 Principe de la recherche Tabou …… 17
I.5.2.3 Algorithme générale de la recherche Tabou …… 18
I.5.3 Algorithmes génétiques …… 18
I.5.3.1 Terminologies…… 19
I.5.3.2 Mécanisme de fonctionnement…… 20
I.6.Conclusion …… 24

Chapitre II : *Réseaux de neurones*

II.1 Introduction…… 26
II.2. Historique …… 27
II.3. Le neurone biologique…… 28
II.4. Le neurone formel…… 28
II.5. Les réseaux de neurones formels…… 30
II.5.1. Les réseaux de neurones non bouclés …… 30
II.5.2. Les réseaux de neurones bouclés …… 31
II.6. Domaine d'applications des réseaux de neurones.…… 32
II.7. Apprentissage des réseaux de neurones…… 32
II.7.1 Apprentissage supervisé …… 33
II.7.2. Apprentissage non supervisé …… 33
II.7.3. Apprentissage hybride…… 33
II.8. Les méthodes d'apprentissage…… 33
II.9. Propriété fondamentale des réseaux de neurones formels…… 34
II.10. Quelques modèles des réseaux de neurones…… 35
II.10.1. Le perceptron multicouche…… 35
II.10.1.1. Algorithme du MLP …… 35
II.10.2. Les réseaux de neurones récurrents…… 38
II.10.2.1. Architecture de HOPFIELD…… 38
II.10.2.2. Réseaux d'Elman…… 39
II.10.3. Le réseau à fonction de base radiale …… 39
II.10.3.1. Les réseaux de neurones à régression généralisée…… 41
II.10.3.2. Les réseaux de neurones probabilistes (Probabilistic Neural Networks)…… 41
II.11.Conclusion…… 42

Chapitre III : *Solution des EDs par les réseaux de neurones*

III.1 Introduction…… 43
III.2. Formulation du problème …… 43
III.3. Le nouveau modèle …… 46
III.3.1. Problème de valeur initiale …… 47
III.3.2.Équations différentielles du second ordre …… 47
III.3.3. Système de problèmes aux valeurs initiales…… 48
III.3.4.Équations différentielles du quatrième ordre …… 50
III.4. Equations différentielles dépendante du temps…… 51
III.5. Les réseaux de neurones…… 52
III.6. Description générale de la méthode…… 54
III.7.Dérivation des dérivées partielles mixtes…… 55
III.8. Illustration de la méthode par des problèmes de test…… 56
III.8.1. Problèmes aux valeurs initiales pour les équations différentielles ordinaires…… 57
III. Conclusion …… 57

Chapitre IV : *Implémentation et résultats*

IV.1 Introduction…… 59
IV.2 Le matériel utilisé…… 59
IV.3 Environnement de développement …… 59
IV.4 Langage de programmation …… 59
IV.5 Présentation générale de l'application …… 59
IV.5.1 Conditions initiales et aux limites…… 60
IV.6 Résolution du problème …… 61
IV.7 Illustration de la méthode par des exemples …… 65
IV.7.1 Exemple 1…… 65
IV.7.1 Exemple 2…… 66
IV.7.1 Exemple 3…… 67
IV.8 Conclusion…… 69

Conclusion générale

Bibliographie

Introduction générale

De nombreuses méthodes ont été développées jusqu'à présent pour résoudre les équations différentielles. Certaines méthodes produisent une solution sous la forme d'un tableau contenant la valeur de la solution d'un groupe de points sélectionné. D'autres utilisent des fonctions de base pour représenter la solution sous forme analytique et transforment le problème initial dans un système d'équations linéaire. La plupart des travaux antérieurs dans la résolution des équations différentielles en utilisant les réseaux de neurones sont limités au cas de la résolution des systèmes d'équations linéaires algébriques qui résultent de la discrétisation du domaine. La solution d'un système d'équations linéaire est tracée sur l'architecture d'un réseau de neurones de type Hopfield. La minimisation de la fonction d'énergie du réseau fournit la solution du système d'équations.

Une autre approche pour la résolution d'équations différentielles ordinaires est basée sur le fait que certains types de spline, par exemple les B-splines linéaires, peuvent être obtenus par la superposition des fonctions d'activation linéaires. La solution d'une équation différentielle linéaire en utilisant les B-splines comme fonctions de base peut être obtenue en résolvant un système d'équations linéaire ou non linéaire afin de déterminer les paramètres des splines. Une telle forme de solution tracée directement sur l'architecture d'un réseau de neurones en remplaçant chaque spline avec la somme des fonctions d'activation linéaires qui correspondent aux unités cachées. Cette méthode tient compte des fonctions de base locales et nécessite en général plusieurs splines (et par conséquent les paramètres du réseau) afin de produire des solutions précises. En outre, il n'est pas facile d'étendre ces techniques à des domaines multidimensionnels.

Dans ce mémoire, nous considérons le problème sous un angle différent. Nous présentons une méthode générale pour résoudre les équations différentielles ordinaires (EDO) et les équations aux dérivées partielles (EDP) qui s'appuient sur les capacités d'approximation de la fonction des réseaux de neurones de type MLP et les résultats dans la construction d'une solution écrite sous une forme analytique fermée différentiable. Cette forme emploie un réseau de neurone direct que l'élément du rapprochement de base, dont les paramètres (poids et biais) sont ajustés pour minimiser la fonction d'erreur appropriée. Pour entraîner le réseau, nous utilisons des techniques d'optimisation, qui à leur tour exigent le calcul du gradient de l'erreur en ce qui concerne les paramètres du réseau. Dans l'approche présentée, la fonction du modèle est exprimée comme la somme

de deux termes: le premier terme satisfait les conditions initiales et aux limites et ne contient pas de paramètres réglables. Le second terme implique un réseau de neurones qui doit être formé de manière à satisfaire l'équation différentielle. Comme il est connu qu'un perceptron multicouche avec une couche cachée peut approcher toute fonction à une précision arbitraire, il est raisonnable de considérer ce type d'architecture de réseau comme un modèle candidat pour le traitement des équations différentielles.

Ce mémoire est composé de quatre chapitres dont le premier présent des généralités sur les méthodes d'optimisation, le deuxième chapitre est consacré aux réseaux de neurones et leurs applications. Le chapitre trois explique l'application des RNs pour la résolution des équations différentielles .Les résultat obtenus seront illustrés et discutés dans le dernier chapitre.

Chapitre I

Méthodes d'optimisation

I.1 Introduction

L'optimisation combinatoire occupe une place très importante en recherche opérationnelle, en mathématiques discrètes et en informatique en générale. Son importance se justifie d'une part par la grande difficulté des problèmes d'optimisation [1].Et d'autre part par de nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire[2] .Bien que les problèmes d'optimisation combinatoire soient souvent faciles à définir, ils sont généralement difficiles à résoudre .En effet, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les données[3].

Étant donnée l'importance de ces problèmes, de nombreuses méthodes de résolution ont été développées en recherche opérationnelle (RO).

Ces méthodes peuvent être classées sommairement en deux grandes catégories :

- Les méthodes exactes (complètes) : qui garantissent la complétude de la résolution.
- Les méthodes approchées (incomplètes):qui perdent la complétude pour gagner en efficacité.

Le principe essentiel d'une méthode exacte consiste à effectuer une énumération explicite de toutes les solutions (c'est-à-dire de les tester une à une, méthode envisageable pour tous les problèmes à variables à valeurs bornées). Cette méthode montre très vite ses limites dès que le nombre de variables augmente puisque sa complexité est en k^n où k représente le nombre de valeurs que peut prendre une variable et n le nombre de variables du problème [4].

Parmi les méthodes exactes, on trouve La programmation dynamique et la méthode Branch & Bound etc.

Par contre Les méthodes approchées qui a pour but de trouver une solution de bonne qualité (C'est-à-dire assez proche de l'optimale) en un temps de calcul raisonnable sans garantir l'optimalité de la solution obtenue. Elles sont fondées principalement sur diverses heuristiques, souvent spécifiques à un type de problème.

Les métaheuristiques constituent une autre partie importante des méthodes approchées et ouvrent des voies très intéressantes en matière de conception de méthodes heuristiques pour l'optimisation combinatoire et pour cela on constaté deux approches ont clairement démontré leur utilité dans de nombreux domaines.

La première de ces approches, appelée Recherche Locale ou bien (voisinage), est couramment utilisée de puis plus de 20 ans et comprend, entre autres :

1- La méthode de descente.
2- Le recuit simulé.
3- Recherche tabou.
4- Méthode du bruitage.
5- Les algorithmes d'acceptation avec seuil.

La deuxième approche, appelée Méthode Évolutive, est plus récente puisqu'elle date du début des années 90. Elle s'est particulièrement fait connaître par :

- Les algorithmes génétiques.
- Les algorithmes de colonies de fourmis.

I.2 Notions de base

I.2.1 Quelques définitions

- Une solution : est une affectation de toutes les variables du problème.
- Une solution optimale : est une solution de coût minimal (ou maximal).
    - Un mouvement : est une opération élémentaire permettant de passer d'une solution a une solution voisine (exemple : changer la valeur d'une variable, échanger deux variables).
    - Le voisinage : d'une solution est l'ensemble des solutions voisines, c'est à dire l'ensemble des solutions accessibles par un mouvement (et un seul).
- Un essai : est une succession de mouvements.
- Une recherche locale : est une succession d'essais. [5]

I.2.2 Optimisation combinatoire

On peut trouver de nombreuses définitions de problèmes d'optimisation combinatoire. En mathématiques, l'optimisation recouvre toutes les méthodes qui permettent de déterminer l'optimum d'une fonction, avec ou sans contraintes.

Dans le cas d'une minimisation, un problème d'optimisation se représentera de la forme suivante :

Minimiser $f(s)$ (f : fonction à optimiser).
Avec g(s)<=0 (m contraintes d'inégalités).
Et h(s)=0 (p contrainte d'égalités). [6]

En théorie, un problème d'optimisation combinatoire se définit par l'ensemble de ses instances, souvent infiniment nombreuses. Dans la pratique, le problème se ramène à résoudre numériquement l'une de ces instances, par un procédé algorithmique.

Á chaque instance du problème est associé un ensemble discret de solutions S, un Sous-ensemble X de S représentant les solutions admissibles (réalisables), ainsi qu'une fonction de coût f (appelée aussi fonction objectif). f assigne à chaque solution $s \in X$ le nombre $f(s)$.

Résoudre un problème d'optimisation combinatoire consiste alors à trouver une solution $s \in X$ optimisant la valeur de la fonction de coût f.
Formellement, on cherche donc $s^* \in X$ tel que $f(s^*) \leq f(s)$ pour tout $s \in X$.
Une telle solution s* s'appelle une solution optimale, ou un optimum global.

Il existe une seconde formalisation du problème d'optimisation combinatoire, plus générale, qui inclue la notion de contraintes et d'affectation de valeurs à des variables. En voici la définition. Soit:

- Un ensemble de variables $(x_1, x_2,\ldots,x_n)$.
- Un ensemble de domaines de définitions $(D_1, D_2,\ldots D_n)$.
    - Un ensemble de contraintes C sur les variables (par exemple, des inéquations, ou bien l'obligation que toutes les variables prennent des valeurs différentes)
- Une fonction objectif f que l'on cherche à minimiser : $f : D_1 \times D_2 \times \ldots \times D_n \rightarrow R$

L'ensemble S des solutions possibles peut alors se définir comme :

$S = \{(x_1, v_1),\ldots, (x_n, v_n)\}$ tels que $v_i \in D_i$, et tels que s satisfasse toutes les contraintes C}

Les problèmes d'affectation sous contraintes possèdent de nombreuses applications pratiques, comme l'affectation de ressources, le groupement, la classification, la planification, l'emploi du temps et l'ordonnancement, dans des domaines très variés… [7]

I.2.3 Les heuristiques

Les méthodes heuristiques sont particulièrement utiles pour les problèmes qui nécessitent une solution en temps réel (ou très court) ou pour résoudre des problèmes difficiles sur des instances numériques de grande taille. Certaines sont ciblées sur un problème particulier (heuristiques spécifique). [4]

I.2.4 Les métaheuristiques

I.2.4.1 Étymologie

Ce mot vient de la combinaison de deux mots grecs:

- Le verbe heuriskein : « trouver ».
- Le suffixe méta : « au-delà », « dans un niveau supérieur ». [4]

I.2.4.2 Définition

Une métaheuristique est constituée d'un ensemble de concepts fondamentaux qui permettent d'aider à la conception de méthodes heuristiques pour un problème d'optimisation. [4]

I.2.4.3 Propriétés fondamentales et fonctionnement

Elles sont :

- Efficaces : elles permettent d'explorer efficacement l'espace de recherche.
- Non déterministe : puisqu'elles ne donnent aucune garantie d'optimalité.

Elles possèdent deux autres propriétés intéressantes :

- Elles ont des mécanismes qui permettent d'éviter d'être bloqué dans des régions de l'espace de recherche. C'est leur capacité pour s'échapper d'un minimum local.
- Les métaheuristiques peuvent faire usage de l'expérience accumulée durant la recherche de l'optimum.

L'exécution des métaheuristiques se déroule en trois phases :

Diversification – Intensification – Mémoire.

- La diversification regroupe les actions ayant pour but de récolter des informations sur le problème dans un ensemble défini (ou espace local) lors de l'intensification.

- L'intensification vise à utiliser les informations de la mémoire et celles récoltées lors de la phase de diversification afin de définir les meilleurs espaces de recherche locaux futurs. Et de les faire parcourir de façon optimale par les fonctions objectif.
- La mémoire est le support de l'apprentissage permettant à l'algorithme de ne tenir compte que des zones où l'optimum est susceptible de se trouver et de garder en mémoire les résultats passés.

Les métaheuristiques progressent itérativement et alternativement entre les phases de diversification, d'intensification et d'apprentissage [4] [8].

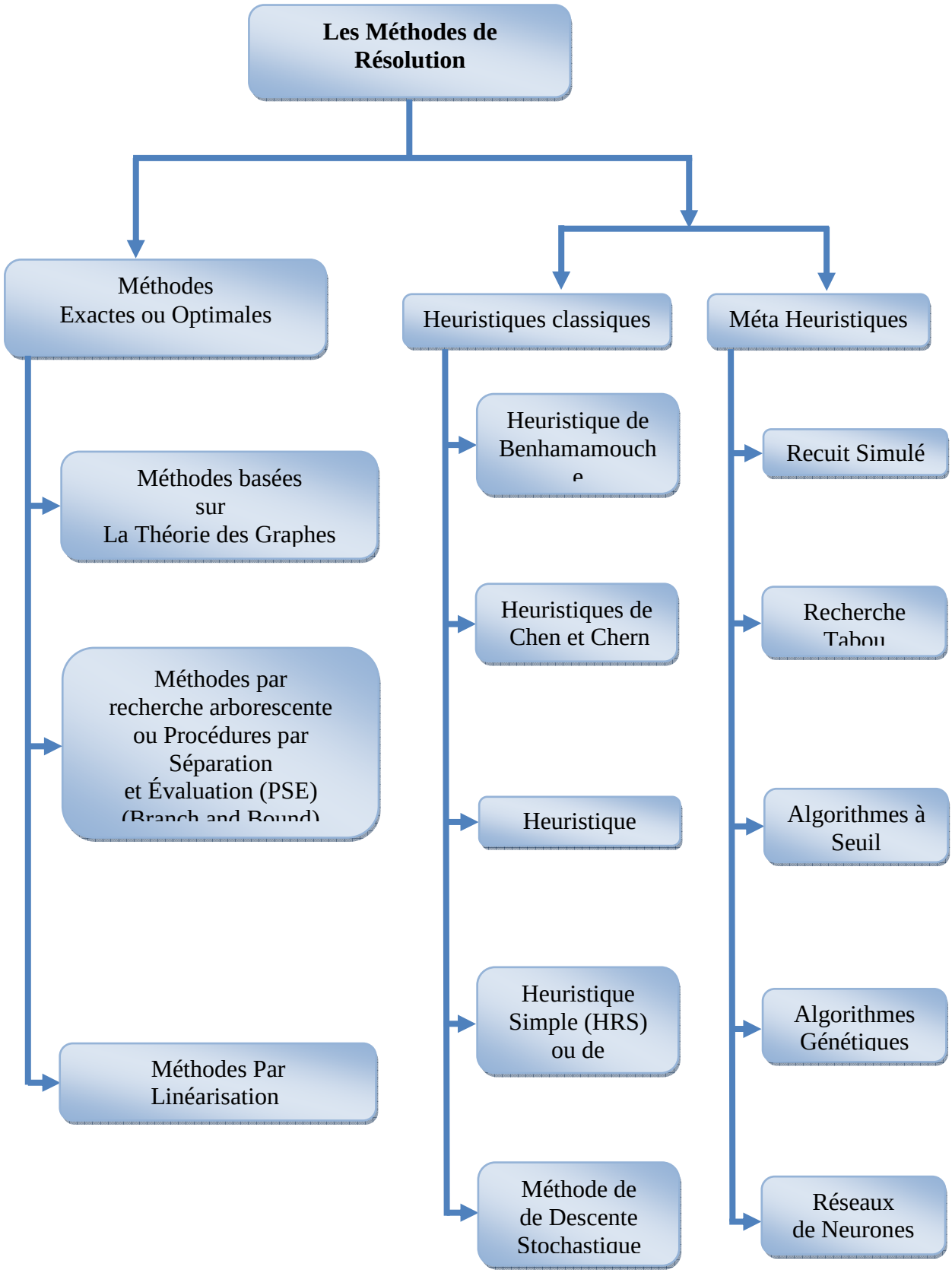


Figure I.1 : *Les différentes catégories de méthodes*

I.2.4.4 Classement

Pour classer l'ensemble des métaheuristiques, il existe en réalité plusieurs possibilités (ce qui est notamment ce qui dû au fait qu'elles sont le fruit de travaux en recherche opérationnelle et en intelligence artificielle). Le classement dépend donc du point de vue. Ci-après, une classification possible de ces algorithmes. On distingue :

- Constructives (algorithmes glouton, méthode Pilote, GRASP).
- Recherche locale (algorithmes de descente appelés aussi méthodes d'améliorations itératives, recuit simulé, algorithme a seuil, recherche Tabou, méthode de bruitage).
- Evolutionnistes (algorithmes génétiques, algorithmes d'évolution, recherche dispersée, méthode des chemins, systèmes de fourmis).
- Réseaux de neurones (Modèle de Hopfield, machine de Boltzmann, réseau auto-adaptatif, réseau élastique).

I.3 Les méthodes exactes

Les méthodes d'optimisation exactes ont l'objectif de déterminer l'optimum exact d'une fonction objectif. Ces méthodes sont basées soit sur une résolution algorithmique ou analytique, soit sur une énumération exhaustive de toutes les solutions possibles. Elles s'appliquent donc aux problèmes qui peuvent être résolus de façon optimale.

I.3.1 Programmation dynamique

Au début c'est le mathématicien américain Abraham Wald qui a proposé cette méthode qu'il nomme « analyse séquentielle », la méthode était introduite pour la première fois en 1940 dans un cadre militaire. En suite et en 1947, cette méthode est devenu accessible au grand public, et c'est Richard Bellman qui a donné l'importance au contrôle optimal et de manière plus générale aux problèmes d'optimisation séquentielle. C'est lui qui est à l'origine de l'appellation programmation dynamique. [9] Elle a connu un grand succès, car la plupart des fonctions économiques de l'industrie étaient à caractère optimal (maximisation ou minimisation). Il s'agit d'une façon particulière d'appréhender un problème algorithmique donné : une solution d'un problème dépend des solutions précédentes obtenues des sous-problèmes, autrement dit, c'est une adaptation de la méthode diviser et régner. [10]

C'est une technique dans laquelle on mémorise des résultats déjà obtenus pour les sous problèmes, et qu'on réutilise pour trouver de nouveaux résultats.

C'est une méthode utile pour obtenir une solution exacte à un problème algorithmique, là où une solution « classique » se trouve être trop complexe, c'est-à-dire trop peu efficace. On parle alors d'optimisation combinatoire.

L'efficacité de cette méthode repose sur le principe d'optimalité énoncé par le mathématicien Richard Bellman : « toute politique optimale est composée de sous-politiques optimales ». [11]

Principe

1. La solution d'un problème P_k contient les solutions de problèmes ayant des tailles plus petites.

2. La construction de la solution pour P_k s'effectue à l'aide d'un faible nombre d'informations sur P_j , $j < k$.

La programmation dynamique s'appuie sur un principe simple : toute solution optimale s'appuie elle-même sur des sous-problèmes résolus localement de façon optimale. Concrètement, cela signifie que l'on va pouvoir déduire la solution optimale d'un problème en combinant des solutions optimales d'une série de sous problèmes. Les solutions des problèmes sont étudiées 'de bas en haut', c'est-à-dire qu'on calcule les solutions des sous-problèmes les plus petits pour ensuite déduire petit à petit les solutions de l'ensemble. [12]

C'est une méthode ascendante **:** On commence d'habitude par les sous problèmes les plus petits et on remonte vers les sous problèmes de plus en plus difficile.

Pour résoudre un problème de taille n, on trouve, généralement, qu'il est composé de plusieurs sous problèmes identiques. Si on résout le même sous problème plusieurs fois, on obtient un algorithme inefficace. Par contre, si on résout chaque sous problème une seule fois(en sauvegardant les résultats) on obtient un algorithme performant. C'est l'idée de base de la programmation dynamique : Éviter de calculer deux fois la même chose, en utilisant une table de résultats, remplie au fur et à mesure qu'on résout les sous problèmes.

Algorithme de programmation dynamique

Etape 1

- Calculer pour chaque valeur possible de x_T :
- La solution optimale satisfait:
- $F_T(x_T) = Opt_{uT}\{f_T(x_T, u_T)\}$

Etape 2

- Calculer successivement, pour k = T -1,..., 1, et pour chaque valeur possible de x_k :
- $F_k(x_k) = Opt_{uk}\{f_k(x_k, u_k) + F_{k+1}(\mu_k(x_k, u_k)))\}$

Etape 3

- Calculer :
- $J^* = F^*(x_0) = Opt_{u0}\{f_0(x_0, u_0) + F_1(\theta_0(x_0,u_0)))\}$ [13]

I.3.2 La méthode Branch and Bound

La procédure de séparation et d'évaluation progressive, est une mise en application du principe latin "Divide and Regnes" qui conseille de diviser ses ennemis pour régner plus aisément.

La méthode de Séparation et d'Évaluation progressive (PSE), est une méthode exacte d'optimisation basée sur une énumération contrôlée qui fournit, en général, une solution optimale. Il s'agit d'une procédure d'optimisation donc de maximisation ou de minimisation d'une fonction f dite fonction objectif sur un domaine D de solutions possibles.

Principe

Cette méthode est basée sur les deux concepts suivants :

1. Le branchement ou bien la séparation, qui consiste à diviser un ensemble de solutions.
2. Estimant pour chaque sous-domaine $d_i \subset D$ de la partition une borne inférieure de F sur d_i (principe d'évaluation).

Algorithme général de la méthode

L'algorithme général correspondant à la procédure de séparation et d'évaluation (PSE) est le suivant :

Étape 1

Initialisation :

Calculer une borne supérieure BS (ou inférieur selon le critère à optimiser).

Étape 2

Séparation :

Sélectionner le sommet (l'espace de recherche) à séparer et créer ses fils (branches).

Étape 3

Évaluation :

Chercher une borne supérieure (inférieure) d'une fonction objectif relative à chaque sous-espace de recherche.

Étape 4

Élimination :

Éliminer les mauvais sous-espaces (suivant le cas : maximisation ou minimisation).

Étape 5

Critère d'arrêt

Si tous les sommets ont été éliminés alors arrêter.

Sinon aller à l'étape 2 jusqu'à obtenir l'optimum global.

Remarque : Impossible de donner un algorithme plus détaillé, chaque cas requiert du savoir faire.

I.4 Les méthodes approchées (heuristiques)

L'heuristique (du grec ancien « je trouve »), parfois orthographiée euristique, est un terme de didactique qui signifie l'art d'inventer, de faire des découvertes.

Une heuristique est un algorithme qui fournit rapidement (en temps polynomial) une solution réalisable, pas nécessairement optimale, pour un problème d'optimisation NP-difficile.

Une heuristique, ou méthode approximative, est donc le contraire d'un algorithme exact qui trouve une solution optimale pour un problème donné. Les algorithmes de résolution exacts étant de complexité exponentielle, il est généralement plus judicieux de faire appel à des méthodes heuristiques pour des problèmes difficiles.

On retiendra cependant que les méthodes de résolution exactes (comme le simplexe) sont de complexités exponentielles mais parfois plus efficaces en pratique qu'une

méthode heuristique. L'usage d'une heuristique est pertinent pour calculer une solution approchée d'un problème et ainsi accélérer le processus de résolution exacte.

Généralement une heuristique est conçue pour un problème particulier, en s'appuyant sur sa structure propre, mais les approches peuvent contenir des principes plus généraux. On parle de métaheuristiques pour les méthodes approximatives générales, pouvant s'appliquer à différents problèmes (comme le recuit simulé par exemple).

I.4.1 Le principe de recherche locale

Une méthode de recherche locale est basée sur l'évolution itérative d'une solution unique. Le passage d'une solution vers une autre se fait grâce à la définition de structure de voisinage qui est un élément très important dans la définition de ce type de méthode.

Nous résumons le principe dans les points suivants :

1. Partir d'une solution sinon approchée du moins potentiellement bonne et d'essayer de l'améliorer itérativement.

Pour améliorer une solution on ne fait que de légers changements (on parle de changement local, ou de solution voisine).

2. Relancer la méthode plusieurs fois en changeant le point de départ pour avoir plus de Couverture.

3. Tout problème est considéré comme un problème d'optimisation (même les problèmes de satisfaction : le coût à optimiser est alors le nombre de contraintes insatisfaites).

I.4.2 Structure de voisinage et minimum locale

Soit S un ensemble de solutions à un problème d'optimisation, et soit f la fonction objectif.

Une structure de voisinage (ou tout simplement un voisinage) est une fonction N qui associe un sous-ensemble de S à toute solution $s \in S$. Une solution $s' \in N(s)$ est dite voisine de s.

Une solution $s \in S$ est un minimum local relativement à la structure de voisinage N si $f(s) \leq f(s') \ \forall s' \in N(s)$.

Une solution $s \in S$ est un minimum global si $f(s) \leq f(s') \ \forall s' \in S$.

Certaines méthodes d'optimisation, qui partent d'une solution initiale et qui l'améliorent en explorant son voisinage immédiat, présentent l'inconvénient de s'arrêter au premier minimum local trouvé.

Comme nous le verrons plus loin, les métaheuristiques contiennent donc souvent une technique ou une astuce permettant d'éviter de se retrouver piégé dans ces minima locaux, en explorant davantage tout l'espace des solutions, de façon à augmenter la probabilité de rencontrer le minimum optimal, c'est-à-dire le minimum global.

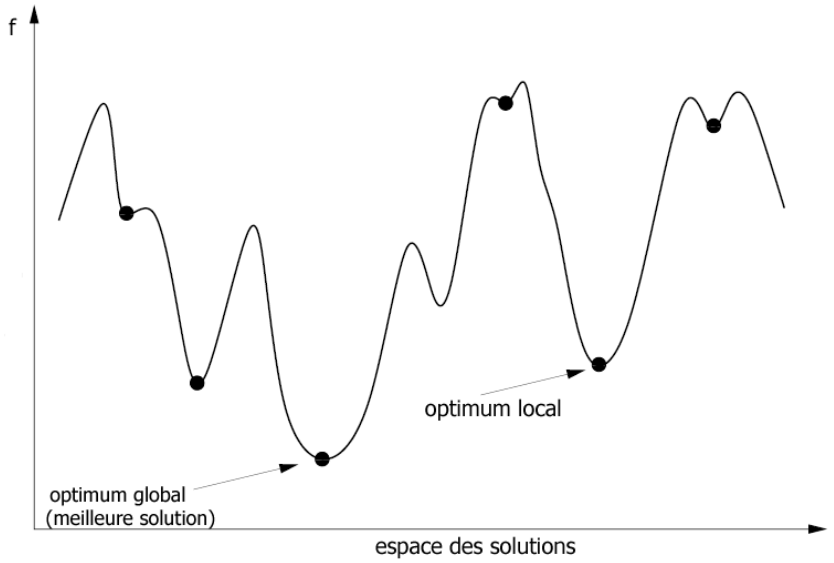


Figure I.2 : A*nalogie entre une fonction numérique à une variable et la fonction de coût d'un problème combinatoire.*

Dans le cadre de l'optimisation combinatoire, en pratique, on aura tout intérêt à définir le voisinage en considérant l'ensemble des modifications élémentaires que l'on peut appliquer à une solution s donnée, par exemple l'ensemble des permutations (si les solutions peuvent s'écrire sous la forme d'une séquence finie d'éléments, comme le cas se présente fréquemment en optimisation combinatoire).

Si cet ensemble est trop grand, on pourra toujours le réduire à un sous-ensemble, aléatoirement, ou en fonction d'un critère précis.

I.4.3 Les méthodes de recherche locale

I.4.3.1 La méthode de descente (hill climbing)

Le principe de la méthode de descente (dite aussi basic local search) ou bien (Méthode de voisinage) consiste à partir d'une solution s et à choisir une solution s' dans un voisinage de s, telle que s'améliore la recherche (généralement telle que f (s') < f(s)).

On peut décider soit d'examiner toutes les solutions du voisinage et prendre la meilleure de toutes (ou prendre la première trouvée), soit d'examiner un sous-ensemble du voisinage.

La méthode de recherche locale la plus élémentaire est la méthode de descente. On peut la schématiser comme suit [14] :

Procédure descente simple (solution initiale s)
Répéter :
Choisir s' dans N(s)
Si f (s') < f(s) alors s ← s'
Jusqu'à ce que f (s') ≥ f(s), $\forall$ s'∈ S
Fin

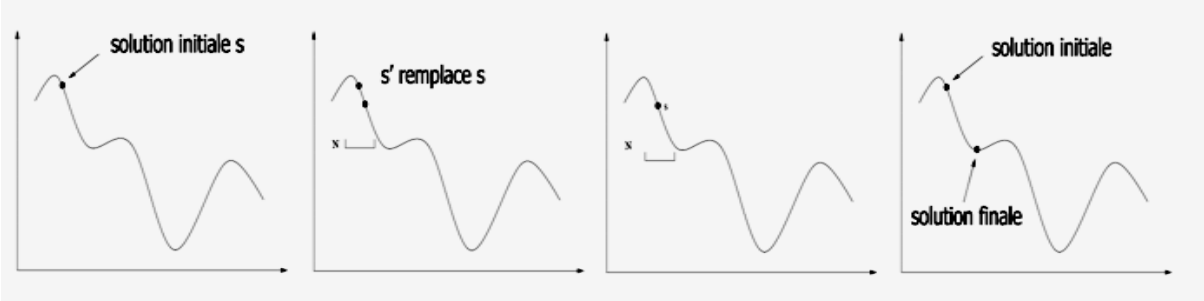


Figure I.3 : E*volution d'une solution dans la méthode de descente*

On peut varier cette méthode en choisissant à chaque fois la solution s' dans N(s) qui améliore le plus la valeur de f. C'est la méthode de plus grande descente.

I.4.3.1.a) Inconvénients

En général, l'efficacité des méthodes de recherche locale simples (descente, ou plus grande descente) est très peu satisfaisante. D'abord, par définition, la recherche s'arrête au premier minimum local rencontré, c'est là leur principal défaut. Pour améliorer les résultats, on peut lancer plusieurs fois l'algorithme en partant d'un jeu de solutions initiales différentes, mais la performance de cette technique décroît rapidement.

En revanche, autoriser de temps à autre une certaine dégradation des solutions trouvées, afin de mieux explorer tout l'espace des configurations, a conduit au développement des deux méthodes que nous explorons aux paragraphes suivants, à savoir le recuit simulé et la méthode Tabou.

I.4.3.1.b) Avantages

Le principal avantage de la recherche locale simple est évidemment sa grande simplicité de mise en œuvre: la plupart du temps, elle ne fait que calculer f(s+i)-f(s), où i correspond à un déplacement élémentaire, et si cette expression peut se simplifier algébriquement, alors on pourra évaluer très rapidement cette différence.

Il est important de remarquer également l'importance du choix de la fonction de voisinage N: un minimum local pour une certaine structure de voisinage ne l'est pas forcément pour une autre. C'est d'ailleurs ce constat qui est à l'origine de la méthode dite de recherche par voisinage variable, qui repose sur la construction de solutions s parmi plusieurs voisinages N_i, plutôt que dans un seul.

I.5 Les métaheuristiques

Les métaheuristiques sont une nouvelle génération de méthodes approchées puissantes et générales, qui sont constituées d'un ensemble de concepts fondamentaux et qui permettent d'aider à la conception des méthodes heuristiques pour un problème d'optimisation, ainsi les métaheuristiques sont adaptables et applicables à une large classe de problèmes.

Grâce à ces métaheuristiques, on peut proposer aujourd'hui des solutions approchées pour des problèmes d'optimisation classiques de plus grande taille et pour de très nombreuses applications qu'il était impossible de traiter auparavant, comme on constate, depuis ces dernières années, que l'intérêt porté aux métaheuristiques augmente continuellement en recherche opérationnelle et en intelligence artificielle. [12]

I.5.1 La méthode du recuit simulé (simulated annealing)

I.5.1.1 Origine

La méthode du recuit simulé est une généralisation de la méthode Monte-Carlo ; son but est de trouver une solution optimale pour un problème donné.

Elle a été mise au point par trois chercheurs de la société IBM : S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi en 1983, et indépendamment par V. Cerny en 1985 à partir de l'algorithme de Metropolis, qui permet de décrire l'évolution d'un système thermodynamique. [15]

La méthode du recuit simulé est basée sur un processus très utilisé en métallurgie pour obtenir un alliage sans défaut, ce processus est appelé « le recuit ». [16].

On commence d'abord par chauffer le métal jusqu'à une certaine température où il devient liquide (les atomes peuvent donc circuler librement). Après avoir atteint ce stade, on abaisse la température très lentement de sorte à obtenir un solide [17]. Si cette baisse de température est brusque on obtient alors du verre ; si au contraire cette baisse de température est très lente (laissant aux atomes le temps d'atteindre l'équilibre statistique), nous obtiendrons des structures de plus en plus régulières, jusqu'à atteindre un état d'énergie minimale correspondant à la structure parfaite d'un Crystal, on dit alors que le système est « gelé ».

Au cas où cet abaissement de température ne se ferait pas assez lentement, il pourrait apparaitre des défauts. Il faudrait alors les corriger en réchauffant de nouveau légèrement la matière de façon à permettre aux atomes de retrouver la liberté de mouvement, leur facilitant ainsi un éventuel réarrangement conduisant à une structure plus stable. [17]

I.5.1.2 Définition

L'idée principale du recuit simulé tel qu'il a été proposé par Metropolis en 1953 est de simuler le comportement de la matière dans le processus du recuit très largement utilisé dans la métallurgie. Le but est d'atteindre un état d'équilibre thermodynamique, cet état d'équilibre (où l'énergie est minimale) représente - dans la méthode du recuit simulé - la solution optimale d'un problème ; L'énergie du système sera calculé par une fonction coût (ou fonction objectif) spécifique à chaque problème (Kendall). La méthode va donc essayer de trouver la solution optimale en optimisant une fonction objectif, pour cela, un paramètre fictif de température a été ajouté par Kirkpatrick, Gelatt et Vecchi.

En gros le principe consiste à générer successivement des configurations à partir d'une solution initiale S_0 et d'une température initiale T_0 qui diminuera tout au long du processus jusqu'à atteindre une température finale ou un état d'équilibre (optimum global).

I.5.1.3 Algorithme

a) Algorithme de Metropolis

Dans l'algorithme de Metropolis, on part d'une configuration donnée, et on lui fait subir une modification aléatoire. Si cette modification fait diminuer la fonction objectif (ou énergie du système), elle est directement acceptée ; Sinon, elle n'est acceptée qu'avec une probabilité égale à $\exp(\Delta E/T)$ (avec E=énergie, et T=température), cette règle est appelée critère de Metropolis. [16].

b) Algorithme du recuit simulé

Le recuit simulé applique itérativement l'algorithme de Metropolis, pour engendrer une séquence de configurations qui tendent vers l'équilibre thermodynamique:

1) Choisir une température de départ $T=T_0$ et une solution initiale $S=S_0$.
2) Générer une solution aléatoire dans le voisinage de la solution actuelle.
3) Comparer les deux solutions selon le critère de Metropolis.
4) Répéter 2 et 3 jusqu'a ce que l'équilibre statistique soit atteint.
5) Décroitre la température et répéter jusqu'a ce que le système soit gelé. [18]

Dans un premier temps, T étant généralement choisi très grand, beaucoup de solutions - même celles dégradant la valeur de f - sont acceptées, et l'algorithme équivaut à une visite aléatoire de l'espace des solutions. Mais à mesure que la température baisse, la plupart des solutions augmentant l'énergie sont refusés, et l'algorithme se ramène à une amélioration itérative classique.

Á température intermédiaire, l'algorithme autorise de temps en temps des transformations qui dégradent la fonction objectif. Il laisse ainsi une chance au système de s'extraire d'un minima local. [16]

Notons aussi que si la température est égale à 0, seules les solutions optimisant f sont acceptées. L'algorithme se comportera donc comme la méthode de la descente du gradient.

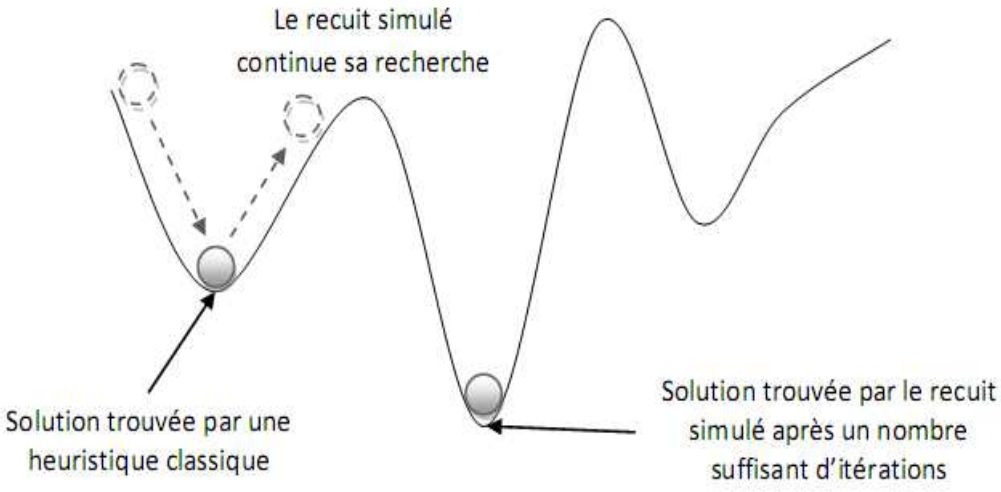


Figure I.4 : *comparaison entre le recuit simulé et une heuristique classique*

I.5.1.4 Etat initial de l'algorithme

La solution initiale peut être prise au hasard dans l'espace des solutions possibles, elle peut aussi être générée par une heuristique classique, telle que la descente du gradient ou l'algorithme glouton. [19]

La température initiale doit être assez élevée, car c'est elle qui fixe la probabilité d'accepter ou de refuser les solutions défavorables à l'optimisation de la fonction f.

I.5.1.5 Variation de la température

Deux approches sont possibles pour décroitre la température :

a) Décroissance par paliers

Pour chaque valeur de la température, on itère l'algorithme de Metropolis jusqu'a atteindre un équilibre statistique, puis on diminue la température.

b) Décroissance continue

On fait baisser la température d'une façon continue, le plus courant est d'utiliser la loi suivante : $T_{i+1} = \alpha . T_i / \alpha < 1$ (en général $\alpha = 0.9$ à 0.99).

Remarque: Le paramètre α est à choisir avec précaution, En effet, s'il est choisi trop grand, la température baissera très rapidement et l'algorithme pourra être bloqué dans un minima local. Si au contraire il est choisi trop petit, la température baissera très lentement et le temps de calcul sera très grand. [19]

I.5.1.6 Amélioration

Cet algorithme est parfois amélioré en ajoutant une variable qui mémorise la meilleure valeur rencontrée jusqu'à présent, sans cela, l'algorithme pourrait converger vers une certaine solution, alors qu'on avait visité auparavant une solution meilleure. [16].

I.5.1.7 Avantages et Inconvénients

a) Avantages

- Facile à implémenter;
- Donne généralement de bonnes solutions par rapport aux algorithmes de recherche classiques.
- Peut être utilisé dans la plupart des problèmes d'optimisation.
- Il converge vers un optimum global (lorsque le nombre d'itérations tend vers l'infini [16]).

Cela fait de lui une option attrayante pour les problèmes d'optimisation difficiles.

b) Inconvénients

Le principal inconvénient du recuit simulé est qu'une fois l'algorithme piégé à basse température dans un minimum local, il lui est impossible de s'en sortir. Plusieurs solutions ont été proposées pour tenter de résoudre ce problème, par exemple en acceptant une brusque remontée de la température de temps en temps, pour relancer la recherche sur d'autres régions plus éloignées. [16].

Appart cela on peut citer quelques autres inconvénients comme :

- La difficulté de déterminer la température initiale :
  - Si elle est trop basse, la qualité de recherche sera mauvaise.
  - Si elle est trop haute, le temps de calcul sera élevé.
- L'impossibilité de savoir si la solution trouvée est optimale.
- Dégradation des performances pour les problèmes où il y a peu de minimas locaux (comparé avec les heuristiques classiques comme la descente du gradient par exemple).

I.5.1.8 Applications

L'algorithme du recuit simulé a montré son efficacité sur les problèmes combinatoires classiques, surtout sur les échantillons de grande taille. Par exemple, l'expérience a montré qu'il ne devenait vraiment efficace sur le problème du voyageur de commerce

qu'au-delà d'environ 800 villes. Il a été testé avec succès sur le problème du partitionnement de graphe.

Des analyses ont montré qu'il était efficace avec certaines catégories de problème où l'ensemble des solutions possèdent certaines propriétés particulières. Ceci expliquerait le succès du recuit simulé dans le domaine du placement des circuits électroniques, où il est très employé.

I.5.2 La recherche Tabou

La méthode Tabou a été introduite simultanément par plusieurs (P.Hansen, F.Glover, B.Jaumard) aux environs de 1986.

Recherche Tabou parce qu'il y a interdiction de reprendre des solutions récemment visitées.

A chaque itération, le moins mauvais voisin est choisit.

Pour éviter les cycles, c'est à dire la répétition infinie d'une séquence de mouvements, les L derniers mouvements sont considérés comme interdits, L étant la taille de la liste Tabou.

À chaque itération, le mouvement effectué est donc le moins mauvais mouvement non Tabou.

I.5.2.1 Définition de la recherche Tabou (RT)

Le mot Tabou vient du Tonga polynésien. Le Tonga indique une chose qui ne peut pas être touchée parce qu'elle est sacré, la signification du mot Tabou (Tabu en anglais) est interdit.

La recherche Tabou est une métaheuristique originalement développée par Glover et indépendamment par Hansen.

C'est une méthode utilisée pour la résolution des problèmes d'optimisation, destinée principalement à guider d'autres méthodes afin de trouver de meilleures solutions à partir d'une solution initiale obtenue par l'une des heuristiques.

La méthode peut être utilisée pour guider n'importe quel processus qui emploie un ensemble de mouvements pour passer d'une solution à une autre et qui fournit une fonction d'évaluation pour mesurer la qualité de ces mouvements.

I.5.2.2 Principe de la recherche Tabou

Le principe de la recherche Tabou repose sur une méthode de déplacement sur l'espace des solutions, tout en cherchant constamment à améliorer la meilleure solution courante et en conservant en mémoire la liste des précédents déplacements et ainsi guider la recherche en dehors de zones précédemment parcourues.

L'idée de base s'inspire des techniques de recherche utilisées en intelligence artificielle. Elle consiste à garder la trace du cheminement passé du processus de recherche dans une ou plusieurs mémoires et à se servir de cette information afin d'en orienté le déroulement futur.

Principe de base : poursuivre la recherche de solutions même lorsqu'un optimum local est rencontré et ce :

- En permettant des déplacements qui n'améliorent pas la solution.
- En utilisant le principe de mémoire pour éviter les retours en arrière (mouvements cycliques).

La RT est basée sur :

- L'utilisation de structures de mémoires flexibles (court, moyen, long terme) permettant l'exploration complète du critère d'évaluation et aussi de l'historique de la recherche.
- Un mécanisme de contrôle basé sur l'alternance entre les conditions qui restreignent (restriction Tabou) et qui libèrent (critère d'aspiration) le processus de recherche.
- L'incorporation des stratégies dites d'intensification et de diversification de la recherche.
    - La stratégie d'intensification utilisant la mémoire à moyen terme, sert à renforcer la recherche dans la région des meilleures solutions trouvées récemment.
    - La stratégie de diversification utilisant la mémoire à long terme, sert à guider la recherche dans de nouvelles régions.

I.5.2.3 Algorithme général de la recherche Tabou

Nous présentons ci-dessous l'algorithme général de la recherche Tabou, en mettant l'accent sur la mémoire à court terme et l'exploration agressive :

1) Obtenir une solution initiale.
2) Créer une liste des mouvements candidats.
3) Choisir le meilleur candidat. Ce choix est basé sur les restrictions Tabou et le critère d'aspiration.
    - On obtient ainsi une autre solution, mais qui ne sera enregistrer que si elle est meilleur que la solution précédente.
4) Appliquer le critère d'arrêt.
    - Continue : changer les candidats d'admissibilité (restriction Tabou et critère d'aspiration). Aller à 2.
    - Stop : passer aux stratégies d'intensification et de diversification.

I.5.3 Algorithmes génétiques

Les algorithmes génétiques (AGs) appartiennent à la famille des algorithmes appelée métaheuristiques évolutives. Le but de cette catégorie d'algorithmes est d'obtenir une solution convenable à un problème d'optimisation en un temps correcte (admissible). De même, le but des algorithmes génétiques est de se rapprocher au maximum d'une solution optimale, sans malheureusement garantir si elle est la meilleure. [20] [21][22].

I.5.3.1Terminologies

Les algorithmes génétiques sont directement dérivés de l'évolution de la nature. Pour mieux comprendre leurs théories, il est important d'introduire quelques terminologies biologiques. [23]

Gène: Est l'unité responsable de l'hérédité qui contrôle les caractères et/ou les aptitudes propres à un organisme. Les chromosomes se décomposent en gènes qui peuvent prendre des valeurs différentes que l'on appelle allèles.

Individu: Les chaînes des systèmes génétiques artificiels sont analogues aux chromosomes des systèmes biologiques. Ils portent les informations génétiques d'un individu.

Population: les AGs n'opèrent pas sur un seul individu à la fois, mais sur une population d'individus afin d'effectuer des opérations de recherches sur un domaine de possibilités plus importantes. A chaque instant t, la population est appelée «génération».

Reproduction: est une opération qui permet de choisir parmi une population les individus qui vont survivre à l'évolution ou mourir. Les plus adaptés ont toutes les chances d'êtres reproduits. Les moins adaptés vont mourir.

Fonction d'adaptation (fitness): est une fonction qui mesure l'adaptation d'un individu i à l'instant t. Il s'agit d'une valeur estimée à partir de la fitness de l'individu avec l'environnement.

Le codage: un chromosome est une représentation génétique d'une solution candidate. Cette représentation est généralement codée par des nombres binaires, ou directement par des chiffres entiers ou réels.

Sélection: choix des individus pour la reproduction.

Croisement: la recherche des nouvelles solutions s'effectue grâce à l'opérateur de croisement. Parmi les individus reproduits, on va apparier des individus et les croiser pour élargir le domaine de recherche de la solution optimale.

Mutation: elle change aléatoirement le nouvel individu en choisissant des bits aléatoirement et en changeant leurs valeurs de 0 à 1 ou de 1 à 0.

I.5.3.2 Mécanisme de fonctionnement

Les algorithmes génétiques opèrent sur une population de solutions possibles, où chaque solution est représentée par un chromosome (la représentation abstrait de la solution). Donc, coder les solutions possibles sous forme de chromosome est la première étape. Par la suite, un ensemble d'opérations de reproduction vont être appliquées sur les chromosomes, pour aboutir à la création de nouvelles solutions du problème. Nous allons introduire les différentes étapes du processus d'exécution d'un algorithme génétique:

a) La représentation génétique

Comme première étape avant l'application des algorithmes génétiques sur un problème, il est nécessaire de définir une représentation génétique des solutions. Il faut trouver une structure pour les chromosomes afin de coder les solutions, cette structure est appelée la représentation génétique. Le choix du codage est lié à la formalisation

mathématique du problème. Cependant, il n'existe aucunes règles qui imposent le codage à choisir. Parmi les codages que nous pouvons utiliser:

- Codage Binaire.
- Codage à caractères multiples.
- Codage sous forme d'arbre.

b) Fonction d'adaptation (Fitness)

Une fois la représentation des solutions est déterminée, la deuxième étape consiste à associer pour chaque solution (chromosome) une valeur d'évaluation qui correspond à sa fitness, et qui indiquera la qualité de l'individu. La fonction de fitness est déterminée selon la nature du problème, elle dépend de ce qu'on veut optimiser. Cependant, le choix de cette fonction affecte les résultats de la sélection des individus qui contribueront à la création des nouveaux individus.

Généralement, les individus ayant une fitness supérieure vont avoir une grande probabilité de se reproduire. En d'autres termes, ils auront plus de chances pour contribuer à la création d'une nouvelle solution.

Souvent, les problèmes pratiques consistent à optimiser une solution en suivant plusieurs critères, ce qui rend la fonction de fitness difficile à déterminer. Dans les problèmes multicritères, il est souvent si difficile de déterminer si une solution est meilleure qu'une autre. Par exemple, une solution est meilleure dans un critère par rapport à une autre, mais moins bonne dans un autre critère. La résolution de ce problème est souvent traitée lors de la modélisation, où le concepteur de l'algorithme génétique doit déterminer les stratégies à suivre.

c) Population initiale

Comme l'indique la définition des algorithmes génétiques, les AGs opèrent sur des populations d'individus, et non pas sur un seul individu. A partir d'une population initiale de solutions, les algorithmes génétiques créent de nouvelles générations de solutions améliorées, qui participeront eux aussi à la génération de nouvelles populations. Cependant, il est nécessaire de choisir une population initiale avant de lancer le processus des algorithmes génétiques. Donc, comment choisir cette population ? Et qu'elle est sa taille ?

Le choix de la population initiale est très crucial à la réussite du processus des AGs, la diversité de l'ensemble des individus initiale influe sur la qualité des résultats obtenus. Il est recommandé de bien choisir les individus initiaux, ainsi que leurs nombre (taille de la population), afin de pouvoir exploiter le maximum de l'espace de recherche.

Généralement les individus initiaux sont choisis aléatoirement. Cependant, par fois des heuristiques peuvent êtres utilisées pour déterminer ces individus initiaux. Par exemple, choisir des individus ayant une grande fitness. Mais tout en conservant la diversité. Le manque de diversité peut mener à exploiter qu'une partie de l'espace de recherche, ce qui peut engendrer la convergence vers une solution optimale mais locale (l'optimum d'un sous ensemble de l'espace de recherche), qui ne sera pas forcément l'optimale de l'ensemble global.

d) Sélection d'individus

Le Rôle de la sélection est de choisir les individus pour participer à la prochaine étape qui est la génération de nouveaux individus (création d'une nouvelle population). Entre autre, éliminer les individus qui sont moins bons, et ne doivent pas participer à la reproduction. La sélection se base sur la valeur de la fonction fitness des individus. Lors de la sélection, un individu peut être sélectionné une ou plusieurs fois, comme il peut ne pas être sélectionné du tout. Il existe plusieurs stratégies de sélection. Parmi les quels on cite: [25]

- Sélection par roulette (wheel)
- Sélection par rang
- Steady State
- Sélection par tournoi
- Sélection par élitisme

e) Croisement

Le croisement, ou bien la recombinaison, est le processus de produire un ou plusieurs individus fils à partir de deux parents. Il existe plusieurs manières de faire la reproduction, parmi lesquelles on trouve la plus connue qui est le croisement multipoints [24]. Le principe de ce croisement consiste à prendre deux chromosomes parents, les découper en N+1 segment par N points (choisis aléatoirement ou suivant une stratégie déterminée par le concepteur de l'AG).

Donc, un fils sera créé en choisissant alternativement des segments de chaque parent. Cette opération est illustrée dans l'exemple suivant:

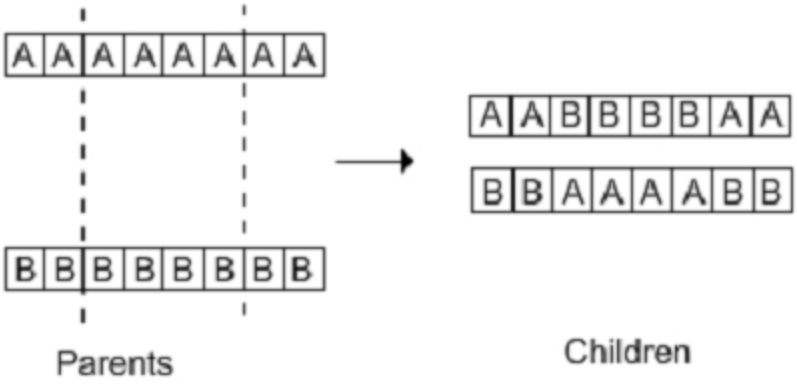


Figure I.5 : *Croisement multipoints*

De plus que le croisement multipoints, il existe un autre type de croisement qui est le croisement uniforme [24]. Le principe de ce croisement consiste à faire copier un gène de chaque parent en utilisant un masque de croisement. Ce qui est illustré dans l'exemple suivant:

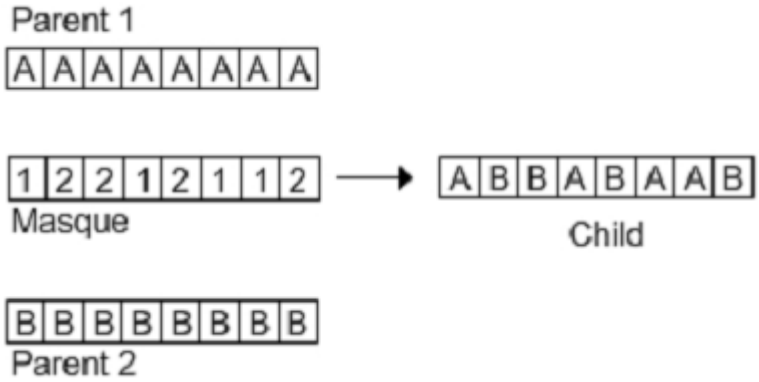


Figure I.6 : *Croisement uniforme*

f) Mutation

La mutation est une simple opération qui consiste à modifier les valeurs des gènes d'un chromosome, elle est utilisée dans le cas où les nouveaux individus nécessitent des modifications. D'un autre coté, la mutation peut garantir l'exploitation de tout l'espace de recherche par le maintient de la diversité de la population [22].

Il existe plusieurs formes de mutation pour les différents types de représentation des individus. Par exemple, pour la représentation binaire, la mutation consiste à inverser la valeur des gènes comme illustré dans l'exemple:

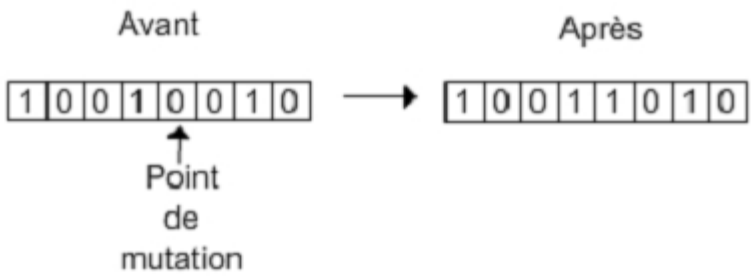


Figure I.7 : *Mutation de bits*

g) La convergence

L'évolution se montre toujours comme étant un processus sans fin. Les algorithmes génétiques ne peuvent jamais s'arrêter sur une solution optimale globale du problème. Donc l'évolution doit s'arrêter à un certain point déterminé en fonction de certains critères. Pour cela, il existe plusieurs façons pour décider la contrainte d'arrêt de l'algorithme, et qui est déterminée par le concepteur selon les besoins du problème. Par exemple, la contrainte la plus simple consiste à arrêter le processus après un certain nombre d'itérations. Une autre méthode consiste à laisser les itérations jusqu'à l'observation de certains changements, ou jusqu'à ce que la population se stabilise (les individus deviennent tous identiques). Donc, le choix de cette condition est souvent lié au problème.

h) Avantages et inconvénients

Parmi les Avantage des algorithmes génétiques est le fonctionnement sur de divers types de problèmes. D'ailleurs, les AGs fonctionnent aussi bien sur les types de problèmes multicritères que sur les problèmes monocritère. De plus, leur façon de fonctionnement permet la parallélisassions du calcul, ce qui améliore davantage le temps de calcul [26].

Les algorithmes génétiques permettent une grande liberté dans le paramétrage et l'implémentation des différents traitements. Ce qui est aussi considérer comme un désavantage, du à la difficulté de choisir pertinemment les bons paramètres pour les

divers opérateurs (population initiale, fonction de fitness, croisement, mutation, sélection, et remplacement) [22] afin d'obtenir le meilleur résultat.

Un autre désavantage consiste en ce qui concerne la convergence de l'algorithme. Il n'y a aucune garantie de convergence, même pas vers un minimum local [26]. Néanmoins, ils peuvent être combinés avec d'autres méthodes de recherche locale afin d'améliorer les résultats [22].

I.6 Conclusion

Les métaheuristiques étant très généralistes, elles peuvent être adaptées à tout type de problème d'optimisation, elles sont souvent moins puissantes que des méthodes exactes sur certains types de problèmes.

Elles ne garantissent pas non plus la découverte de l'optimum global en un temps fini.

La notion d'efficacité se rapporte généralement à deux objectifs contradictoires : la vitesse et la précision.

1) La vitesse est souvent mesurée en nombre d'évaluations de la fonction objectif.

2) La précision se rapporte à la distance entre l'optimum trouvé par la métaheuristique et l'optimum réel, alors un algorithme rapide est peu précis, et inversement.

Généralement, un choix doit être fait quant au critère d'arrêt adéquat. Un nombre d'évaluation ou un temps imparti est souvent utilisé, mais on peut également choisir d'atteindre une valeur donnée de la fonction objectif, le choix du critère d'arrêt influencera sur la qualité de l'optimisation.

Il est admis que, d'un point de vue très général, aucune métaheuristique n'est réellement meilleure qu'une autre. En effet, une métaheuristique ne peut prétendre être plus efficace sur tous les problèmes, mais un choix de paramètres et une implémentation donnée puissent être plus adaptées que d'autres sur certaines classes de problèmes.

Chapitre II

Réseaux de neurones

II.1 Introduction

L'intelligence artificielle est en progression depuis l'invention de l'ordinateur et l'utilisation des programmes informatiques. Il existe en effet de nombreux programmes capables de réaliser des choses de plus en plus complexes : diriger un robot, Résoudre des problèmes, joués aux échecs, etc. Mais ils sont très rarement capables de rivaliser avec le cerveau humain, et c'est pour cela que de nombreuses tâches sont encore irréalisables par les ordinateurs.

D'où vient cette différence, la première chose est évidemment la capacité de calcul.

Le cerveau est une machine assurément incroyable. Cependant, les ordinateurs n'ont pas cette faculté d'apprentissage, ils ne connaissent pas le progrès si personne ne les modifie.

Voilà le challenge pour l'intelligence artificielle : savoir apprendre.

D'un autre côté, La biologie a apporté un grand nombre d'informations sur le fonctionnement du cerveau, des neurones, etc. Des mathématiciens ont alors tenté de reproduire le fonctionnement du cerveau en intégrant ces connaissances en biologie dans des programmes informatiques, et en leur donnant la possibilité d'apprendre.

D'autre part, les réseaux de neurones, fabriqués de structures cellulaires artificiels, constituent une approche permettant d'aborder sous des angles nouveaux les problèmes de perception, de mémoire, d'apprentissage et de raisonnement. Ils s'avèrent aussi des alternatives très prometteuses pour contourner certaines des limitations des ordinateurs classiques. Grâce à leur traitement parallèle de l'information et à leurs mécanismes inspirés des cellules nerveuses (Neurones), ils infèrent des propriétés émergentes permettant de solutionner des complexes. [27]

II.2 Historique

Les réseaux de neurones sont nés il y a plus de soixante ans. Les premiers à proposer un modèle sont deux bio-physiciens de Chicago, McCulloch et Pitts [28], qui inventèrent en 1943 le premier neurone formel qui est un dispositif à plusieurs entrées et une sortie et qui modélise certaines propriétés du neurone biologique. Il peut être dans un état soit inhibé soit excité, et changer au cours du processus. Le neurone formel reste l'élément de base des réseaux de neurones actuels.

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 multicouches.

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 multi couches ne possède pas les défauts mis en évidence par Minsky. Proposé pour la première fois par Werbos, le Perceptron multicouches apparaît en 1986 introduit par Rumelhart, et, simultanément, sous une appellation voisine, chez Le Cun (1985). Ces systèmes reposent sur la rétro propagation

du gradient de l'erreur dans des systèmes à plusieurs couches, chacune de type Adaline de Bernard Widrow, proche du Perceptron de Rumelhart.

De nos jours, l'utilisation des réseaux de neurones dans divers domaines ne cesse de croître. Les applications sont multiples et variées.

II.3 Le neurone biologique

Le neurone biologique est une cellule vivante spécialisée dans le traitement des signaux électriques. 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. Ces axones conduisent les signaux électriques de la sortie d'un neurone vers l'entrée (synapse) d'un autre neurone.

Les neurones font une sommation des signaux reçus en entrée et en fonction du résultat obtenu vont fournir un courant en sortie. (Figure II.1)

Les neurones sont des cellules nerveuses décomposables en 4 parties principales:

- les dendrites : sur lesquelles les autres cellules entrent en contact synaptique : c'est par les dendrites que se fait la réception des signaux.
- Le noyau : Le corps de la cellule, c'est l'unité de traitement.
- L'axone : où passent les messages accumulés dans le corps de la cellule : l'envoi de l'information se fait par l'axone.
- Les synapses par lesquelles la cellule communique avec d'autres cellules, ce sont des points de connexion par où passent les signaux de la cellule.

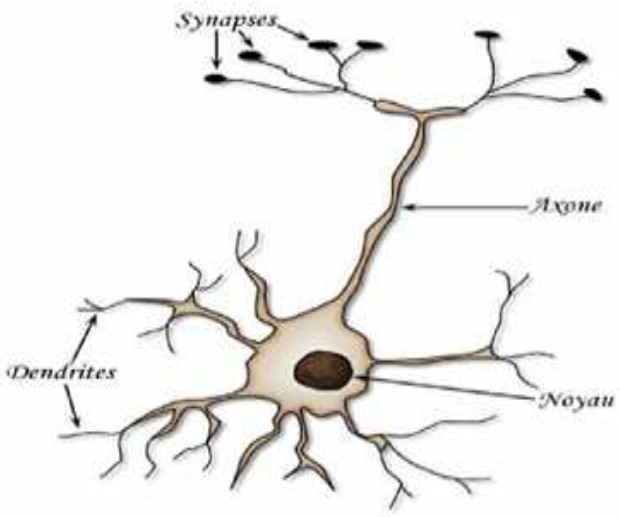


Figure II.1: *Le neurone biologique.*

II.4 Le neurone formel

Un neurone formel est une fonction algébrique paramétrée, à valeurs bornées, de variable réelles appelées entrées.

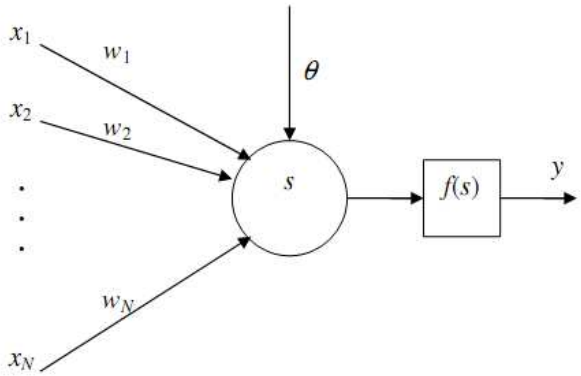


Figure II.2: *Le neurone formel.*

En règle générale, le calcul de la valeur de cette fonction peut se décomposer en deux étapes :

Une combinaison linéaire des entrées :

$$s = \theta + \sum_{i=1}^{n} w_i . x_i \qquad (2.1)$$

Les w_i sont appelés poids synaptiques ou simplement poids, θ est appelé biais. Le biais peut être considéré comme la pondération de l'entrée 0 fixée à 1. s est appelé potentiel du neurone.

La sortie du neurone est :

$$y = f(s) = f(\theta + \sum_{i=0}^{n} w_i . x_i) \qquad (2.2)$$

La fonction f est la fonction d'activation du neurone. Il existe plusieurs fonctions d'activation [28].Elle permet de définir l'état interne du neurone en fonction de son entrée totale, citons à titre d'exemple quelques fonctions souvent utilisées :

Nom de la fonction	Relation d'entrée/sortie	Icône	Nom Matlab
seuil	$a = 0 \quad \text{si } n < 0$ $a = 1 \quad \text{si } n \geq 0$		hardlim
seuil symétrique	$a = -1 \quad \text{si } n < 0$ $a = 1 \quad \text{si } n \geq 0$		hardlims
linéaire	$a = n$		purelin
linéaire saturée	$a = 0 \quad \text{si } n < 0$ $a = n \quad \text{si } 0 \leq n \leq 1$ $a = 1 \quad \text{si } n > 1$		satlin
linéaire saturée symétrique	$a = -1 \quad \text{si } n < -1$ $a = n \quad \text{si } -1 \leq n \leq 1$ $a = 1 \quad \text{si } n > 1$		satlins
linéaire positive	$a = 0 \quad \text{si } n < 0$ $a = n \quad \text{si } n \geq 0$		poslin
sigmoïde	$a = \frac{1}{1+\exp^{-n}}$		logsig
tangente hyperbolique	$a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$		tansig
compétitive	$a = 1 \quad \text{si } n \text{ maximum}$ $a = 0 \quad \text{autrement}$		compet

Tableau II.1 : *Fonction d'activation.*

II.5 Les réseaux de neurones formels

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

II.5. 1 Les réseaux de neurones non bouclés [29]

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 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.

La Figure II.3 représente un réseau de neurones non bouclé qui a 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 multicouches.

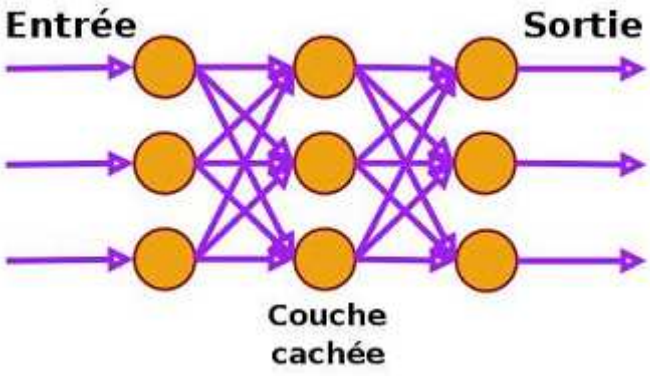


Figure II.3: *Perceptron multicouches.*

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 linéaires.

II.5.2 Les réseaux de neurones bouclés

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.

Un réseau de neurones bouclé à temps discret est donc régi par une (ou plusieurs) équations aux différences non linéaires, résultant de la composition des fonctions réalisées par chacun des neurones et des retards associés à chacune des connexions.

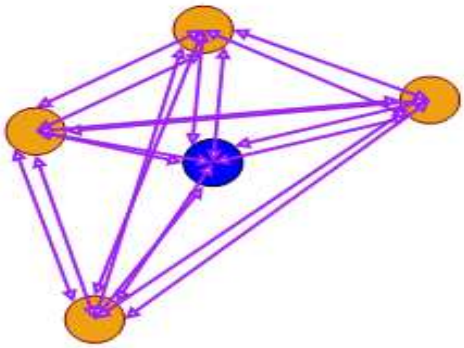


Figure II.4: *Réseau de neurone « Hopfield ».*

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

II.6 Domaine d'applications des réseaux de neurones

- Aérospatial : pilotage automatique, simulation du vol…
- Automobile : système de guidage automatique,…
- Défense : guidage de missile, suivi de cible, reconnaissance du visage, radar, traitement du signal, compression de données, suppression du bruit…
- Electronique : prédiction de la séquence d'un code, vision machine, synthétiseur vocal, modèle non linéaire,…
- Finance : Prévision du coût de la vie
- Secteur médical : Analyse EEC et ECG
- Télécommunications : Compression de données …

II.7 Apprentissage des réseaux de neurones

L'apprentissage est vraisemblablement la propriété la plus intéressante des réseaux de neurones. Elle ne concerne cependant pas tous les modèles, mais les plus utilisés.

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. L'apprentissage des réseaux de neurones fait appel à des exemples de comportement.

Durant cette phase de fonctionnement, le réseau adapte sa structure (le plus souvent, les poids des connexions) afin de fournir sur ses neurones de sortie les valeurs désirées. Cet apprentissage nécessite des exemples désignés aussi sous l'appellation d'échantillon d'apprentissage ainsi qu'un algorithme d'apprentissage. Après initialisation des poids du

réseau (en général des valeurs aléatoires), il y a présentation des exemples au réseau et calcul des sorties correspondantes. Une valeur d'erreur ou de correction est calculée et une correction des poids est appliquée.

Au niveau des algorithmes d'apprentissage, il a été défini deux grandes classes selon que l'apprentissage est dit supervisé ou non supervisé. Cette distinction repose sur la forme des exemples d'apprentissages. Dans le cas de l'apprentissage supervisé, les exemples sont des couples (Entrée, Sortie associée) alors que l'on ne dispose que des valeurs (Entrée) pour l'apprentissage non supervisé.

II.7.1 Apprentissage supervisé [30]

Un superviseur, ou professeur, fournit au réseau des couples d'entrées-sorties. Il fait apprendre au réseau l'ensemble de ces couples, par une méthode d'apprentissage, comme la rétro propagation du gradient de l'erreur, en comparant pour chacun d'entre eux la sortie effective du réseau et la sortie désirée. L'apprentissage est terminé lorsque tous les couples entrée-sortie sont reconnus par le réseau. Ce type d'apprentissage se retrouve, entre autres, dans le perceptron.

II.7.2 Apprentissage non supervisé [31]

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

II.7.3 Apprentissage hybride

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

II.8 Les méthodes d'apprentissage

Dans les systèmes experts, les connaissances de l'expert ont une forme énumérée. Elles sont exprimées sous forme de règles. Dans le cas des réseaux de neurones, les connaissances ont une forme distribuée. Elles sont codées dans les poids des connexions, la topologie du réseau, les fonctions de transfert de chaque neurone, le seuil de ces fonctions, la méthode d'apprentissage utilisée. Il existe un certain nombre de méthodes d'apprentissage:

- La règle de Hebb: c'est la méthode d'apprentissage la plus ancienne (1949), elle est inspirée de la biologie. Elle traduit le renforcement des connexions liant deux neurones activés. Si un des deux neurones au moins n'est pas activé, le poids de la connexion n'est pas modifié [32].
- La rétro propagation du gradient de l'erreur: cet algorithme est utilisé dans les réseaux de type feed forward, ce sont des réseaux de neurones à couches, ayant une couche d'entrée, une couche de sortie, et au moins une couche cachée. Il n'y a pas de récursivité dans les connexions, et pas de connexions entre neurones de la même couche. Le principe de la rétro propagation consiste à présenter au réseau un vecteur d'entrées, de procéder au calcul de la sortie par propagation à travers les couches, de la couche d'entrée vers la couche de sortie en passant par les couches cachées. Cette sortie obtenue est comparée à la sortie désirée, une erreur est alors obtenue. A partir de cette erreur, est calculé le gradient de l'erreur qui est à son tour propagé de la couche de sortie vers la couche d'entrée, d'où le terme de rétro propagation. Cela permet la modification des poids du réseau et donc l'apprentissage. L'opération est réitérée pour chaque vecteur d'entrée et cela jusqu'à ce que le critère d'arrêt soit vérifié [33].
- Les algorithmes génétiques: ils représentent une modélisation de la sélection naturelle. Une population d'individus est générée aléatoirement. Un certain nombre d'individus répondant le mieux aux critères de sélection est choisi. A partir de cette population d'élites, une nouvelle population est générée par reproduction, mutation, ou crossover sur les individus de départ, ou parents. L'opération est recommencée jusqu'à la vérification du critère d'arrêt [34].
- Ou tout autre algorithme d'optimisation : recuit simulé, algorithme de Sollis & Wets…

II.9 Propriété fondamentale des réseaux de neurones formels

L'approximation parcimonieuse : Les réseaux de neurones formels, possèdent une propriété remarquable qui est à l'origine de leur intérêt pratique dans des domaines très divers: ce sont des approximateurs universels parcimonieux.

Sans entrer dans les détails mathématiques, la propriété d'approximation peut être énoncée de la manière suivante: toute fonction bornée suffisamment régulière peut être approchée avec une précision arbitraire, dans un domaine fini de l'espace de ses variables, par un réseau de neurones comportant une couche de neurones cachés en nombre fini, possédant tous la même fonction d'activation, et un neurone de sortie linéaire [35].

Cette propriété n'est pas spécifique aux réseaux de neurones. Il existe bien d'autres familles de fonctions paramétrées possédant cette propriété; c'est le cas notamment des ondelettes, des fonctions radiales, des fonctions splines, par exemple [36].

II.10 Quelques modèles des réseaux de neurones

II.10.1 Le perceptron multicouches

Le réseau MLP (Multi Layer Perceptron) est un modèle de réseau de neurones comprenant une ou plusieurs couches cachées, dont les connexions sont directes et totales entre les couches. Werbos, pour sa thèse de doctorat en 1982, développa la théorie de la rétro propagation du gradient de l'erreur. Cet algorithme fut popularisé en 1986 par Rumelhart qui l'utilisa pour la modification des poids et des biais dans les réseaux de neurones multicouches.

La figure ci-dessous donne un exemple d'un MLP contenant une couche d'entrée, deux couches cachées et une couche de sortie. La couche d'entrée présente toujours une couche virtuelle associée aux entrées du système (X), Les couches suivantes sont des couches de Neurones.

La première couche est reliée aux entrées, ensuite chaque couche est reliée à la couche précédente. C'est la dernière couche qui produit les sorties (S) du MLP. Les sorties des autres couches ne sont pas visibles à l'extérieur du réseau, et elles sont appelées pour cette raison couches cachées [37].

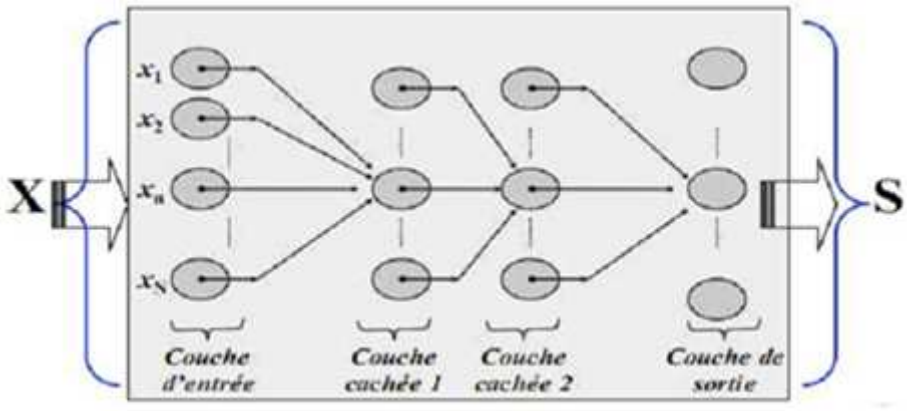


Figure II.5: *Perceptrons multicouches.*

II.10.1.1 Algorithme du MLP

MLP utilise la règle de rétro propagation du gradient (Back-propagation), Il a fallu attendre le début des années 1980 pour qu'une règle efficace soit mise au point pour l'apprentissage des réseaux multicouches. Cette règle, découverte simultanément par des équipes françaises et américaines, est en fait une généralisation de la règle du delta. Elle consiste simplement en une rétro propagation du gradient, qui est une méthode d'optimisation universelle.

L'apprentissage d'un perceptron multicouche se fait avec une rétro propagation, l'algorithme décrit en Figure II.6. En définissant une erreur sur chaque couche du perceptron.

L'erreur de la dernière couche est effectivement l'erreur du réseau, et pour chaque couche cachée les erreurs sont calculées à partir des neurones de la couche suivante. Pour chaque neurone l'erreur est la somme des erreurs de chaque neurone de la couche suivante, pondérée par le poids qui le lie au neurone dont on calcule l'erreur et par la dérivée de la fonction d'activation. Pour cette raison cet algorithme est appelé rétro propagation de l'erreur [38].

L'apprentissage d'un MLP est défini comme un problème d'optimisation qui consiste à trouver les coefficients du réseau en minimisant une fonction d'erreur globale, la définition de cette fonction est primordiale, car celle-ci sert à mesurer l'écart entre les sorties désirées du modèle et les sorties du réseau obtenus. Dont la définition est :

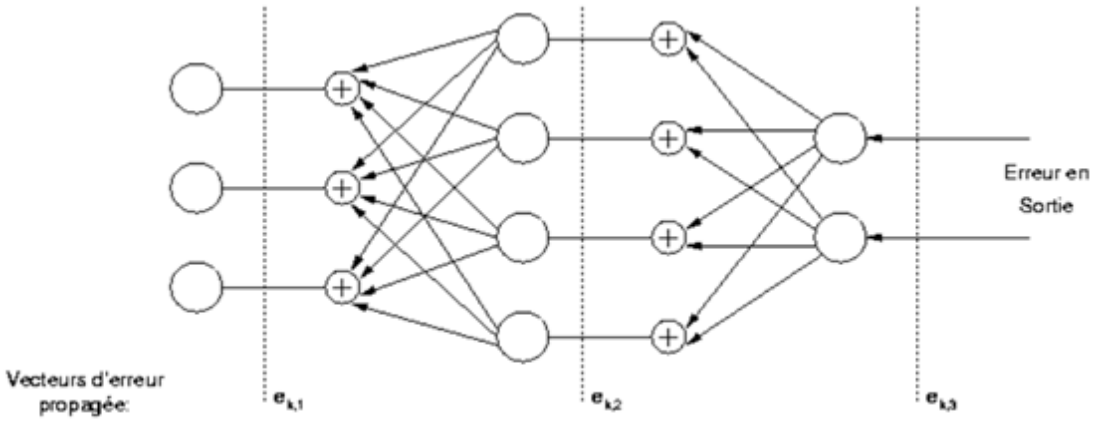


Figure II.6: *Principe de la rétro propagation.*

Pour chaque exemple n (nϵN) calculer une fonction d'erreur quadratique selon l'équation suivante: [39]

$$e(n) = \frac{1}{2} \sum_{1 \le j \le i} \left| d_j(n) - Y_j(n) \right|^2 \qquad (2.3)$$

Où :

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

Pour tout l'ensemble d'apprentissage **N**, la fonction d'erreur quadratique moyenne(**EQM**).

$$E(n) = \frac{1}{N} \sum_{n=1}^{N} e(n) \qquad (2.4)$$

Il a été démontré qu'un réseau MLP (Multi Layer perceptron) avec seulement une couche cachée dont la fonction d'activation est de type sigmoïde et une sortie avec une fonction d'activation linéaire peut approximer n'importe quelle fonction continue pourvu qu'un nombre suffisant de neurones soit fourni [40,41]. Cette propriété est dite propriété d'approximation universelle.

Algorithme de l'apprentissage du MLP

1-Début : Initialisation des poids du MLP
2-Faire :

- **Propagation Directe :** Pour chaque vecteur de la base d'apprentissage :

  2-1 Affecter $X_k = X_n$ (n=1….N) ; envoyer aux neurones cachés.

  2-2 Chaque neurone caché calcule son entrée nette :

  $$Z_in_j = V_{j0} + \sum_{1 \le n \le N} T_n V_{jn}$$

  2-3 Appliquer sa fonction de sortie $Z_j = F(Z_in_j)$
  2-4 Chaque neurone de sortie calcule son entrée

  $$Y_in_m = W_{j0} + \sum_{1 \le j \le J} Z_j W_{mj}$$

  2-5 Appliquer sa fonction de sortie

  $$Y_m = F(Y_in_m)$$

- **Rétro propagation** : Chaque neurone de sortie reçoit son étiquette t_m

  2-6 Calculer les signaux d'erreur de sortie

  $$\delta_m = (t_m - Y_m)\ F'(Y_in_m)$$

  2-7 Calculer la modification des poids-couche de sortie

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

  2-8 Rétro propager les signaux d'erreurs δ_m vers la couche cachée précédente.
  2-9 Chaque neurone caché calcule sa contribution à partir de la rétro Propagation des signaux d'erreurs de sortie.

  $$\delta_in_j = \sum_{1 \le m \le M} \delta_m W_{mj}$$

  2-10 Chaque neurone caché calcule son signal d'erreur

  $$\delta_j = \delta_in_j\ F'(Z_in_j)$$

  2-11 calculer les modifications de poids-couche caché

  $$\Delta V_{jn} = \eta\ X_n\ \delta j$$

2-12 Mises à jour des poids(W) et biais (V)

V_{jn} (nouveau) ← ΔV_{jn} + V_{jn} (ancien)

W_{mj} (nouveau) ← ΔW_{mj} + W_{mj} (ancien)

Jusqu'à critère d'arrêt satisfait (fixation du nombre d'itération)

3-Retourner les poids du réseau MLP.

4-Fin

II.10.2 Les réseaux de neurones récurrents

Le perceptron ne possède pas de liaison récurrente, il représente la première application de réseau de neurones de McCulloch et Pitts. Les limites du modèle du perceptron ont amené les chercheurs à proposer d'autres modèles avec des topologies différentes. Dans les années 80, Hopfield propose un modèle inspiré de la physique, les verres de Spin /MEZART & al. 91/, et relance la recherche neuronale.

II.10.2.1 Architecture de HOPFIELD

Les réseaux de Hopfield sont des réseaux de neurones totalement interconnectés, récursifs. Il n'y a plus de notions de couche comme dans le perceptron. Les réseaux de neurones récurrents et auto-récurrents permettent d'obtenir des résultats intéressants comparativement aux réseaux à couche [42].

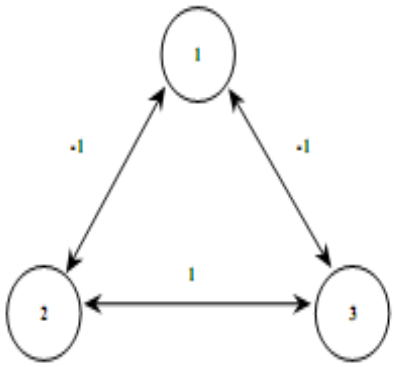


Figure II.7: *Exemple de réseau récurrent symétrique.*

La fonction de transfert utilisée est une fonction binaire à seuil nul telle que :

$$\begin{cases} \forall x \in \mathbb{R}^{+}, f(x) = 1 \\ \forall x \in \mathbb{R}^{*-}, f(x) = 0 \end{cases} \quad (2.5)$$

II.10.2.2 Réseaux d'Elman

Le réseau d'Elman est généralement un réseau de deux-couche avec la rétroaction de la première couche de sortie à la première couche d'entrée.

Cette connexion récurrent permet au réseau d'Elman à détecte et produit des modèles variables dans le temps. Un réseau d'Elman de deux couches est montré ci-dessous.

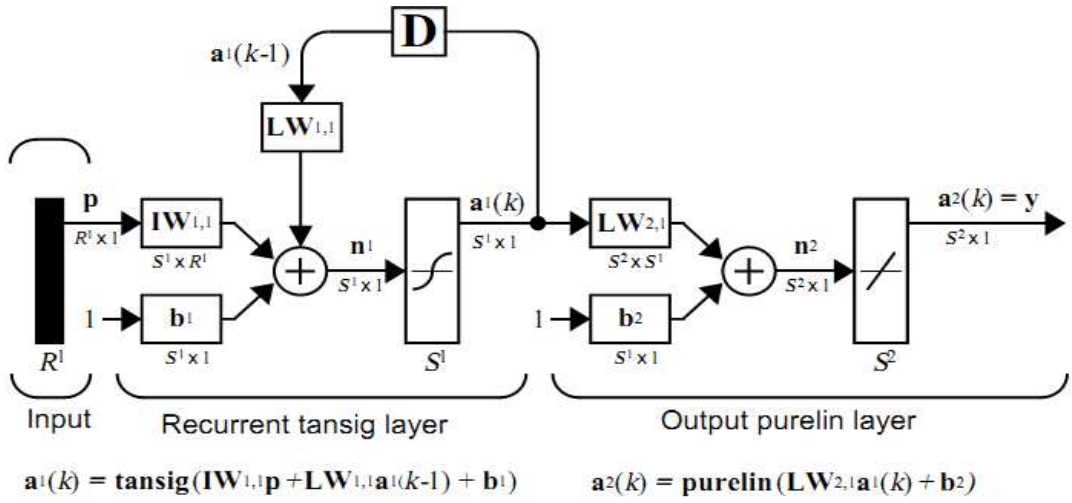


Figure II.8: *Le réseau Elman.*

II.10.3 Le réseau à fonction de base radiale

Le réseau à fonction de base radiale **RBF** (Radial Basis Function) est basé sur une architecture qui s'organise en deux couches seulement ; une couche cachée et une couche de sortie [40].

Comme le montre la figure II.9, la couche cachée, constituée des noyaux (ou neurones) RBF effectue une transformation non linéaire de l'espace d'entrée. La couche de sortie calcule une combinaison linéaire des sorties de la couche cachée. Chaque noyau élémentaire calcule la distance entre l'entrée et son centre qu'il passe ensuite dans un non linéarité concrétisée par une fonction d'activation (ϕ) qui est généralement de type gaussienne [40]. La valeur que prend la sortie du noyau gaussien est d'autant plus importante que l'entrée est plus proche de son centre et tend vers zéro, lorsque la distance entrée centre devient importante. La sortie du réseau RBF est donnée par :

$$y_i = \sum_{k=1}^{N_1} w_{kj} \phi_k (\|x - c_k\|) \qquad (2.6)$$

$$\phi(\xi) = \exp(-\frac{\xi^2}{e\eta^2}) \qquad (2.7)$$

$\|.\|$ dénote la norme euclidienne, $\boldsymbol{x}$ le vecteur d'entrée, $\boldsymbol{c_k}$ est le centre associé au noyau $\boldsymbol{k}$. $\boldsymbol{N_1}$ le nombre de noyaux de la couche cachée et $\boldsymbol{w_{kj}}$ les poids associés à la couche de sortie. Le paramètre '$\boldsymbol{\eta}$' permet de contrôler la vitesse de décroissance de la fonction ϕ.

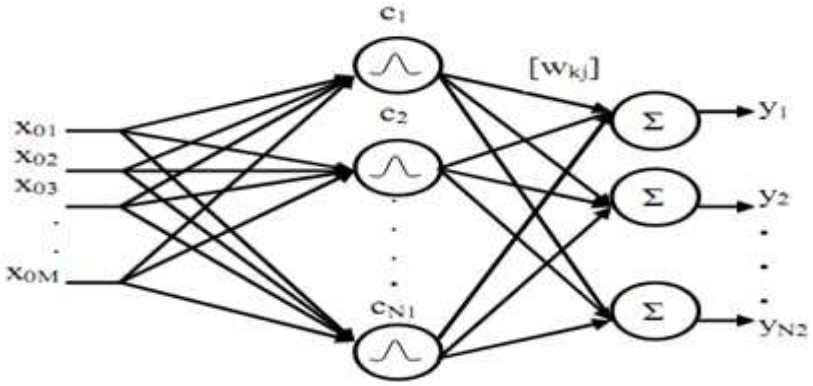


Figure II.9: *Architecture du réseau RBF.*

L'utilisation habituelle des RBF conserve une fonction d'activation linéaire en sortie mais l'utilisation d'une fonction non linéaire reste possible.

II.10.3.1 Les réseaux de neurones à régression généralisée (Generalized Regression Networks)

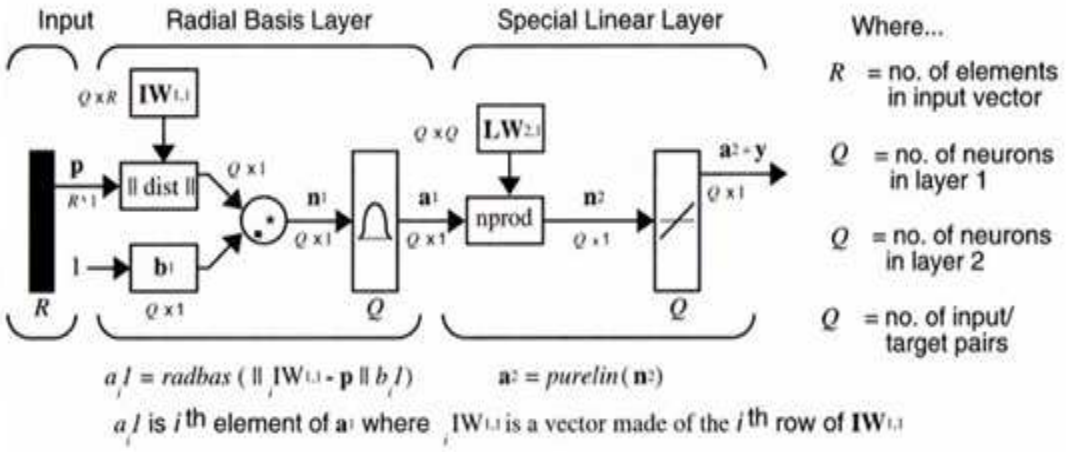


Figure II.10: *Architecture du réseau de neurones à régression généralisée.*

Dans un réseau à régression généralisée (GRNN), il y a un réseau à base radiale auquel on ajoute une couche de sortie constituée d'une fonction de transfert linéaire. nprod signifie une multiplication élément par élément, moralisé par la somme des éléments de a.

Ces réseaux sont aussi utilisés en tant qu'approximation de fonction, mais sont plus lourds d'utilisation que les perceptrons multicouches.

II.10.3.2Les réseaux de neurones probabilistes (Probabilistic Neural Networks)

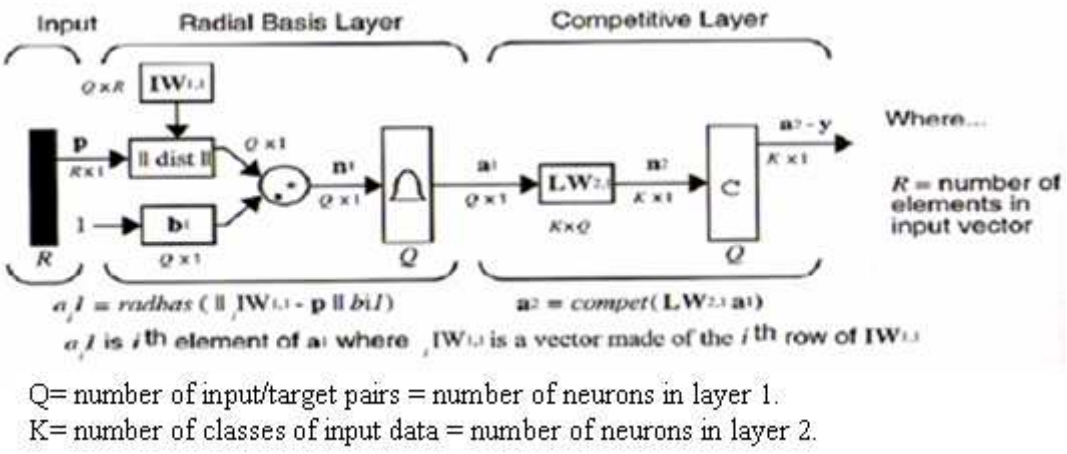


Figure II.11: *Architecture du réseau de neurones probabiliste.*

Ces réseaux sont généralement utilisés pour des problèmes de classification. La première couche qui est un réseau à base radiale, donne une information sur la ressemblance entre la donnée d'entrée et le jeu de données utilisé lors de l'apprentissage. La deuxième couche produit comme sortie un vecteur de probabilité. Finalement, une fonction de transfert compétitive produit 1 ou 0.

II.11 Conclusion

L'avantage fondamental des réseaux de neurones par rapport aux modèles statistiques traditionnels réside dans le fait qu'ils permettent d'automatiser la découverte des dépendances les plus importantes du point de vue de la prédiction du processus.

Les performances du réseau de neurones sont très souvent liées à son architecture. Or, le choix de l'architecture est aussi un problème difficile à résoudre. On ne peut pas choisir au hasard le nombre de neurones de la couche d'entrée ou de la couche intermédiaire. Seule l'expérience permet de répondre à cette question car il n'y a pas de théorie dans ce domaine.

Les réseaux de neurones à apprentissage supervisé présentent une forme de calibrage qui consiste à minimiser l'erreur entre un jeu de valeurs expérimentales et le jeu de

valeurs de sortie du réseau de neurones associé, calculé algébriquement en fonction de ses paramètres.

En fait, l'apprentissage non supervisé qui a pour objectif la détection de régularités dans des ensembles de données , permettant de simplifier ces derniers et de n'en retenir que les éléments les plus caractéristique .la classification non supervisée ou clustering plus précisément pour objectif la décomposition de la base de données en sous-groupes ,ou cluster ,homogènes et distincts .

Chapitre III

Solution des Equations Différentielles par les réseaux de neurones

III.1 Introduction

Les méthodes numériques de prédicteur-correcteur, Runge-Kutta, différences finies, éléments finis, volumes finis, et la collocation constitue une stratégie qui nous permet d'attaquer plusieurs problèmes en mathématiques appliquées, où nous simulons un problème du monde réel avec une équation différentielle, sous réserve de certaines conditions initiales ou aux limites. Dans la différence finie et la méthode d'éléments finis, nous rapprochons de la solution en utilisant les opérateurs numériques des dérivées de la fonction et de trouver la solution à des grilles spécifiques pré attribués [43]. Les méthodes de Prédicteur-correcteur et Runge-Kutta sont largement appliqués au cours des points de grille pré attribués pour résoudre des équations différentielles ordinaires [44]. Dans les approches spectrales et de collocation, une série tronquée des fonctions orthogonales spécifiques sont employées pour la solution rapprochée de l'équation différentielle. Dans les techniques spectrales et la collocation le rôle des fonctions d'approximation comme une fonction de base est important. Les fonctions d'approximation qui sont utilisées dans les méthodes spectrales sont choisies parmi les différentes classes de polynômes de Jacobi [45], toujours les mailles de discrétisation sont prés. En 1990 Lee et Kang [46] ont utilisé des ordinateurs avec des processeurs parallèles pour résoudre une équation différentielle du premier ordre avec des modèles de réseau de neurone de Hopfield. Meade et Fernandez [47,48] ont résolu des équations différentielles ordinaires linéaires et non linéaires en utilisant des réseaux de neurone de type MLP et les B_1-splines. Il n'est pas facile d'étendre ces techniques à des domaines multidimensionnels [49]. Le but de ce chapitre est de présenter une méthode basée sur les réseaux de neurones et les techniques d'optimisation, qui permet de résoudre des équations différentielles d'ordre élevé en cherchant la forme analytique fermé de la fonction spécifique [51,52] comme une solution approchée. Cette forme analytique est une combinaison de deux termes. Un premier terme est responsable aux conditions initiales ou aux limites et le second terme contient les paramètres réglables de réseau de neurone pour être calculé.

III.2 Etat de l'art

La plupart des travaux précédents pour résoudre les problèmes de valeurs initiales et aux limites en utilisant les réseaux de neurones sont basés sur la substitution de la solution approchée dans les équations différentielles correspondantes. En 1994, Dissanayake et Phan-Thien [49] résolu l'equation linéaire de poisson et l'équation non linéaire de chaleur

pour quatre chiffres décimaux de précision. La solution numérique de problèmes elliptiques de second ordre jusqu'à six chiffres décimaux de précision ont été donnés par Van Milligen et al [70]. En 1998, Lagaris et al [51] a introduit un réseau de neurones artificiels pour la résolution des équations non linéaires du seconde ordre avec des valeurs initiales et aux limites (Dirichlet et Neumann) avec sept chiffres décimaux de précision. L'équation de Hamilton-Jacobi-Bellman, qui est une équation non linéaire de premier ordre, a été résolue par un réseau de neurone à trois couches [71], tandis que le problème d'eau souterraine de second ordre (dépendant du temps) à une dimension a été résolu par les réseaux de neurones et les algorithmes génétiques [72]. Dans [73], deux équations différentielles non linéaires avec deux points de Dirichlet et des conditions de Neumann ont été résolus avec la valeur moyenne des erreurs absolues d'ordre 10^{-2}. En outre, dans cette référence, l'équation de Laplace avec des conditions aux limites sur un domaine de forme irrégulière est résolue avec des erreurs résiduelles d'ordre 10^{-2}. La Référence [74] contient une méthode pour simuler une solution non stationnaire d'un lit fixe non catalytique du réacteur gaz solide par réseaux de neurones artificiels, avec l'erreur quadratique moyenne d'ordre 10^{-9}.

Alli et al. [75] En 2003, résolu une équation d'onde à deux dimensions et le problème de vibration d'une poutre latérale avec les deux extrémités fixes jusqu'à cinq chiffres décimaux de précision dans un processus neuronal. En 2003, un problème de chaleur contrôlée avec précision de trois chiffres décimaux été résolu par un réseau de neurones à trois couches [76]. Les solutions pour Kuramoto-Sivashinsky et de Navier-Stokes en utilisant les réseaux de neurones sont concernés par la Référence. [77].La Rférence [78] contient la modélisation du comportement dynamique de l'équation de Kuramoto-Sivashinsky à l'aide de réseaux de neurone.

III.3 Formulation du problème

Dans cette section, nous nous préoccupons de la valeur de condition initiale et au limite des problèmes généraux de la forme:

$$\begin{cases} D\left[x, y(x), \dfrac{dy}{dx}, \dfrac{d^2y}{dx^2}, \ldots, \dfrac{d^ny}{dx^n}\right] = 0 & \text{x [a , b]} \\ C\left[x, y(x), \dfrac{dy}{dx}, \dfrac{d^2y}{dx^2}, \ldots, \dfrac{d^ny}{dx^n}\right] = 0 & \text{x=a et/ou b} \end{cases} \qquad (3.1)$$

Où

D : est un opérateur différentiel de degré n

C : est un opérateur des conditions initiales ou aux limites.

X : est une variable indépendante appartient à [a, b]

Y : est une variable inconnue dépendante qui doit être calculé.

Remarque :

Dans le cas de grandes valeurs de l'entier n, on parle du problème d'une équation différentielle d'ordre élevé.

Supposons que la solution générale rapproché pour le problème (3.1) est sous la forme $y_T(x,P)$ où y_T est une variable dépendante de x et P, et P est un paramètre ajustable impliquant des poids et des biais dans la structure du réseau de neurone à trois couches de type MLP. La solution y_T est une solution d'approximation à y pour les valeurs optimisées des poids et des biais inconnus. Ainsi le problème de trouver la solution rapprochée pour le problème (3.1) sur certains points de collocation dans l'intervalle [a,b] est équivalant à calculer $y_T(x,P)$ pour satisfait le problème d'optimisation suivant [53]:

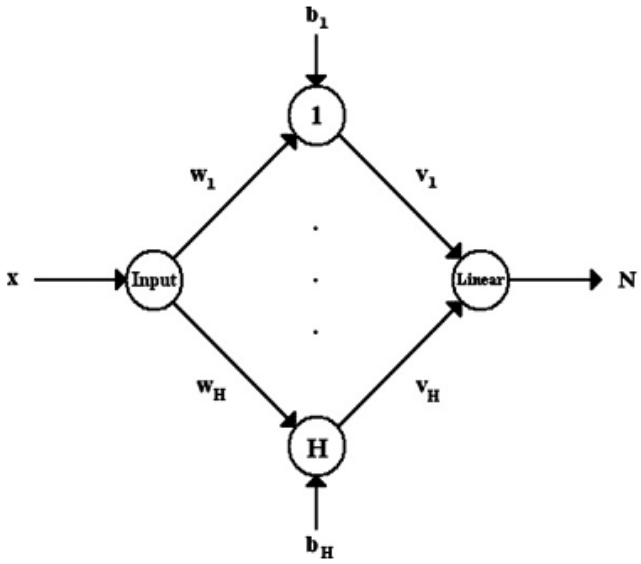


Figure III.1 : *Perceptron multicouches avec une entrée, une couche cachée et une sortie.*

$$\begin{cases} Min \int_a^b \left\| D\left[x, y_T(x,P), \frac{dy_T(x,P)}{dx}, \frac{d^2 y_T(x,P)}{dx^2}, ..., \frac{d^n y_T(x,P)}{dx^n} \right] \right\|^2 dx \\ C\left[x, y_T(x,P), \frac{dy_T(x,P)}{dx}, \frac{d^2 y_T(x,P)}{dx^2}, ..., \frac{d^n y_T(x,P)}{dx^n} \right] = 0, \quad \text{x= a et/ou x=b} \end{cases} \tag{3.2}$$

Où nous utilisons $\|.\|$ pour la norme euclidienne. Cela signifie que nous sommes arrivés à une formulation alternative de minimisation continue. Parce que sur des intégrales clôtures association avec le sigma, et parce que nous sommes à la recherche de la solution $y_T(x, P)$ parmi toutes les fonctions qui appartiennent aux fonctions différentiables continues sur l'intervalle [a, b]. Cette solution $y_T(x, P)$ s'annule en $x = a$ et / ou $x = b$.

Le problème se résume dans la fonction suivante :

$$\begin{cases} Min \sum_{i=1}^{m} \left\{ D\left[x_i, y_T(x_i,P), \frac{dy_T(x_i,P)}{dx}, \frac{d^2 y_T(x_i,P)}{dx^2}, ..., \frac{d^n y_T(x_i,P)}{dx^n} \right] \right\}^2 \\ C\left[x, y_T(x,P), \frac{dy_T(x,P)}{dx}, \frac{d^2 y_T(x,P)}{dx^2}, ..., \frac{d^n y_T(x,P)}{dx^n} \right] = 0, \quad x = a \ et/ou \ \ b \end{cases} \quad (3.3)$$

Où $\{xi\}_{i=1}^{m}$ sont quelques points de collocation arbitraire sur l'intervalle [a, b] pour une norme discrets de moindres carrés. Ceci est un problème de minimisation d'un maillage discret sous contrainte pour le problème (3.1).

Afin de transférer le problème (3.3) en un problème d'optimisation sans contrainte qui est plus simple à traiter, nous définissons la solution d'approximation sous la forme suivante:

$$y_T(x,P) = \alpha(x) + \beta\left[x, N(x,P)\right] \quad (3.4)$$

Où le premier terme du côté droit n'implique pas des paramètres ajustables et satisfait les conditions initiales et aux limites. Le second terme du côté droit est un réseau de neurones de type MLP à trois couches composé d'une entrée X et d'une sortie $N(x, P)$.

Dans la prochaine section, ce modèle de réseau de neurone avec quelques poids et des biais est considéré et formés afin de calculer la solution du problème (3.1).

La Minimisation en (3.3) peut être considérée comme un processus d'apprentissage pour le réseau de neurones proposé (Figure. III.1) [54]. L'erreur correspondante à chaque entrée x est $E(x)$ doit être minimisé. Ce résidu est la conséquence directe de la substitution de la fonction d'approximation $y_T(x, P)$ au lieu de $y(x)$ dans le problème (3.1).

Afin de calculer le résiduel, il peut avoir besoin de calculer la sortie $N(x, p)$ avec ses dérivées par rapport à x comme une variable d'entrée [55].

Prenons un perceptron à trois couches (Figure. III.1**)** avec une entrée ***x***, une couche cachée constituée de ***H*** fonctions sigmoïde et une unité de sortie ***N (x, P)***. Pour toute entrée ***x***, la sortie du perceptron est $N=\sum_{i=1} v_i s(z_i)$ où :

$z_i = w_i x + b_i$

w_i : est un paramètre de poids de la couche d'entrée à la couche cachée.

v_i **:** est un paramètre de poids de la couche cachée à la couche de sortie.

b_i **:** est un biais pour l'unité i dans la couche cachée.

z_i **:** est une sortie pour l'unité i dans la couche cachée.

s **:** est une fonction sigmoïde arbitraire.

Il est facile d'exprimer les k^{éme} dérivées de ***N(x, P)*** en termes des dérivés de la fonction sigmoïde pour k=1, 2,….n

$$\frac{d^k N}{dx^k} = \sum_{i=1}^{H} v_i w_i^k s^{(k)}(z_i) \qquad (3.5)$$

Nous pourrions faire l'usage de l'induction pour prouver l'expression (3.5). Pour $k = 1$, nous avons :

$$\frac{dN}{dx} = \sum_{i=1}^{H} v_i \frac{ds(z_i)}{dx} = \sum_{i=1}^{H} v_i \frac{ds(z_i)}{dz_i}\frac{dz_i}{dx} = \sum_{i=1}^{H} v_i w_i s'(z_i) \qquad (3.6)$$

Si la fonction (3.5) est correct pour un entier ***k*** alors elle sera vérifiée pour l'entier ***k+1*** aussi, puisque

$$\frac{d^{k+1} N}{dx^{k+1}} = \frac{d}{dx}\frac{d^k N}{dx^k} = \sum_{i=1}^{H} v_i w_i^k \frac{ds^{(k)}(z_i)}{dx} = \sum_{i=1}^{H} v_i w_i^k \frac{ds^{(k)}(z_i)}{dz_i}\frac{dz_i}{dx} = \sum_{i=1}^{H} v_i w_i^{k+1} s^{(k+1)}(z_i) \qquad (3.7)$$

Aussi il y a plusieurs choix pour la fonction sigmoïde, ici on choisit $s(x)=1/(1+e^{-x})$ car il est possible de dériver toute les dérivées de $s(x)$ en termes de fonction sigmoïde elle-même :

$$\begin{aligned} s' &= -s^2 + s \\ s'' &= 2s^3 - 3s^2 + s \\ s''' &= -6s^4 + 12s^3 - 7s^2 + s \\ s^{(4)} &= 24s^5 - 60s^4 + 50s^3 - 15s^2 + s \end{aligned} \qquad (3.8)$$

Ainsi, il est possible d'utiliser toute technique d'optimisation pour calculer le résiduel ***E(x)*** obtenu à partir de substitution de la fonction d'approximation Y_T **(x, w, v, b)** dans le problème (3.1) comme la méthode de Nelder meade, la rétro-propagation…

III.4 Résolution du problème par les réseaux de neurones

Les idées développées dans la section précédente sont facilement applicables à des équations différentielles d'ordre arbitraire. Nous attirons l'attention aux problèmes de valeurs initiales et aux limites, par la suite on discutera des détails de la procédure générale pour la formulation des fonctions d'approximation correspondantes. Ceci est une considération importante car les problèmes du modèle mis en place exigent que la fonction d'approximation soit formulées en utilisant les valeurs initiales et aux limites de l'équation différentielle, ainsi que les paramètres réglables du réseau de neurones. La première étape consiste à définir une formulation générale pour les fonctions $\boldsymbol{\alpha}$ et $\boldsymbol{\beta}$ dans (3.4). Nous discutons des idées générales, en considérant les différents cas général. L'accent est mis sur les problèmes qui impliquent des équations différentielles d'ordre élevé, bien que nous donnions également quelques indications sur d'autres problèmes connexes.

III.4.1 Problème de valeur initiale

Considérons la valeur initiale d'une équation différentielle d'ordre un:

$$\begin{cases} \dfrac{d\,y\,(x)}{d\,x} = f\,(x\,,\,y\,) & x \in [a\,,\,b\,] \\ y\,(a\,) = A \end{cases} \qquad (3.9)$$

La fonction d'approximation relative sera sous la forme

$$y_T\,(x\,,P\,) = A + (x - a\,)\,N\,(x\,,P\,) \qquad (3.10)$$

Cette solution par l'intention satisfait la condition initiale du problème (3.9). La fonction d'erreur qui doit être minimisée est :

$$E(P) = \sum_{i=1}^{m} \left\{ \frac{dy_T(x_i, P)}{dx} - f\left[x_i, y_T(x_i, P)\right] \right\}^2 \qquad (3.11)$$

Où $\boldsymbol{\{x\}_{i=1}^{m}}$ sont des points distincts appartenant à l'intervalle [a, b]. En dérivant la fonction d'approximation $\boldsymbol{y_T\,(x,\,P)}$ en (3.10), nous obtenons :

$$\frac{dy_T(x, P)}{dx} = N(x, P) + (x - a)\frac{dN(x, P)}{dx} \qquad (3.12)$$

Ainsi l'expression de (3.5) avec la fonction sigmoïde introduite dans la section précédente sont applicables.

III.4.2 Équations différentielles du second ordre

Pour le problème de valeur à la limite concernant l'équation différentielle du second ordre, considérons le problème général suivant:

$$\begin{cases} \dfrac{d^2y(x)}{dx^2} = f(x, y, \dfrac{dy}{dx}) & x \in [a,b] \\ y(a) = A \qquad y(b) = B \end{cases} \tag{3.13}$$

La fonction d'approximation relative sera sous la forme :

$$y_T(x,P) = \frac{bA - aB}{b-a} + \frac{B-A}{b-a}x + (x-a)(x-b)N(x,P) \tag{3.14}$$

Si nous changeons les conditions aux limites ***y(a)=A*** et ***y'(a)=A'*** la fonction d'approximation sera transformé en :

$$y_T(x,P) = A - A'a + A'x + (x-a)^2 N(x,P) \tag{3.15}$$

Dans les deux cas la fonction d'erreur aura la forme suivante:

$$E(P) = \sum_{i=1}^{m} \left\{ \frac{d^2 y_T(x_i,P)}{dx^2} - f\left[x_i, y_T(x_i,P), \frac{dy_T(x_i,P)}{dx} \right] \right\}^2 \tag{3.16}$$

Pour le problème général d'ordre n de valeur initiale:

$$\begin{cases} \dfrac{d^n y(x)}{dx^n} = f(x, y, \dfrac{dy}{dx}, \ldots, \dfrac{d^{n-1}y}{dx^{n-1}}) & x \in [a,b] \\ y^{(i)}(a) = A_i & i = 0, \ldots, n-1 \end{cases} \tag{3.17}$$

La fonction d'approximation peut être écrite sous la forme :

$$y_T(x,P) = \sum_{i=0}^{n-1} u_i x^i + (x-a)^n N(x,P) \tag{3.18}$$

Où $\{u_i\}_{i=0}^{n-1}$ représentent la solution du système de triangle supérieur de n équations linéaires.Ce système de triangle supérieur est sous la forme :

$$\sum_{i=j}^{n-1} \binom{j}{i} j! a^{i-j} u_i = A_j \qquad j = 0, \ldots, n-1 \tag{3.19}$$

Par conséquent, la fonction d'erreur qu'il faut minimiser, devient comme suit :

$$E(P)=\sum_{i=1}^{m}\left\{\frac{d^n y_T(x_i,P)}{dx^n}-f\left[x_i,y_T(x_i,P),\frac{dy_T(x_i,P)}{dx},...,\frac{d^{n-1}y_T(x_i,P)}{dx^{n-1}}\right]\right\}^2 \qquad (3.20)$$

III.4.3 Système de problèmes aux valeurs initiales

L'idée de base derrière le modèle de solution peut être facilement étendue pour le système de ***k*** équations différentielles. Considérons le système suivant des équations différentielles du premier ordre de la valeur initiale:

$$\begin{cases}\dfrac{dy_i(x)}{dx}=f_i(x,y_1,...y_k) & x\in[a,b]\\ y_i(a)=A_i & i=1,.....k\end{cases} \qquad (3.21)$$

Correspondant à chaque fonction d'approximation $y_T(x,P_i)$ pour $(i = 1, ..., k)$, nous considérons un perceptron à trois couches $N_i(x, P_i)$ et nous introduisons :

$$\forall i=1,...,k: \qquad y_{T_i}(x,P_i)=A_i+(x-a)N_i(x,P_i) \qquad (3.22)$$

Ainsi la fonction d'erreur correspondante au total qui doit être minimisé sur tous les paramètres réglables de réseau de neurone sera :

$$E(P)=\sum_{i=1}^{m}\sum_{j=1}^{k}\left\{\frac{dy_{T_j}(x_i,P_j)}{dx}-f_j\left[x_i,y_{T_1}(x_i,P_1),...,y_{T_k}(x_i,P_k)\right]\right\}^2 \qquad (3.23)$$

Pour le système de ***k*** équation différentielle du second ordre :

$$\begin{cases}\dfrac{d^2y_i(x)}{dx^2}=f_i(x,y_1,...,y_k,\dfrac{dy_1}{dx},...,\dfrac{dy_k}{dx}) & x\in[a,b], i=1,...,k\\ y_i(a)=A_i \qquad y_i(b)=B_i\end{cases} \qquad (3.24)$$

La fonction d'approximation est de la forme :

$$y_{T_i}(x,P_i)=\frac{bA_i-aB_i}{b-a}+\frac{B_i-A_i}{b-a}x+(x-a)(x-b)N_i(x,P_i) \qquad \forall i=1,...,k \qquad (3.25)$$

Si nous changeons les conditions aux limites avec les conditions initiales suivantes:

$$\forall i=1,...,k: \qquad y_i(a)=A_i, \qquad y'_i(a)=A'_i$$

Alors les fonctions d'approximation peuvent être changées en :

$$\forall i = 1,...,k \qquad y_{T_i}(x, P_i) = A_i - A'_i\, a + A'_i\, x + (x-a)^2 N_i(x, P_i) \qquad (3.26)$$

La fonction d'erreur dans ces cas est sous la forme suivante:

$$E(P) = \sum_{i=1}^{m}\sum_{j=1}^{k}\left\{\frac{d^2 y_{T_j}(x_i, Pj)}{dx^2} - fj\left[x_i, y_{T_1}(x_i, P_1),..., y_{T_k}(x_i, P_k), \frac{dy_{T_1}(x_i, P_1)}{dx},..., \frac{dy_{T_k}(x_i, P_k)}{dx}\right]\right\}^2 \qquad (3.27)$$

Si nous prenons maintenant le système d'équation différentielle de valeur initiale de tout ordre n :

$$\begin{cases} \dfrac{d^n y(x)}{dx^n} = f_i(x, y_1,..., y_k, \dfrac{dy_1}{dx},..., \dfrac{dy_k}{dx},..., \dfrac{d^{n-1} y_1}{dx^{n-1}},..., \dfrac{d^{n-1} y_k}{dx^{n-1}}) \\ y_i^{(j)}(a) = A_{ij} \qquad i = 1,...,k, \qquad\qquad j = 0,...,n-1 \quad x \in [a,b] \end{cases} \qquad (3.28)$$

Alors la fonction d'approximation correspondante sera sous la forme :

$$\forall i = 1,...,k \qquad y_{T_i}(x, P_i) = \sum_{i=0}^{n-1} u_{ij} x^j + (x-a)^n N_i(x, P_i) \qquad (3.29)$$

Où $\{u_{ij}\}_{i=1}^{k}$ pour $(j = 0, ..., n-1)$ représentent la solution au système de triangle supérieur de n équations linéaires de la forme suivante :

$$\sum_{k=j}^{n-1} \binom{j}{k} j!\, a^{k-j} u_{ik} = A_{ij} \qquad i = 1,...,k \qquad j = 0,...,n-1 \qquad (3.30)$$

Dans laquelle la fonction d'erreur peut être écrite sous la forme :

$$E(P) = \sum_{i=1}^{m}\sum_{j=1}^{k}\left\{\frac{d^n y_{T_j}(x_i, P_j)}{dx^n} - f_j\left[x_i, y_{T_1}(x_i, P_1),..., y_{T_k}(x_i, P_k), \frac{dy_{T_1}(x_i, P_1)}{dx},....,\right.\right.$$

$$\left.\left. \times \frac{dy_{T_k}(x_i, P_k)}{dx},...., \frac{d^{n-1} y_{T_1}(x_i, P_1)}{dx^{n-1}},..., \frac{d^{n-1} y_{T_k}(x_i, P_k)}{dx^{n-1}}\right]\right\}^2 \qquad (3.31)$$

III.4.4 Équations différentielles du quatrième ordre

Considérons l'équation différentielle suivante du quatrième ordre:

$$\begin{cases} \dfrac{d^4y(x)}{dx^4} = f\left(x, y, \dfrac{dy}{dx}, \dfrac{d^2y}{dx^2}, \dfrac{d^3y}{dx^3}\right) & x \in [a,b] \\ y(a) = A \quad y(b) = B \quad y'(a) = A' \quad y'(b) = B' \end{cases} \tag{3.32}$$

La solution de l'équation (3.32) doit être sous la forme :

$$y_T(x,P) = Z(x) + M(x)N(x,P) \tag{3.33}$$

Maintenant, on peut se demander si, pour une condition à la limite générale donnée dans (3.32), il y a toutes les formes générales pour les fonctions ***Z(x)*** et ***M(x)*** pour construire $y_T(x,P)$. Il s'avère que cette fonction d'approximation ne peut être satisfaite dans les conditions aux limites à moins que nous ayons :

$$\begin{cases} Z(a) = A \\ Z(b) = B \\ Z'(a) = A' \\ Z'(b) = B' \end{cases} \quad \begin{cases} M(a)N(a,P) = 0 \\ M(b)N(b,P) = 0 \\ M(a)N'(a,P) + M'(a)N(a,P) = 0 \\ M(b)N'(b,P) + M'(b)N(b,P) = 0 \end{cases} \tag{3.34}$$

Évidemment, la fonction $M(x)=(x-a)^2\ (x-b)^2$ satisfait la deuxième série d'équations (3.34). Ce qui motive l'introduction de $Z(x) = a'x^4 + b'x^3 + c'x^2 + d'x$ qui est un polynôme général de degré quatre où ***a', b', c'*** et ***d'*** sont des constantes. Ainsi de la première série d'équations (3.34) nous devons avoir :

$$\begin{cases} a'a^4 + b'a^3 + c'a^2 + d'a = A \\ a'b^4 + b'b^3 + c'b^2 + d'b = B \\ 4a'a^3 + 3b'a^2 + 2c'a + d' = A' \\ 4a'b^3 + 3b'b^2 + 2c'b + d' = B' \end{cases} \tag{3.35}$$

Après la résolution de ce système de quatre équations à quatre inconnues, nous allons arriver à la forme générale de polynôme ***Z(x).*** Nous sommes maintenant en mesure d'introduire :

$$E(P) = \sum_{i=1}^{m} \left\{ \frac{d^4y_T(x_i,P)}{dx^4} - f\left[x_i, y_T(x_i,P), \frac{dy_T(x_i,P)}{dx}, \frac{d^2y_T(x_i,P)}{dx^2}, \frac{d^3y_T(x_i,P)}{dx^3}\right] \right\}^2 \tag{3.36}$$

Pour la minimisation.

III.5 Résolution du problème par les réseaux de neurones en introduisant la notion du temps

La puissance principale du calcul de réseau de neurones résulte plusieurs de neurones qui se relient et s'adaptent afin de former les réseaux. La topologie la plus simple pour ces réseaux est un groupe de neurones qui sont organisés dans une couche, appelée le réseau de couche ou le perceptron simple. L'architecture des réseaux de neurones multicouches est constituée par un groupe de réseaux d'une couche [59].

Dans les réseaux de neurones de type MLP avec plusieurs entrée et une sortie, il n'y pas de boucle de rétroaction et les sorties des neurones d'une couche forment les entrées pour la prochaine couche (Figure III.2).

En conséquence, les réseaux de neurones artificiels peuvent faire une cartographie non linéaire à partir des entrées aux sorties du système correspondant [60]. Ceci est approprié à l'analyse des systèmes décrit par des problèmes des valeurs initiales et aux limites qui n'ont pas de solutions analytiques ou leurs solutions analytiques ne sont pas facilement calculables.

Une des applications des perceptrons multicouches est la capacité d'approximation pour des valeurs réelles des fonctions à variables multiples dans une forme analytique fermée.

Notamment ces réseaux de neurones sont des approximateurs universels. Ils peuvent être formés à approximer n'importe quelle fonction mesurable définie sur un hyper cube avec n'importe quel précision désiré [61]. Les moindres conditions pour les fonctions d'activation dans les couches internes sont non-linéarité et l'intégrabilité [62]. La précision d'approximation est liée au nombre de neurones dans les couches cachées et non liée au nombre des couches cachées [63].

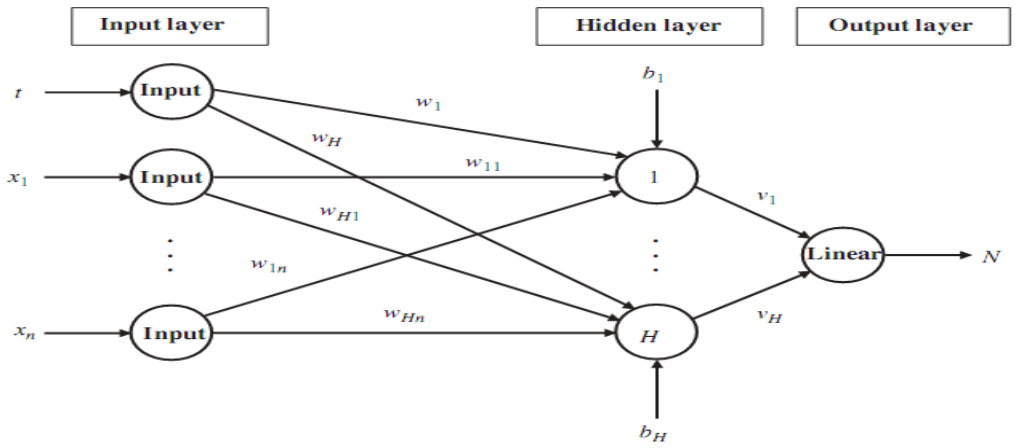


Figure III.2 : *Structure générale d'un perceptron multicouche avec une couche cachée.*

Théorème 1 : Pour toute fonction continue f : $[0\ 1]^n \rightarrow R$, il existe des fonctions g : R→R $s_i : [0,1] \rightarrow [0,1]$ *(i=0,…,2n)* et des constantes $0 \leq v_j \leq 1$ *(j=1…n)* de sorte que:

$$f(x_1,...,x_n) = \sum_{i=0}^{2n} g\left[\sum_{j=1}^{n} v_j s_i(x_j)\right] \qquad (3.37)$$

Preuve : La preuve est semblable dans l'esprit au théorème d'existence de superposition de Kolmogorov [64-65].

Cette conclure que les perceptrons à trois couches, y compris les ***n(2n +1)*** des neurones, peut calculer toute fonction continue de n variables, employant exactement les fonctions d'activation continues et strictement croissantes et non-linéaires [66].

Théorème 2 : Pour toute fonction continue f : $[0\ 1]^n \rightarrow R$, et une fonction bornée s:R→R et une donnée ε > 0,il existe H nombre naturel et des constantes réelles b_i, v_i, w_{ij} (*i=1…H, j=1…n*)de sorte que:

$$\left| f(x_1,...,x_n) - \sum_{i=1}^{H} v_i s\left[(\sum_{j=1}^{n} w_{ij} x_j) + b_i\right] \right| < \varepsilon \qquad (3.38)$$

Preuve : On peut facilement prouver ce théorème en utilisant les théorèmes de Pierre-Weierstrass et de Cybenko [67].

Dans certains cas, l'apprentissage des perceptrons multicouches est long et de mauvais tempérament. Pour éviter ces difficultés, augmentation d'information d'entrée et utilisation appropriée des techniques de minimisation d'erreur est utile [68]

III.6 Description générale de la méthode

Présentons le problème général de valeur d'initial –aux limites dépendant du temps du système des équations partielles :

$$\begin{cases} \forall i_1 = 1,\ldots, p_1 : D_{i_1}\left[t, x, \ldots, \dfrac{\partial^{\alpha_0+\alpha_1+\ldots+\alpha_n}}{\partial t^{\alpha_0}\partial x_1^{\alpha_1}\ldots\partial x_n^{\alpha_n}} y_i(t,x)\ldots \right] = 0 \quad t \in [t_0, t_{\max}], x \in \Omega \\ \forall i_2 = 1,\ldots, p_2 : I_{i_2}\left[t_0, x, \ldots, \dfrac{\partial^{\alpha_0+\alpha_1+\ldots+\alpha_n}}{\partial t^{\alpha_0}\partial x_1^{\alpha_1}\ldots\partial x_n^{\alpha_n}} y_i(t_0,x)\ldots \right] = 0 \quad x \in \Omega, \quad 1 \le i \le m \\ \forall i_3 = 1,\ldots, p_3 : B_{i_3}\left[t, x, \ldots, \dfrac{\partial^{\alpha_0+\alpha_1+\ldots+\alpha_n}}{\partial t^{\alpha_0}\partial x_1^{\alpha_1}\ldots\partial x_n^{\alpha_n}} y_i(t,x)\ldots \right] = 0 \quad t \in [t_0, t_{\max}], \quad x \in \partial\Omega \end{cases} \tag{3.39}$$

Où les fonctions multi variables à valeurs réelles D_{i1}, I_{i2}, B_{i3} sont dénotés comme un système connu dépendant du temps et non-linéaire des équations partielles avec des conditions initiales et aux limites, respectivement. ***t*** est la variable de temps, ***x*** est la variable d'espace à valeurs réelles, ***Ω*** inclus dans $\boldsymbol{R^n}$ est un domaine borné, $\boldsymbol{(\alpha_0, \alpha_1 \ldots \alpha_n) \in N_0^{n+1} (N_0 = N \cup \{0\})}$ est un multi variable d'index, $\boldsymbol{\partial^{\alpha 0+\alpha 1+\ldots+\alpha n}/\partial t^{\alpha 0}\partial x_1^{\alpha 1}\ldots\partial x_n^{\alpha n}}$ est l'opérateur différentiel partiel à valeurs réelles d'ordre $\boldsymbol{\alpha_0+\alpha_1+\ldots+\alpha_n}$ et $\boldsymbol{y(t,x)=[y_1(t,x), y_2(t,x) \ldots y_m(t,x)]}$ est la fonction vectorielle inconnue des variables ***n+1*** au-dessus du domaine cylindrique $[t_0, t_{max}] \times \Omega$ inclus dans R^{n+1} ce qui est la fonction différentiable de n'importe quel ordre désiré avec le respect à la variable qui doit être calculée.

Selon les théorèmes de Kolmogorov et de Cybenko, nous pouvons établir une solution approximative $\boldsymbol{y_T(t,x,P)=[y_{T1}(t,x,P_1), y_{T2}(t,x,P_2) \ldots y_{Tm}(t,x,P_m)]}$ qu'inclut ***m*** réseaux de neurones de type MLP et contiennent des paramètres réglables $\boldsymbol{P=(P_1, P_2 \ldots P_m)}$ (poids et biais) pour la solution $\boldsymbol{y(t,x)=[y_1(t,x), y_2(t,x) \ldots y_m(t,x)]}$.Il est possible d'obtenir des valeurs appropriées pour les paramètres réglables ***P*** tels que $\boldsymbol{y_T}$ donne la bonne approximation au ***y***[69] .

$$\min_P E(t,x,P) = \min_P \sum_{i_1=1}^{P_1} \left\{ \int_{t_0}^{t_{\max}} \int_{\Omega} \left| D_{i_1}\left[t, x, \ldots, \frac{\partial^{\alpha_0+\alpha_1+\ldots+\alpha_n}}{\partial t^{\alpha_0}\partial x_1^{\alpha_1}\ldots\partial x_n^{\alpha_n}} y_{T_i}(t,x,P_i)\ldots \right] \right|^P dx dt \right\}^{1/P}$$

$$+ \sum_{i_2=1}^{P_2} \left\{ \int_{\Omega} \left| I_{i_2}\left[t_0, x, \ldots, \frac{\partial^{\alpha_0+\alpha_1+\ldots+\alpha_n}}{\partial t^{\alpha_0}\partial x_1^{\alpha_1}\ldots\partial x_n^{\alpha_n}} y_{T_i}(t_0,x,P_i)\ldots \right] \right|^P dx \right\}^{1/P} \tag{3.40}$$

$$+ \sum_{i_3=1}^{P_3} \left\{ \int_{t_0}^{t_{\max}} \int_{\partial\Omega} \left| B_{i_3}\left[t, x, \ldots, \frac{\partial^{\alpha_0+\alpha_1+\ldots+\alpha_n}}{\partial t^{\alpha_0}\partial x_1^{\alpha_1}\ldots\partial x_n^{\alpha_n}} y_{T_i}(t,x,P_i)\ldots \right] \right|^P dx dt \right\}^{1/P} \qquad 1 \le i \le m$$

Maintenant pour la discrétisation du domaine cylindrique $\boldsymbol{[t_0, t_{max}] \times \Omega}$ par les points arbitraires $\{(t_k, x_l)\}_{(k,l)=1}^{(q,p)}$on doit employer la méthode de collocation [79] ce qui transforme le problème continu de minimisation (3.40) en problème de minimisation discret:

$$\min_P E(P) = \min_P \sum_{i_1=1}^{P_1} \left\{ \sum_{k=1}^{q} \sum_{l=1}^{r} \left| D_{i_1} \left[t_k, x_l, ..., \frac{\partial^{\alpha_0+\alpha_1+...+\alpha_n}}{\partial t^{\alpha_0} \partial x_1^{\alpha_1} ... \partial x_n^{\alpha_n}} y_{T_i}(t_k, x_l, P_i) ... \right] \right|^P \right\}^{1/P}$$

$$+ \sum_{i_2=1}^{P_2} \left\{ \sum_{l=1}^{r} \left| I_{i_2} \left[t_0, x_l, ..., \frac{\partial^{\alpha_0+\alpha_1+...+\alpha_n}}{\partial t^{\alpha_0} \partial x_1^{\alpha_1} ... \partial x_n^{\alpha_n}} y_{T_i}(t_0, x_l, P_i) ... \right] \right|^P dx \right\}^{1/P}$$

$$+ \sum_{i_3=1}^{P_3} \left\{ \sum_{k=1}^{q} \sum_{l=1}^{r_0} \left| B_{i_3} \left[t_k, x_l^0, ..., \frac{\partial^{\alpha_0+\alpha_1+...+\alpha_n}}{\partial t^{\alpha_0} \partial x_1^{\alpha_1} ... \partial x_n^{\alpha_n}} y_{T_i}(t_k, x_l^0, P_i) ... \right] \right|^P \right\}^{1/P} \quad 1 \le i \le m \qquad (3.41)$$

Où $\{x_l^0\}_{l=1}^{r0}$ sont les composants d'espace des points de collocation de la limite.

III.7 Dérivation des dérivées partielles

Pour toutes fonctions différentiables d'ordre désirées $\gamma_i : R^{n+2} \rightarrow R$ $(i=1...m)$ nous supposons que $y_{Ti}(t, x, P) = \gamma_i[t, x, N_i(t, x, p)]$.La dérivée par rapport à t et x de y_{Ti} a besoin de la connaissance au sujet des dérivés partiels du réseau Ni [71,57].

Pour simplifier, nous prenons N au lieu du Ni .Considérons un réseau de neurone de type MLP avec trois couches comprenant $(n+1)$ entrées à valeurs réelles, une couche cachée avec H neurones et une couche de sortie. (Figure III.2).

Pour les valeurs d'entrée $(t, x_1,...x_n)$ la sortie du réseau est :

$$N(t, x_1, ..., x_n) = \sum_{i=1}^{H} v_i s \left[w_i t + (\sum_{j=1}^{n} w_{ij} x_j) + b_i \right] \qquad (3.43)$$

où :

v_i : est un paramètre de poids de la i[éme]couche cachée à la couche de sortie.

w_i :est un paramètre de poids pour le temps de la couche d'entrée à la i[éme]couche caché.

W_{ij} : est un paramètre de poids de la j[éme]couche d'entrée à la i[éme]couche caché.

b_i :est un biais pour la i[éme]couche cachée.

s : est la fonction d'activation.

Ici, le $(n+3)H$ paramètres réglable sont les composantes de vecteur $(w_1...w_H, w_{11}...w_{H1},..., w_{1n}...w_{Hn}, v_1...v_H, b_1...b_H)$.

Théorème 3 : nous avons :

$$\frac{\partial^k}{\partial t^k} N = \sum_{i=1}^{H} v_i w_i^k s^{(k)} \left[w_i t + (\sum_{j=1}^{n} w_{ij} x_j) + b_i \right] \qquad (3.44)$$

$$\frac{\partial^k}{\partial x_l^k} N = \sum_{i=1}^{H} v_i w_{il}^k s^{(k)} \left[w_i t + (\sum_{j=1}^{n} w_{ij} x_j) + b_i \right] \qquad l = 1, ..., n \qquad (3.45)$$

Où $s^{(k)}$ est la $k^{ième}$ dérivée de s.

Lemme : Dans le problème (3.39) à condition initial et au limite, si les solutions approchées y_{Ti} **(i=1…m)** sont sous la forme :

$$y_{T_i}(t, x, P) = \varphi_i(t, x) + \psi_i \left[t, x, N_i(t, x, P) \right] \qquad i = 1, ..., m \qquad (3.47)$$

Où φ_i satisfait les conditions et ψ_i disparaît dans les conditions,

Alors

$y_T = (y_{T1} \ldots y_{Tm})$ satisfait les conditions initiaux et aux limites.

III.8 Conclusion

Dans ce chapitre on a présenté une méthode basée sur les réseaux de neurones de type MLP et les techniques d'optimisation pour résoudre les problèmes de valeurs initiales et aux limites. Cette méthode peut entraîner l'amélioration des méthodes numériques pour résoudre les problèmes des valeurs initiales ou aux limites, sans utiliser des points de discrétisation pré attribuées.

Quelques avantages de la méthode présentée comprennent:

- La possibilité de résoudre les systèmes d'équations aux dérivées partielles dépendants du temps.
- La généralité de la méthode dans la résolution des problèmes d'ordre supérieur et les problèmes non linéaires.
- La détermination de la solution approchée de problème sous une forme analytique fermée.
- L'évaluation rapide de la solution, traitée avec un certain nombre de paramètres [58].

Dans le chapitre suivant on va appliquer cette méthode pour la résolution d'un système d'équation qui régit le phénomène de stockage de l'énergie thermique (système d'équations de chaleur).

Chapitre IV

Implémentation et Résultats

IV.1 Introduction

Après avoir abordé l'aspect théorique de notre projet, il serait intéressant de passer à l'aspect pratique, c'est à dire la conception et l'implémentation de notre application .Cette application consiste à résoudre le système d'équations qui régit le phénomène de stockage d'énergie thermique .Pour démontrer le comportement et les propriétés de la méthode, nous entreprenons des exemples de différent ordre.

IV.2 Environnement de développement

Notre application a été réalisée sur un ordinateur portable (Toshiba Satellite C660) ayant les caractéristiques suivantes:

• Micro processeur Core I3 M360.
• 4Go de RAM.
• Disque dur 500 Go.
• Système d'exploitation "Microsoft Windows XP Professionnel ".

IV.3 Langages de programmation

Le langage de programmation utilisé pour le calcul est **MATLAB 7.5** version **R2007b**.

MATLAB est un environnement puissant, complet et facile à utiliser destiné au calcul scientifique. Il apporte aux ingénieurs, chercheurs et à tout scientifique interactif intégrant calcul numérique et visualisation. Il dispose de plusieurs centaines de fonctions mathématiques, scientifiques et techniques .L'approche matricielle de MATLAB permet de traiter les données sans aucune limitation de taille et de réaliser des calculs numériques et symboliques de façon fiable et rapide.

IV.4 Présentation générale du problème de transfert de chaleur

Pour illustrer notre méthode, nous choisissons de résoudre un problème de transfert de chaleur. Les équations de l'énergie qui régissent le phénomène de stockage de la chaleur pour un modèle à deux phases séparées s'écrit:

- pour le fluide :

$$\rho_F C_F \varepsilon \frac{\partial T}{\partial t} + \rho_F C_F \varepsilon u \frac{\partial T}{\partial x} = \varepsilon K \frac{\partial^2 T}{\partial x^2} + h a(\Theta - T) \qquad (4.1)$$

- Pour le matériau de changement de phase (MCP) :

$$\rho_{MCP} C_{MCP} (1-\varepsilon) \frac{\partial \Theta}{\partial t} = h a(T - \Theta) \qquad \textit{Phase solide} \qquad (4.2)$$

$$\rho_{MCP} h_L (1-\varepsilon) \frac{\partial \Phi}{\partial t} = h a(T - \Theta^{\otimes}) \qquad \textit{Deux phases} \qquad (4.3)$$

$$\rho_{MCP} C_{MCP} (1-\varepsilon) \frac{\partial \Theta}{\partial t} = h a(T - \Theta) \qquad \textit{Phase liquide} \qquad (4.4)$$

Á ce système d'équations aux dérivées partielles, nous devons associer des conditions initiales et aux limites.

- **Conditions initiales et aux limites**
  - ***Á l'instant initial***

$$T(t = 0, x) = \Theta(t = 0, x) = T_a$$

  - ***Á (x =0)***

$$T(t, x = 0) = \Theta(t, x = 0) = T_e$$

  - ***Á (x =H= 0.6)***

$$\frac{\partial T}{\partial x}(t, x = 0.6) = \frac{\partial \Theta}{\partial x}(t, x = 0.6) = 0$$

Les caractéristiques de notre problème sont :

Domaine I (*t*, *x*)	$t \in [0, t_{max}], x \in [0, 0.6]$
ρ_F : Masse volumique du fluide	1.029 $m^3.kg^{-1}$
C_F : Chaleur spécifique du fluide	1.009*10^3 $J.Kg^{-1}.°C^{-1}$
ε : Porosité	0.4056
u : Vitesse interstitielle du fluide	0.5 $m.s^{-1}$
K : Conductivité thermique	2.96*10^{-2} $w.m^{-1}.°C^{-1}$
h : Coefficient d'échange thermique	17.6905 $w.m^{-2}.°C^{-1}$
a : Taux de surface de contact	93.853 m^{-1}
ρ_{MCP} : Masse volumique du MCP « Phase solide »	920 $m^3.kg^{-1}$
C_{MCP} : Chaleur spécifique du MCP« Phase solide »	8.4 $J.Kg^{-1}.°C^{-1}$
h_L : Chaleur latente de fusion	189 $J.Kg^{-1}$
ρ_{MCP} : Masse volumique du MCP « Phase liquide »	795 $m^3.kg^{-1}$
C_{MCP} : Chaleur spécifique du MCP« Phase liquide »	2.1 $J.Kg^{-1}.°C^{-1}$
Ta : Température ambiante	26.4°*C*
Te : Température d'entrée	70°*C*
$\Theta^{\otimes}$: Température de fusion	58°*C*
Φ : Titre du mélange de phases du MCP	m_L/m

Table IV.1: *Les caractéristiques du problème de transfert de chaleur.*

IV.5 Résolution du problème

Après avoir expliqué le problème de stockage d'énergie thermique. Nous passons à la résolution de ce dernier, et cela, en définissant la solution d'approximation sous la forme suivante:

$$y_T(t,x,P) = \alpha(t,x) + \beta[t,x,N(t,x,P)] \qquad (4.5)$$

Pour la phase solide

$$\alpha(t,x) = Ta + t((\frac{Ta+Te}{Te})x^2 - 2H(\frac{Ta+Te}{Te})x + (\frac{Ta+Te}{Te})) \qquad (4.6)$$

$$\beta[t,x,N(t,x,P)] = t(x-H)^2 N(t,x,P) \qquad (4.7)$$

Donc :

$$T(t,x,P) = Ta + t((\frac{Ta+Te}{Te})x^2 - 2H(\frac{Ta+Te}{Te})x + (\frac{Ta+Te}{Te})) + t(x-H)^2 N_1(t,x,P) \quad (4.8)$$

$$\Theta(t,x,P) = Ta + t((\frac{Ta+Te}{Te})x^2 - 2H(\frac{Ta+Te}{Te})x + (\frac{Ta+Te}{Te})) + t(x-H)^2 N_2(t,x,P) \quad (4.9)$$

- **Pour les deux phases**

$$T(t,x,P) = Ta + t((\frac{Ta+Te}{Te})x^2 - 2H(\frac{Ta+Te}{Te})x + (\frac{Ta+Te}{Te})) + t(x-H)^2 N_3(t,x,P) \quad (4.10)$$

$$\Phi(t,x,P) = (t-t_\alpha) N_4(t,x,P) \quad (4.11)$$

t_α : Le temps pour atteindre la température de fusion.

- **Pour la phase liquide**

$$\alpha(t,x) = Ta + t((\frac{Ta+Te}{Te})x^2 - 2H(\frac{Ta+Te}{Te})x + (\frac{Ta+Te}{Te})) \quad (4.12)$$

$$\beta[t,x,N(t,x,P)] = t(x-H)^2 N(t,x,P) \quad (4.13)$$

Donc :

$$T(t,x,P) = Ta + t((\frac{Ta+Te}{Te})x^2 - 2H(\frac{Ta+Te}{Te})x + (\frac{Ta+Te}{Te})) + t(x-H)^2 N_5(t,x,P) \quad (4.14)$$

$$\Theta(t,x,P) = Ta + t((\frac{Ta+Te}{Te})x^2 - 2H(\frac{Ta+Te}{Te})x + (\frac{Ta+Te}{Te})) + t(x-H)^2 N_6(t,x,P) \quad (4.15)$$

Tel que N_1 ,N_2,N_3,N_4,N_5et N_6 sont deux réseaux de neurones de la même architecture (figure IV.1)

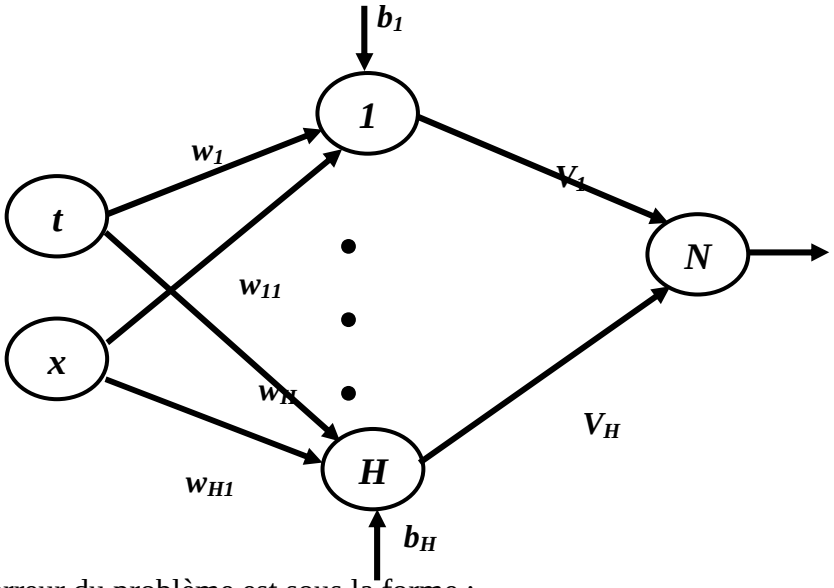


Figure IV.1: *Réseaux de neurones multicouches.*

La fonction d'erreur du problème est sous la forme :

- Pour la phase solide et la phase liquide

$$E=\sum_{i=1}^{m}\sum_{j=1}^{k}\left[\rho_F C_F \varepsilon\frac{\partial T(t_j,x_i)}{\partial t_j}+\rho_F C_F \varepsilon u\frac{\partial T(t_j,x_i)}{\partial x_i}-\varepsilon K\frac{\partial^2 T(t_j,x_i)}{\partial x_i^2}+\rho_{MCP} C_{MCP}(1-\varepsilon)\frac{\delta\Theta}{\delta t_j}\right]^2 \quad (4.16)$$

- Pour les deux phases

$$E=\sum_{i=1}^{m}\sum_{j=1}^{k}\left[\rho_F C_F \varepsilon\frac{\partial T(t_j,x_i)}{\partial t_j}+\rho_F C_F \varepsilon u\frac{\partial T(t_j,x_i)}{\partial x_i}-\varepsilon K\frac{\partial^2 T(t_j,x_i)}{\partial x_i^2}+\rho_{MCP} h_L(1-\varepsilon)\frac{\delta\Phi}{\delta t_j}\right]^2 \quad (4.17)$$

Afin de calculer l'erreur, on doit calculer les dérivées des fonctions suivantes :

$$\frac{\delta\alpha(t,x)}{\delta t}=((\frac{Ta+Te}{Te})x^2-2H(\frac{Ta+Te}{Te})x+(\frac{Ta+Te}{Te}) \quad (4.18)$$

$$\frac{\delta\beta(t,x)}{\delta t}=(x-H)^2N(t,x,p)+t(x-H)^2\frac{\delta N}{\delta t} \quad (4.19)$$

$$\frac{\delta\alpha(t,x)}{\delta x}=t(2(\frac{Ta+Te}{Te})x-2H(\frac{Ta+Te}{Te})) \quad (4.20)$$

$$\frac{\delta\beta(t,x)}{\delta x}=2t(x-H)N(t,x,p)+t(x-H)^2\frac{\delta N}{\delta x} \quad (4.21)$$

$$\frac{\delta^2\alpha(t,x)}{\delta x^2}=2t(\frac{Ta+Te}{Te}) \quad (4.22)$$

$$\frac{\delta^2\beta(t,x)}{\delta x^2}=2tN(t,x,p)+4t(x-H)\frac{\delta N}{\delta x}+t(x-H)^2\frac{\delta^2 N}{\delta x^2} \quad (4.23)$$

$$\frac{\delta\Phi(t,x,P)}{\delta t}=N(t,x,P)+(t-t_\alpha)\frac{\delta N}{\delta t} \quad (4.24)$$

IV.6 Résultats obtenus :

Dans ce travail on a choisit un réseau de neurones multi couches et on va varier le nombre de neurones dans la couche cachée. Dans le calcule de la température de fluide ***T,*** la température de MCP Θ ainsi que pour le calcule du titre Φ :

IV.6.1 Essai 1

on a pris un réseau de neurones composé de trois neurones dans la couche cachée (*H=3*) pour le calcule de la température de fluide ***T*** et la température de ***MCP*** Θ , et un réseau de neurones composé de trois neurones dans la couche cachée (*H=3*) pour le calcule du titre **Φ**.

Les résultats obtenus sont illustrés par les figures suivantes :

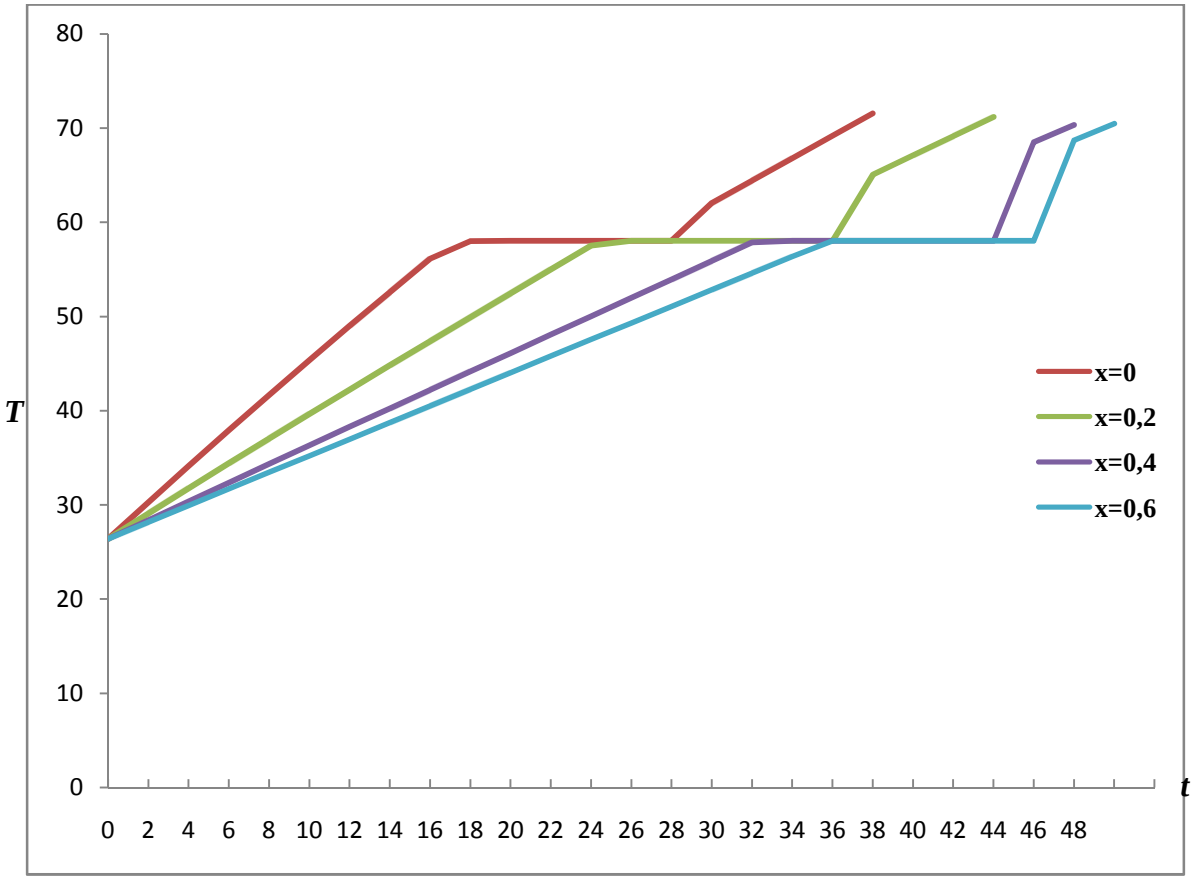


Figure IV.2: *Evolution de la température de fluide (solide-deux phase-liquide)*

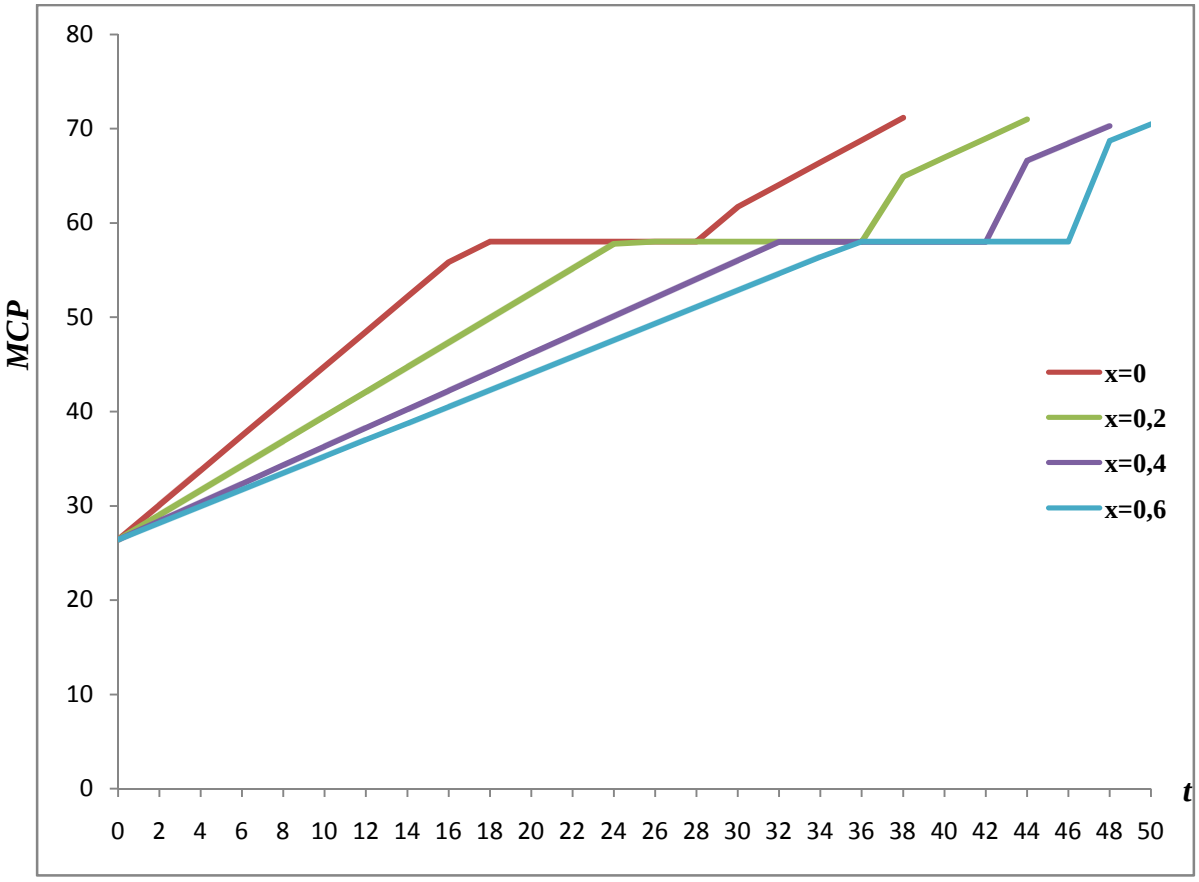


Figure IV.3: *Evolution de la température de MCP (solide-deux phase-liquide).*

L'erreur obtenue est : 5.4566

IV.6.2 Essai 2

On a pris un réseau de neurones composé de cinq neurones dans la couche cachée (*H=5*) pour le calcule de la température de fluide ***T*** et la température de ***MCP*** Θ , et un réseau de neurones composé de trois neurones dans la couche cachée (*H=3*) pour le calcule du titre Φ .

Les résultats obtenus sont illustrés par les figures suivantes :

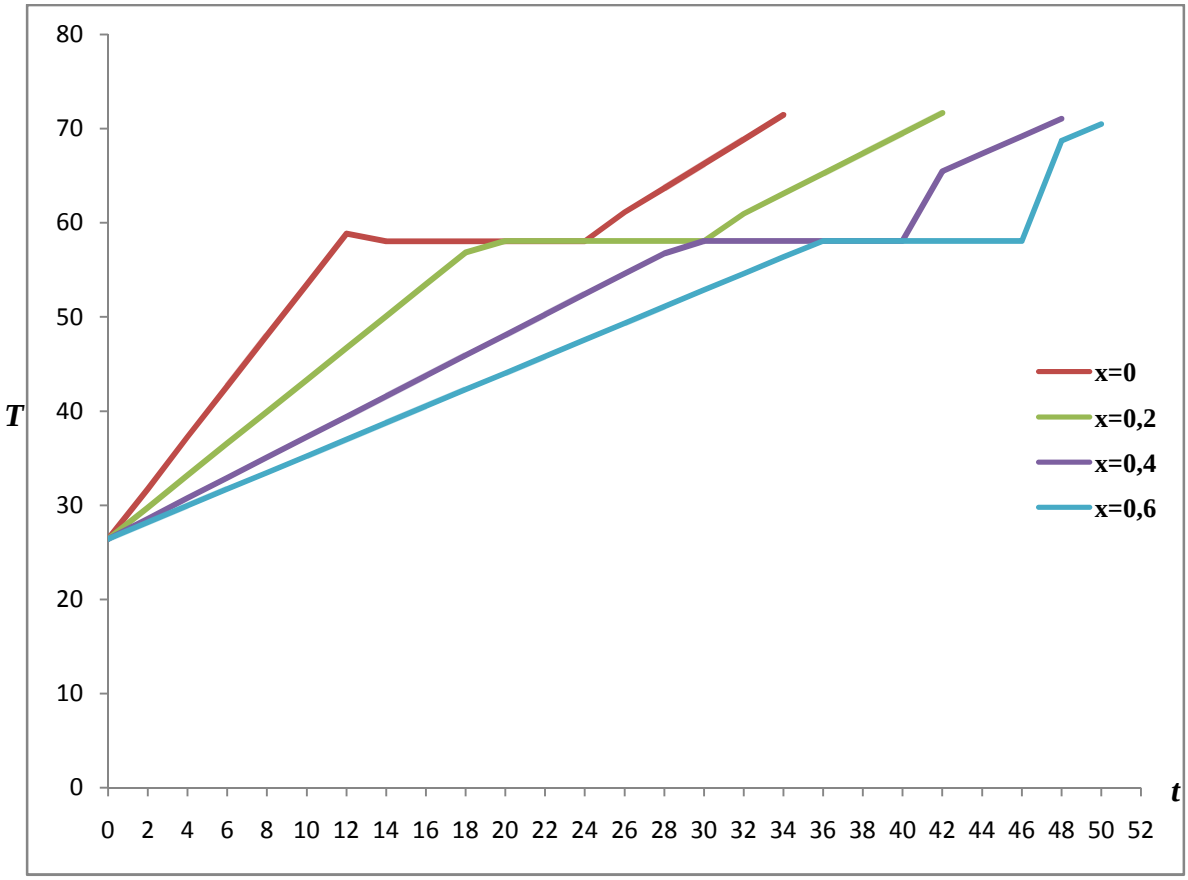

80
70
60
50
40
30
20
10
0
T
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52
t
x=0
x=0,2
x=0,4
x=0,6

Figure IV.4: *Evolution de la température de fluide (solide-deux phase-liquide)*

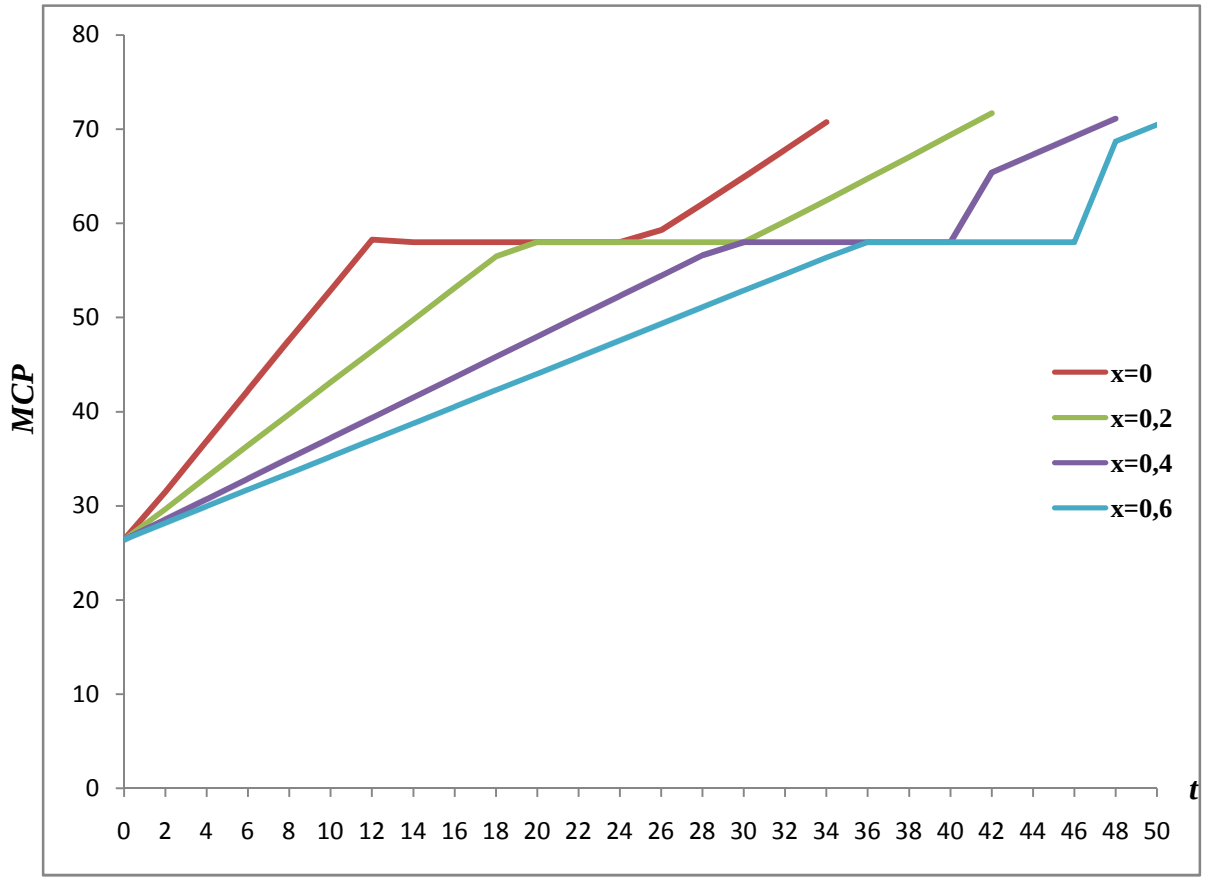


Figure IV.5: *Evolution de la température de MCP (solide-deux phase-liquide).*

L'erreur obtenue est : 0.0305

IV.6.3 Essai 3

De la même façon on prend (*H=7*) pour le calcule de la température de fluide ***T*** et la température de ***MCP*** Θ , et (*H=3*) pour le calcule du titre Φ .

Les résultats obtenus sont illustrés par les figures suivantes :

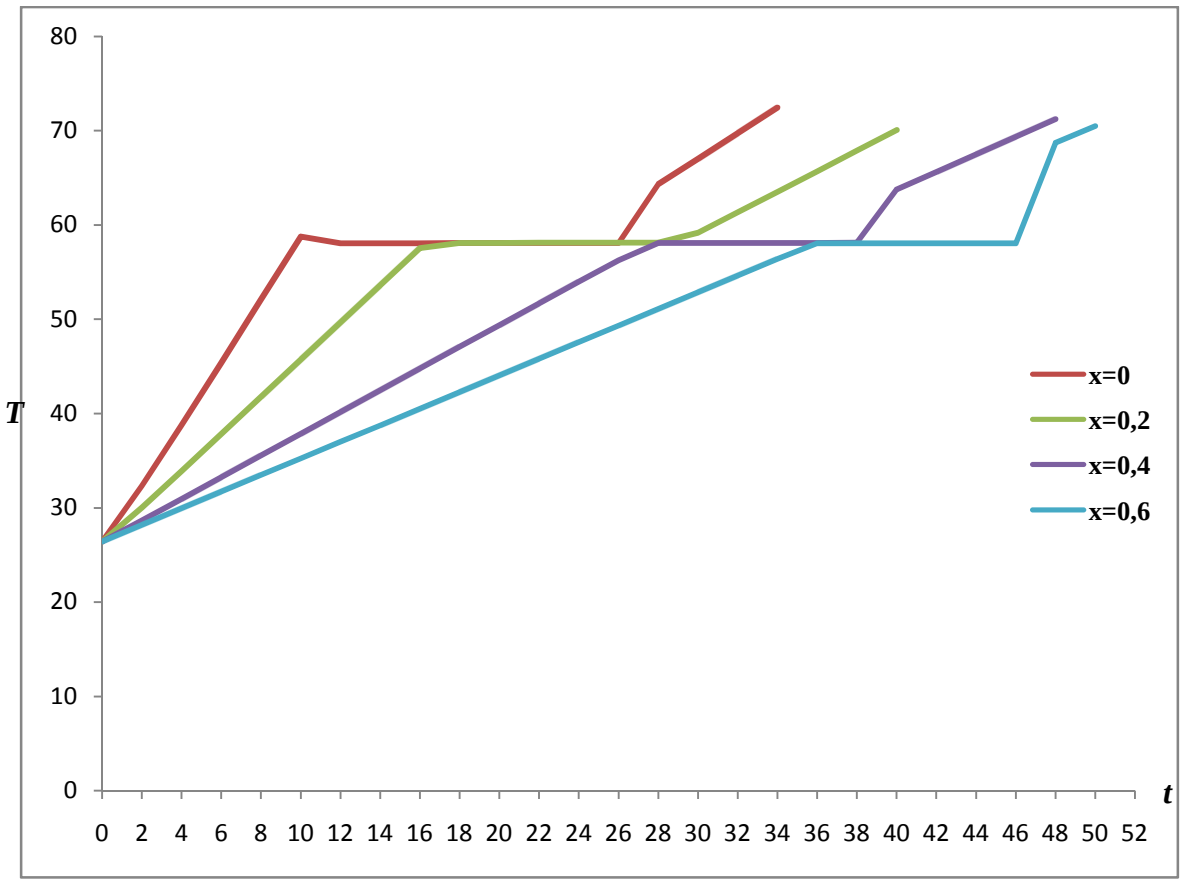


Figure IV.6: *Evolution de la température de fluide (solide-deux phase-liquide)*

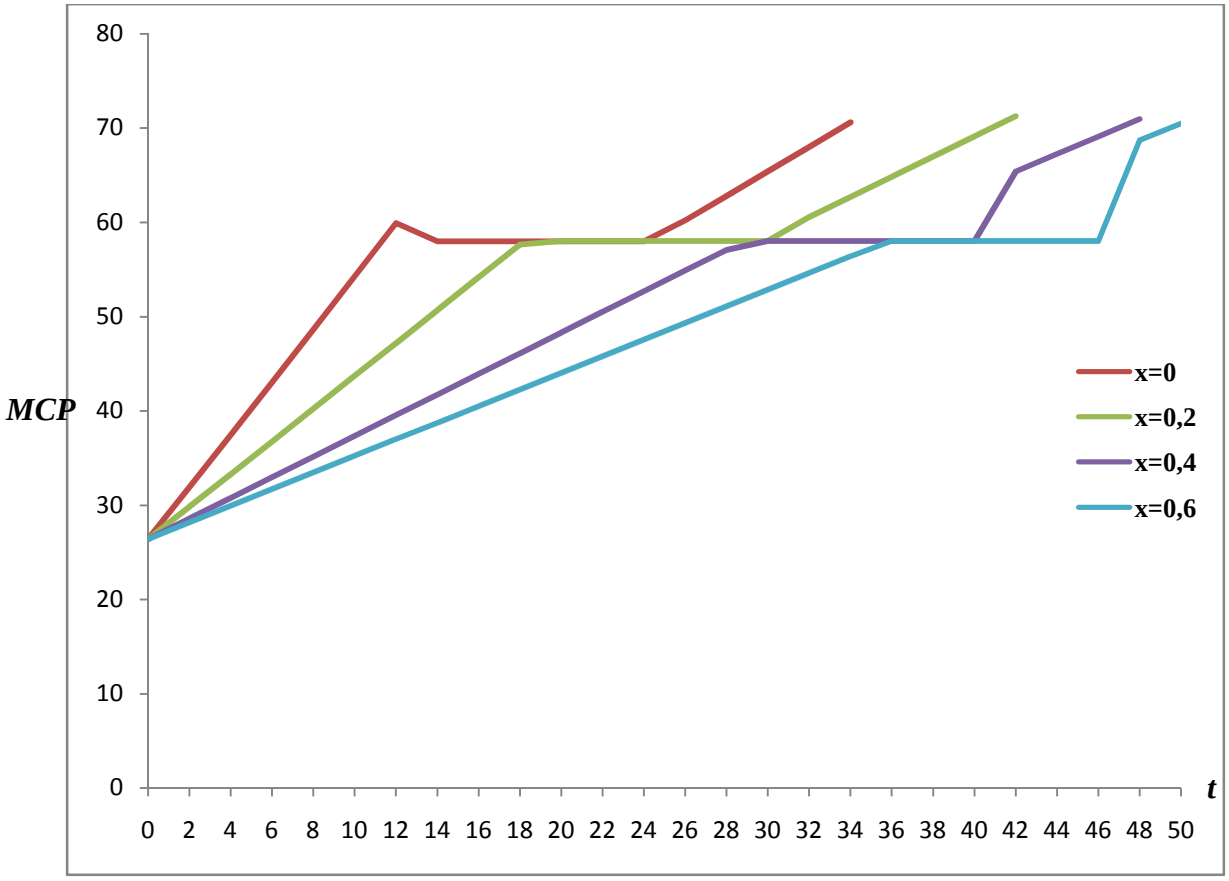


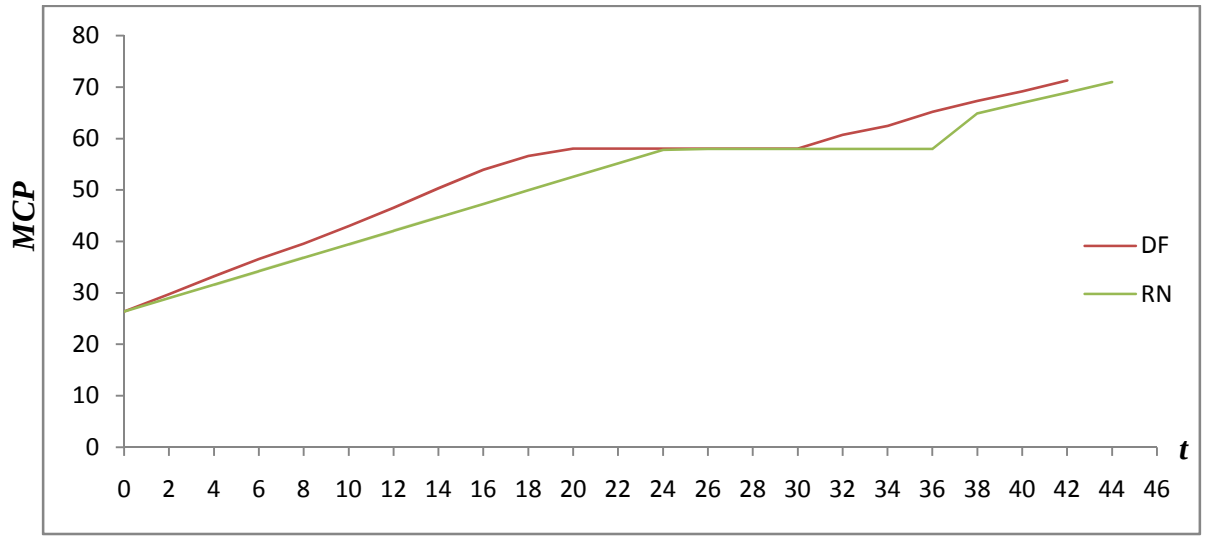
Figure IV.7: *Evolution de la température de MCP (solide-deux phase-liquide).*

L'erreur obtenue est : 0.1180

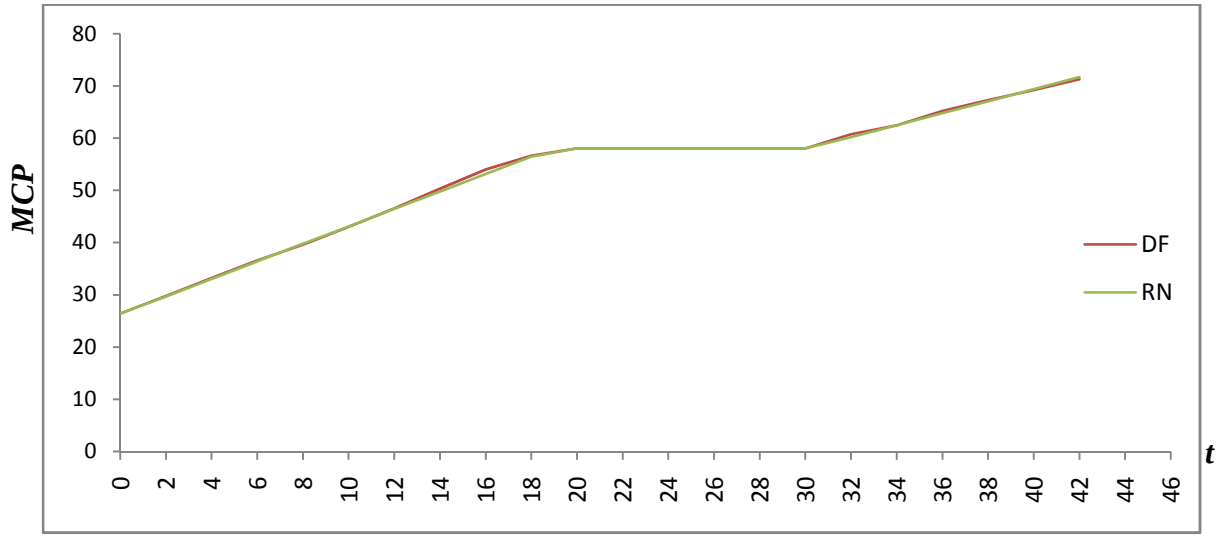
IV.7 Discussion et comparaison

En variant le nombre de neurones de la couche cachée, on a obtenu différent résultats qu'on va comparer avec les résultats obtenus par la méthode de différence fini qui a été déjà appliquée dans le même domaine.

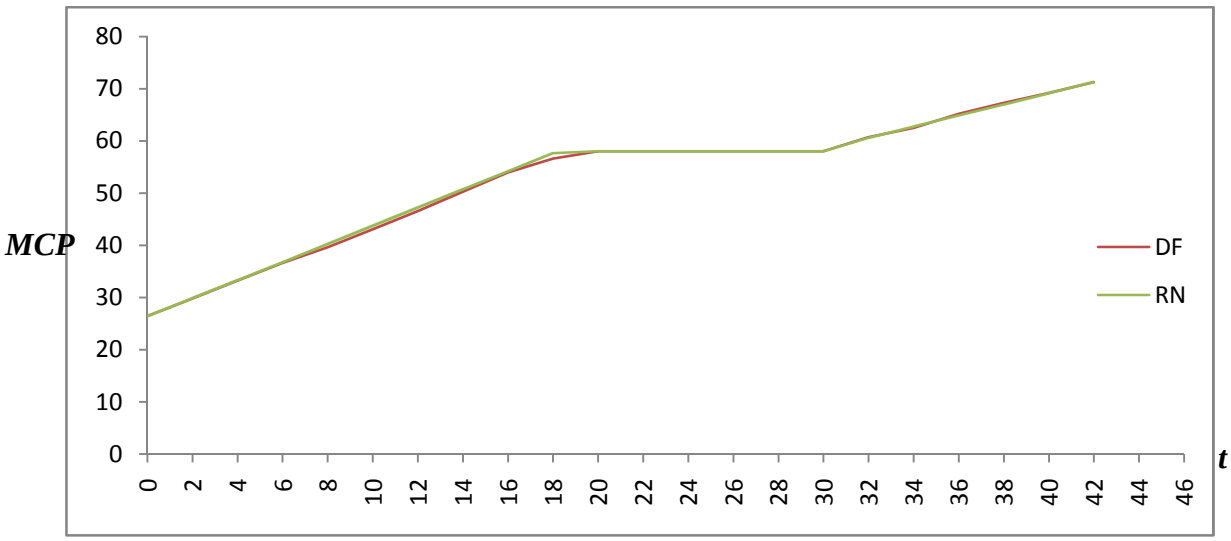
La comparaison est montrée par les figures suivantes :



Nombre de neurones dans la couche cachée est :3

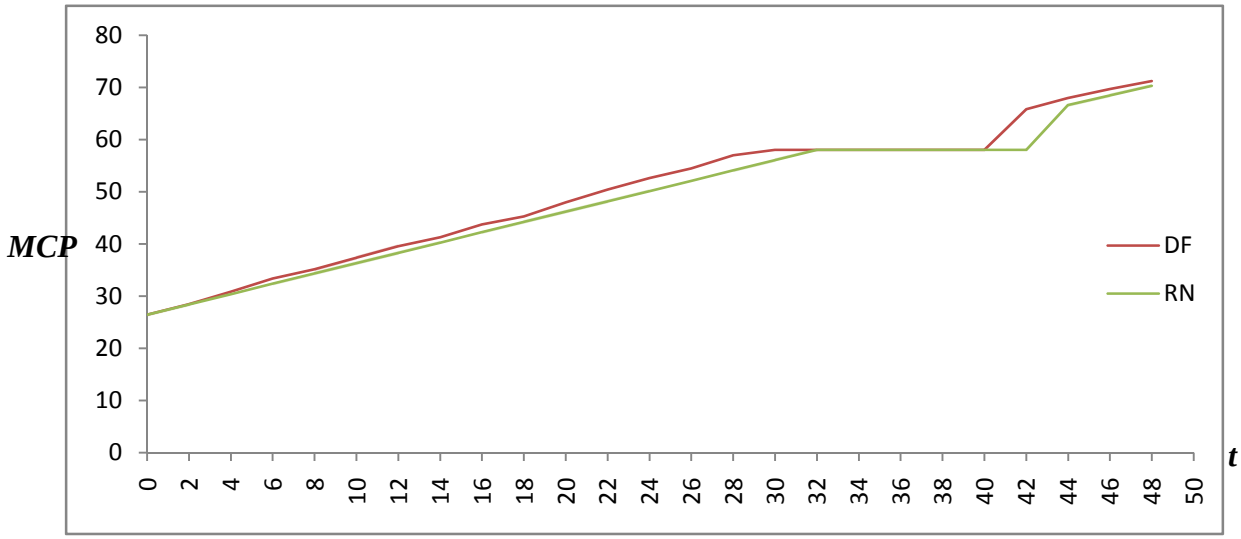


Nombre de neurones dans la couche cachée est : 5

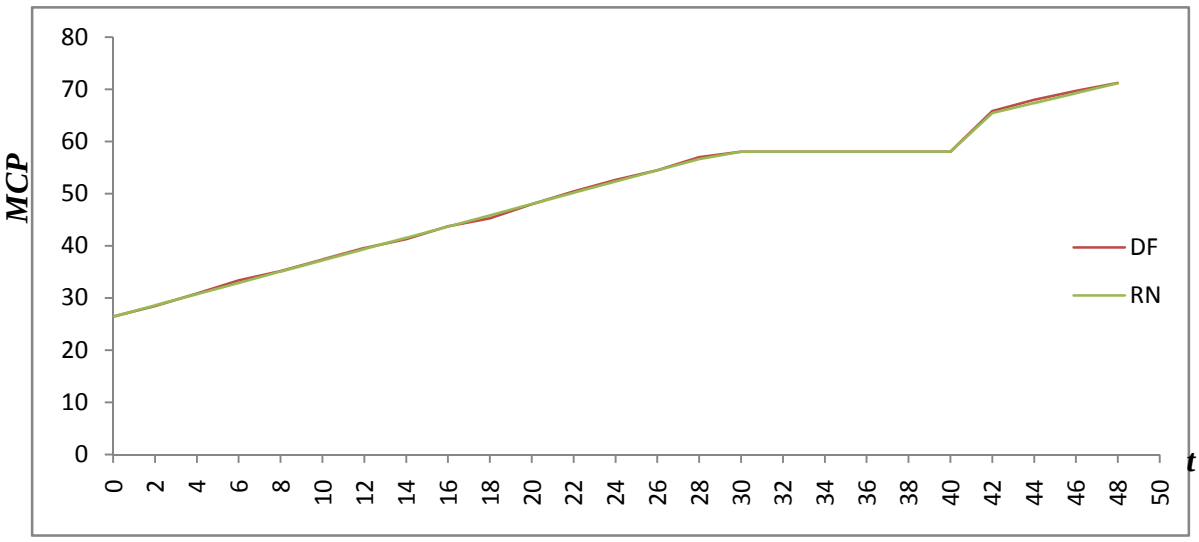


Nombre de neurones dans la couche cachée est : 7

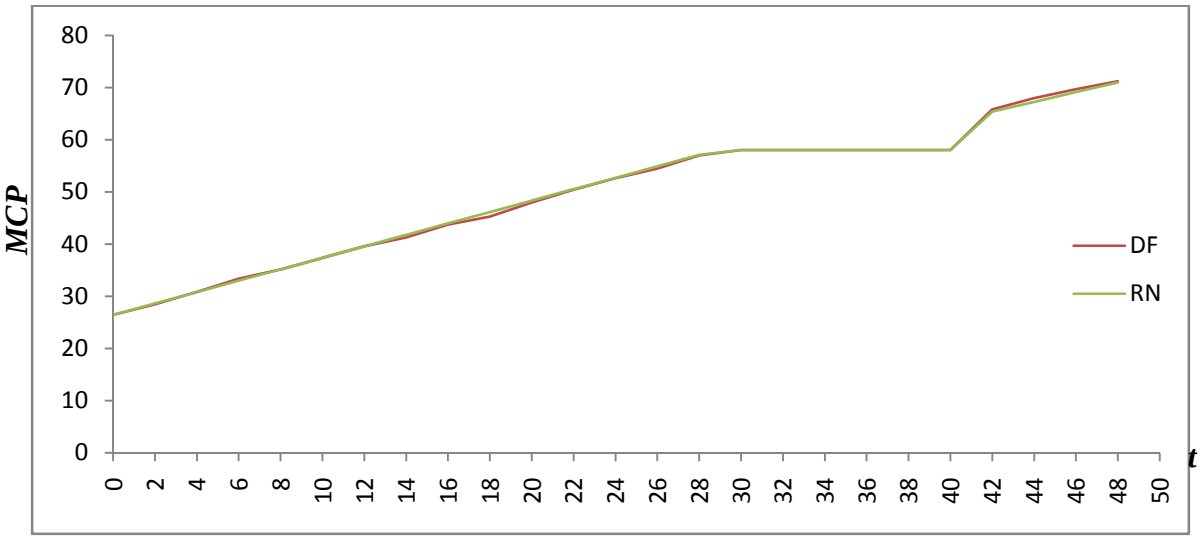
Figure IV.8: *comparaison entre les résultats obtenus par les réseaux de neurones et les résultats obtenus par la différence finie pour* ***x=0.2*** *pour (H=3, 5,7) (solide-deux phase-liquide).*



Nombre de neurones dans la couche cachée est : 3



Nombre de neurones dans la couche cachée est :5



Nombre de neurones dans la couche cachée est : 7

Figure IV.9: *comparaison entre les résultats obtenus par les réseaux de neurones et les résultats obtenus par la différence finie pour* ***x=0.4*** *pour (H=3, 5,7) (solide-deux phase-liquide).*

La comparaison des trois essais avec la différence fini pour les deux points montre que le meilleur résultat est obtenu par un réseau de neurones composé de cinq neurones dans

la couche cachée (*H=5*) pour le calcule de la température de fluide T et la température de MCP Θ, et un réseau de neurones composé de trois neurones dans la couche cachée (*H=3*) pour le calcule du titre Φ (essai 2) ; en plus l'erreur est minimal.

Nos simulations numériques prouvent que la solution proposée donne un bon accord entre les valeurs calculées par les réseaux de neurones et les valeurs calculées par la différence fini.

IV.8 Illustration de la méthode par des exemples

IV.8.1 Exemple 1: avec la solution exacte y(x) =$e^x + 1$, x ∈ [0, 1] sous la forme

$$\begin{cases} \dfrac{dy}{dx} = e^x \qquad x \in [0,1] \\ \\ y(0) = 2 \end{cases}$$

La fonction d'approximation pour ce problème est :

$$y_T(x,P) = 2 + xN(x,P) = 2 + x\sum_{i=1}^{H} \frac{v_i}{1+e^{-w_i x - b_i}}$$

Tandis que la fonction d'erreur est sous la forme :

$$E = \sum_{i=1}^{m} \left(\frac{dy_T(x_i,P)}{dx_i} - e^{x_i} \right)^2$$

Cette fonction d'erreur pour *H = 5* unités sigmoïdes dans la couche cachée est formée.

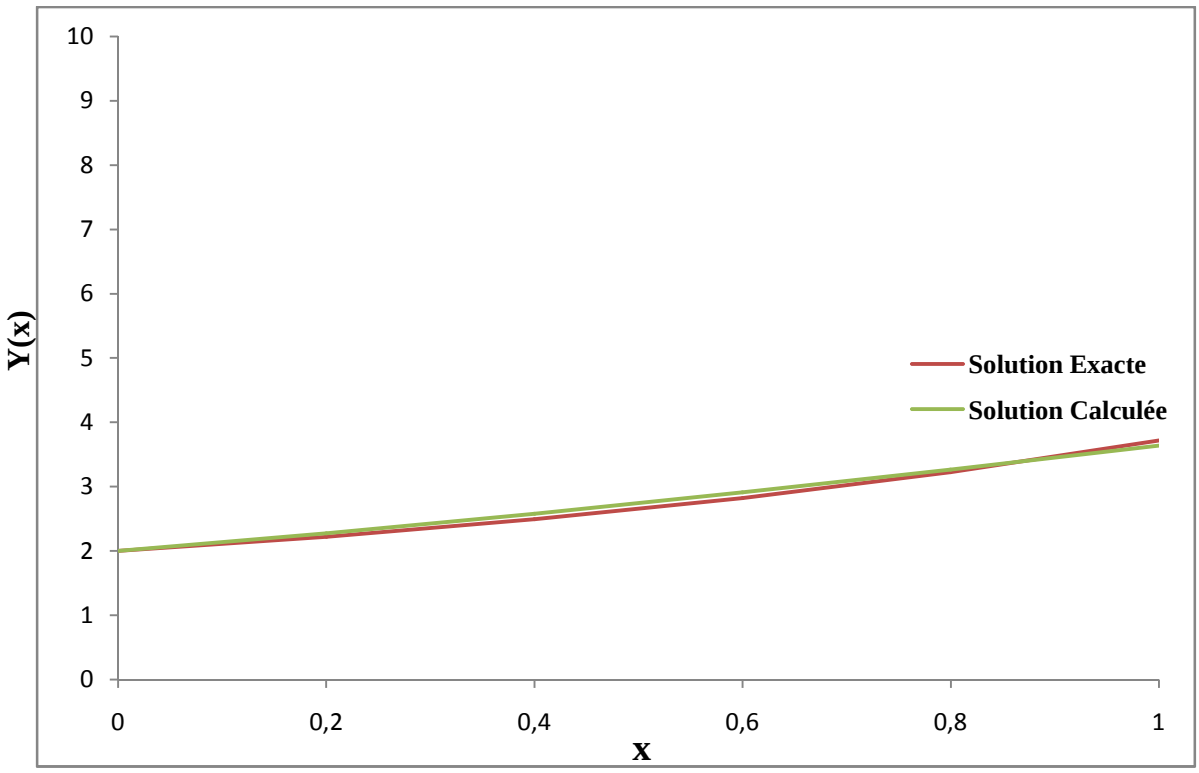


Figure IV.10: *Graphe de comparaison de la solution exacte avec de la solution calculée à l'intérieur du domaine [0, 1] pour l'exemple1.*

IV.8.2 Exemple 2: avec la solution exacte $y(x) = e^{\frac{-x}{5}} \sin(x)$, $x \in [0,2]$ sous la forme :

$$\begin{cases} \frac{d^2y}{dx^2} + \frac{1}{5}\frac{dy}{dx} + y = \frac{-1}{5} e^{\frac{-x}{5}} \cos(x) & x \in [0,2] \\ y(0) = 0 \qquad y(1) = \sin(1) e^{\frac{-1}{5}} \end{cases}$$

La fonction d'approximation pour ce problème est :

$$y_T(x, P) = x \sin(1) e^{\frac{-1}{5}} + x(1-x) N(x, P)$$

La fonction d'erreur est sous la forme :

$$E = \sum_{i=1}^{m} \left(\frac{d^2 y_T(x_i, P)}{dx^2} + \frac{1}{5}\frac{dy_T(x_i, P)}{dx} + y_T(x_i, P) + \frac{1}{5} e^{\frac{-x_i}{5}} \cos(x_i) \right)^2$$

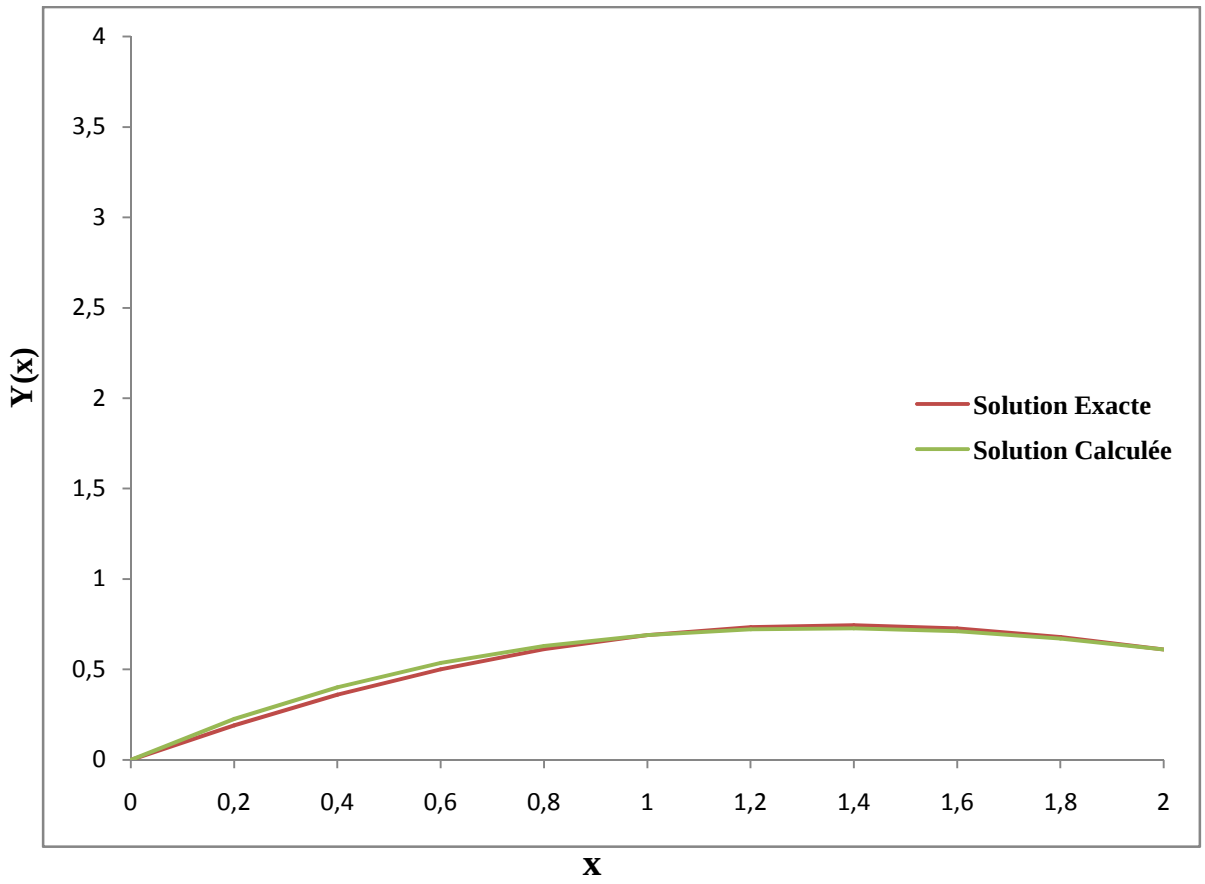


Figure IV.11: *Graphe de comparaison de la solution exacte avec de la solution calculée à l'intérieur du domaine [0, 2] pour l'exemple2.*

Cette fonction d'erreur pour *H = 3* unités sigmoïdes dans la couche est formée.

IV.8.3 Exemple 3: C'est un exemple de PDE avec la solution exacte Ψ(x, y)= 2x+y+1, sous la forme :

$$\begin{cases} \dfrac{d\Psi}{dx}+\dfrac{d\Psi}{dy}=3 \quad x\in[0,1] \quad y\in[0,1] \\ \Psi(0,y)=y+1 \quad \Psi(x,0)=2x+1 \\ \Psi(1,y)=y+3 \quad \Psi(x,1)=2x+2 \end{cases}$$

La fonction d'approximation pour ce problème est :

$$\Psi_T(x,y,P)=2x+y+1+x(1-x)y(1-y)N(x,y,P)$$

La fonction d'erreur est sous la forme :

$$E = \sum_{i=1}^{m} \left(\frac{d\Psi_T(x_i, y_i, P)}{dx} + \frac{d\Psi_T(x_i, y_i, P)}{dy} - 3 \right)^2$$

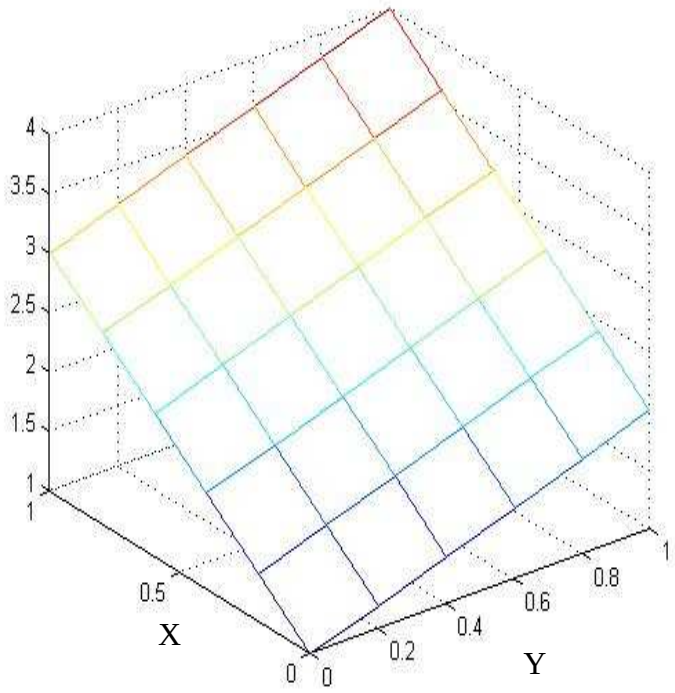


Figure IV.12*: Graphe de la solution exacte à l'intérieur du domaine [0, 1] pour l'exemple3.*

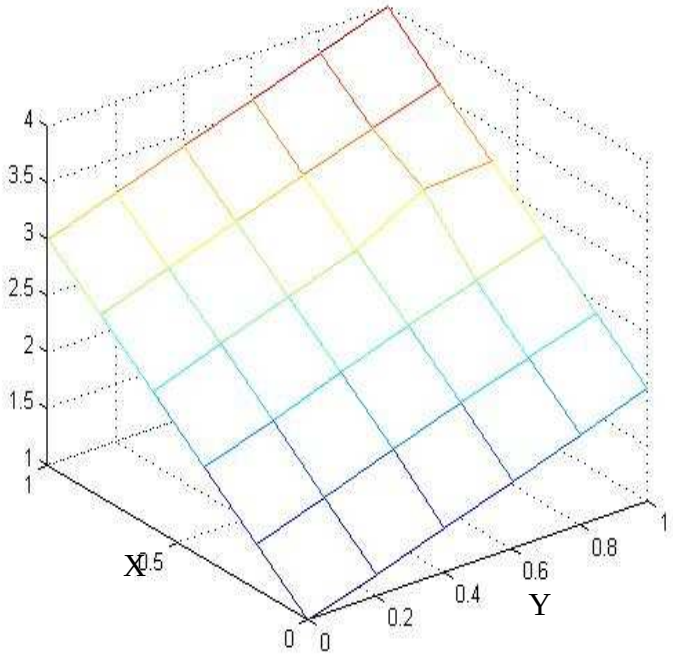


Figure IV.13: *Graphe de la solution calculée à l'intérieur du domaine [0, 1] pour l' exemple3.*

IV.9 Conclusion

On présente une méthode pour la résolution des problèmes à valeur initial et/ou aux limites basés sur les réseaux de neurones et les techniques fonctionnelles de minimisation.

L'apprentissage des réseaux de neurones mène à des valeurs optimales pour les paramètres réglables. Ensuite la solution d'approximation en ce qui concerne les variables du problème original est construite. La méthode de minimisation utilisée ici est la méthode de rétro propagation.

Conclusion générale

Dans ce mémoire, Nous nous sommes intéressés au problème de la résolution des équations différentielles et des équations aux dérivées partielles.

Les méthodes classiques comme la Méthode des Eléments Finis, la Méthode des Différences Finies et la Méthode des Volumes Finis sont restées encore à nos jours les méthodes les plus utilisées pour résoudre ce genre de problème.

Ces méthodes bénéficient d'un fondement théorique très solide, et de nombreuses techniques sont venues les améliorer au fil du temps. Cependant, leur mise en œuvre reste difficile et couteuse dans certains cas.

Dans ce but nous avons présenté une approche basée sur les réseaux de neurones et les méthodes d'optimisations (Rétro propagation). La solution d'approximation de l'équation différentielle est écrite comme somme de deux parties. La première partie satisfait la conditions initiale et/ou limite et ne contient aucun paramètre réglable. La deuxième partie est un réseau de neurones multicouche.

Cette approche a été appliquée pour la résolution d'un système d'équation qui régit le phénomène de stockage de l'énergie thermique (système d'équations de transfert de chaleur).

Ceci a donné comme avantages: l'évaluation rapide de la solution ; Traitement avec un petit nombre de paramètres.

Les tests que nous avons établir ont prouvé que la méthode utilisée est très économique en matière de temps des calculs et de programmation par rapport aux méthodes des différences finies ou celles des éléments finis.

Les résultats obtenus ont été comparés avec la méthode de différence finie et elles montrent qu'il y a un bon accord entre les deux résultats.

Enfin, il est important de noter que si la dimension de l'équation augmente, le nombre de points pour l'apprentissage devient large.

Comme perspectives, nous visons à prolonger notre approche et cela pour traiter d'autres problèmes de la même nature par exemple les problèmes de valeur propre pour les opérateurs différentiels.

La méthode présentée peut facilement être employée pour traiter des problèmes dans des domaines de dimensions plus élevées.

BIBLIOGRAPHIE

[1] C.H. Papadimitriou, K. Steiglitz, Combinatorial optimization – algorithms and complexity. Prentice Hall, 1982.

[2] C.C. Ribeiro, N. Maculan (Eds.), Applications of combinatorial optimization.
Annals of Operations Research 50, 1994.

[3]M.R. Garey, D.S. Johnson, Computers and intractability: a guide to the theory of NP-completeness,W.H.Freeman and Company, New York, 1979.

[6]Y.Collette and P.Siarry. Optimisation multiobjectif. Eyrolles 2002.

[7] P.Christophe.Les métaheuristiques en optimisation combinatoire. Paris 2006.

[8] K.Belkadi. « Les métaheuristiques » Cours, Usto ; 2006

[9] Grégoire Allaire « Analyse numérique et optimisation », Les éditions de l'école polytechnique, Novembre 2007.

[11] Jean-François Hêche,Thomas M.Liebling, Dominique de Werra, « Recherche opérationnelle pour ingénieurs II », Presses polytechniques et universitaires romandes.2003

[13] Cours de Master SIS « Outils d'aide à la décision », , Laboratoire des Sciences et de l'information et des systèmes. 2009-2010.

[15] LIACS .Simulated Annealing [Rapport]; Leiden University. 2009.

[16] Autin Baptiste.Les métaheuristiques en optimisation combinatoire [Rapport] Mémoire de fin d'etudes / Conservatoire National Des Arts et Metiers. – PARIS. 2006.

[17] D.Olivier. Méthode du recuit simulé [Rapport] : complément TD/TP Recherche stochastique.2001.

[18] Abecasis Goncalo. Simulated Annealing, Cours Biostatistiques - chapitre19/.2011.

[19] Kendall Graham AI Methods - Simulated Annealing [Rapport] : complément de cours /CS - Nottingham University.2000

[22] Xavier Hüe, "Genetic Algorithms for Optimisation: Background and Applications", Edinburgh Parallel Computing Centre, University of Edinburgh, 1997.

[23] Benyamina Redha, Houari Djalal, "Conception d'un système hybride R.N.MLP-AGs pour une classification de données de télédétection",2004/2005.

[24] Souquet Amédée, Radet Francois-Gérard "Algorithmes Génétiques", 2004.

[26] Max Moorkamp, "Genetic Algorithms: A step by step tutorial", Dublin Institute for Advanced Studies, Barcelona, 29th November 2005.

[27] M,Parizeau « Réseaux de Neurones », Université de LAVAL, 2004.

[28] Marc Parizeau. Réseaux de Neurones. GIF-21140 et GIF-64326-Université LAVAL.2006.

[29] C.Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.

[30] Yann Morere. Identification par Réseaux de Neurones.1995-1996.

[31] Sylvain Barthelemy « Les Réseaux de Neurones », 2000.

[32] Ed. Eyrolles. Des Réseaux de Neurones .1993.

[33] Yann Morere. Identification par Réseaux de Neurones.1995-1996.

[34] Ed. Addison-Wesley. Algorithmes Génétiques.1994.

[35] K. Hornik, M. Stinchcombe, H. White, "Multilayer Feedforward Networks"1989.

[36]G.Dreyfus. « Les Réseaux De Neurones ». Mécanique Industrielle et Matériaux, n°51 septembre 1998.

[37] Matoussi Hamadi et Krichéne Abdelmoula Aida « Prévision du risque des crédits de gestion par les modèles basés sur les réseaux de neurones artificiels », Sfax.2010.

[38] Roussillon Cycril, « Prévision De Température par réseaux Neuronaux », Lycée Victor Hugo BESANCON, 2004.

[39] Hafiane Mohamed Lamine « Conception d'un capteur de pression intelligent », Mémoire présenté pour l'obtention du diplôme de Magister en Microélectronique, Université de Batna, 2005**.**

[40] M.Ibnnkahla, "Application of neural networks to digital communication", Signal processing 80, pp 1185-1215, 2000.

[41] K. Hornik, "Multilayer feed forward networks are universal approximators," Neural Networks, vol.2, pp .359-366. 1989.

[42] Yann Morere. Identification par Réseaux de Neurones.1995-1996.

[43] J. Stoer, R. Bulirsch, Introduction to Numerical Analysis, second ed., Springer-Verlag, New York, 1993.

[44] J.D. Lambert, Computational Methods in Ordinary Differential Equations, John Wiley & Sons, New York, 1983.

[45] D. Gottlieb, S.A. Orszag, Numerical analysis of spectral methods: theory and applications, CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 26, SIAM, Philadelphia, 1977.

[46] H.Lee, I.S. Kang, Neural algorithms for solving differential equations, Journal of Computational Physics 91-110–131. 1990

[47] A.J. Meade Jr, A.A. Fernandez, The numerical solution of linear ordinary differential equations by feedforward neural networks,Mathematical and Computer Modelling 19 (12)1–25. 1994.

[48] A.J. Meade Jr, A.A. Fernandez, Solution of nonlinear ordinary differential equations by feedforward neural networks, Mathematical and Computer Modelling 20 (9) 19–44. 1994.

[49] M.W.M.G. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations,Communications in Numerical Methods in Engineering 10-195–201. 1994

[50] Bo-An Liu, B. Jammes, Solving ordinary differential equations by neural networks, in: Proceeding of 13th European Simulation Multi-Conference Modelling and Simulation: A Tool for the Next Millennium, Warsaw, Poland, June 1–4, 1999.

[51] I.E. Lagaris, A.Likas,D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Transactions on Neural Networks 9 (5) (1998) 987–1000.

[52] K. Valasoulis, D.I. Fotiadis, I.E. Lagaris, A. Likas, Solving differential equations with neural networks: implementation on a DSP platform, in: Proceeding of 14th International Conference on Digital Signal Processing, Santorini, Greece, July 2002, pp. 1265–1268.

[53] D.R. Kincaid, E.W. Cheney, Numerical Analysis: Mathematics of Scientific Computing, third ed., Brooks/Cole, Pacific Grove, CA, 2002.

[54] A.Likas,D.A. Karras, I.E. Lagaris, Neural-network training and simulation using a multidimensional optimization system, International Journal of Computer Mathematics 67-33–46. 1998

[55] P.Van der Smagt, Minimization methods for training feedforward neural networks, Neural Networks 7 (1) 1–11.1994.

[56] A. Malek, T.N. Phillips, Pseudospectral collocation methods for fourth-order differential equations, IMA Journal of Numerical Analysis 15-523–553.1995.

[57] A. Malek, R. Shekari Beidokhti, Numerical solution for high order differential equations using a hybrid neural network-optimization method, Appl. Math. Comput. 183- 260–271.2006.

[58] R. Yentis, M.E. Zaghoul, VLSI implementation of locally connected neural network for solving partial differential equations, IEEE Trans. Circ. Syst. Fund. Theor. Appl. 43 (8)687–690.1996.

[59] F. Rosenblatt, Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms, Spartan Books, Washington, DC, 1962.

[60] K. Hornik,M. Stinchcombe, H.White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, Neural Network 3- 551–560.1990.

[61] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Network 2 (5)359–366.1989.

[62] V.Y. Kreinovich, Arbitrary nonlinearity is sufficient to represent all functions by neural networks: a theorem, Neural Network 4- 381–383.1991.

[63] E.K. Blum, L.K. Li, Approximation theory and feed forward networks, Neural Network 4-511–515.1991.

[64] A.N. Kolmogorov, On the representation of continuous functions of several variables by superpositions of continuous functions of one variable and addition, Dokl. Akad. Nauk SSSR 28 (2) 55–59).1963

[65] G.G. Lorentz, Approximation of Functions, Athena Series: Selected Topics in Mathematics, Holt, Rinehart and Winston, New York, 1966.

[66] K. Atkinson,W. Han, Theoretical Numerical Analysis: a Functional Analysis Framework, second ed., Springer-Verlag, New York, 2005.

[67] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (4) (1989) 304–314.feed-forward neural nets, Neural Network 10 (8)1455–1463.1997

[68] T. Leephakpreeda, Novel determination of differential-equation solutions: universal approximation method, J. Comput. Appl. Math. 146 - 443–457.2002

[69] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Network 9 (5)987–1000.1998

[70] B.P. Van Milligen, V. Tribaldos, J.A. Jimenez, Neural network differential equation and plasma equilibrium solver, Phys. Rev. Lett. 75 (20) 3594–3597.1995.

[71] X. Wang, K. Shimizu, Approximate solution of Hamilton–Jacobi–Bellman equation by using neural networks and matrix calculus techniques, IEICE Trans. Fundam. Electron. Commun. Comput. Sci. E 84-A (6)1549–1556.2001.

[72] L.P. Aarts, P. Van der Veer, Neural network method for solving partial differential equations, Neural Process. Lett. 14- 261–271.2001.

[73] T. Leephakpreeda, Novel determination of differential-equation solutions: universal approximation method, J. Comput. Appl. Math. 146- 443–457.2002

[74] D.R. Parisi, M.C. Mariani, M.A. Laborde, Solving differential equations with unsupervised neural networks, Chem. Eng. Process. 42-715–721.2003.

[75] H. Alli, A. Ucar, Y. Demir, The solutions of vibration control problems using artificial neural networks, J. Franklin Inst. 340- 307–325.2003.

[76] N. Sukavanam, V. Panwar, Computation of boundary control of controlled heat equation using artificial neural networks, Int. Commun. Heat Mass Transfer 30 (8) 1137–1146.2003.

[77] N. Smaoui, S. Al-Enezi, Modelling the dynamics of nonlinear partial differential equations using neural networks, J. Comput. Appl. Math. 170- 27–58.2004.

[78] N. Smaoui, A hybrid neural network model for the dynamics of the Kuramoto–Sivashinsky equation, Math. Probl. Eng. 3-305–321.2004.

[79] R.Shkari Beidokhi,A.Malek , Solving initial boundary value problems for systems of partial differential equations using neural networks and optimization techniques.Departement of mathematics,Tarbiat modares University.Iran.2009

Site web:

[4]http://ww2.eivpparis.fr/chachoua/TER2007/%5BN%B027%5D_RapportTER_LE_GALL_ROBIN.pdf

[5] http://www.irit.fr/~Philippe.Muller/Cours/Optim_cb/cours_0809.pdf

[10] www.ens.uqac.ca/.../chapitre5propreprogrammationdynamique.pdf

[12] http://fr.wikipedia.org

[14] The Metaheuristic Network . http://www.metaheuristics.net/

[20] http://www.literrature.org/authors/darwin-charles/

[21] http://fr.wikipedia.org/wiki/Algorithme

[25] http://sis.univ-tln.fr/~tollari/TER/AlgoGen1/node5.html