N° d'ordre: 22 / 2009-M / MT

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L4ENSEIGNEMENT SUPERIEURE ET
DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE
<< HOUARI BOUMEDIENE>>
FACULTE DE MATHEMATIQUES

MEMOIRE

Présenté pour l'obtention du diplôme de Magister
EN: Mathématiques

Spécialité : Probabilités et Statistiques

Par : Mekki Riad

Sujet

Complexité des algorithmes dans la modélisation stochastique par réseaux de neurones

Soutenu le 27/04/2009, devant le jury composé de:

Mr-	**Aider Meziane**	**Président**
Mr-	**Djedour Mohamed**	**Directeur de Thèse**
Mrme-	**Djaballah Khedidja**	**Examinatrice**
Mr-	**Chaabane Djamel**	**Examinateur**
Mr-	**Ladjouz Salim**	**Invité**

Dédicace

Je dédie ce modeste travail à

A ma très chère mère qu'elle trouve l'aboutissement de ses sacrifices.

A ma femme Ryma, ma sœur Naïma, et mon frère Mustapha et mes amis qui ont été tout le temps à mes cotés.

A mon oncle Abdelkader pour son soutien et encouragement.

A tous mes amis et collèques de travail qui m'ont soutenu dans cette éprouve de prés ou de loin et qui ont été pour moi une source inépuisable.

RIAD

Remerciements

Je remercie le bon Dieu qui m'a orienté au chemin du savoir et les portes de la science.

Je remercie tout d'abord Monsieur Djedour. Mohamed de m'avoir proposé ce sujet de mémoire, et de l'attention et l' aide Qu'il a porté à mon travail.

Je remercie Monsieur. Aider Meziane pour avoir accepter de présider le jury

Je remercie Madame Djaballah Khedidja, Monsieur Ladjouz Salim et Monsieur Chaabane Djamel d'avoir accepté de faire partie du jury.

Sans oublier ; tous mes remerciements pour les enseignants du département de Probabilité & Statistique.

En fin, un grand merci au membres de ma famille qui m'ont soutenue et encouragés tout au long de ces années.

Sommaire

Introduction

I Introduction :

L'apprentissage et l'interpolation sont deux variantes extrêmes d'un même problème dont l'objet est de construire une fonction censée approcher raisonnablement une fonction inconnue dont on ne connaît qu'un certain nombre d'échantillons. Ces problèmes apparaissent dans des cadres varies qui vont de la résolution d'équations au dérivées partielles, en passant par la modélisation de formes en synthèse d'images, au domaine de l'apprentissage, des réseaux de neurones et du contrôle adaptatif.

Par rapport aux modèles linéaires ou à d'autres modèles statistiques classiques, les modèles connexionnistes offrent souvent des possibilités intéressantes, mais au prix d'un apprentissage plus difficile. Une des difficultés rencontrées consiste à fixer un certain nombre de "méta paramètres", par exemple liés à la complexité du modèle (nombre d'unités dans une couche cachée, etc.). Souvent, il n'existe pas d'autre choix que d'en tester plusieurs valeurs possibles et de choisir la meilleure, au sens d'un certain critère.

A cet effet, le présent mémoire illustre l'utilisation des réseaux de neurones a fonction de base radiale dans le contexte de la modélisation non linéaire.

Ce mémoire est composé de deux parties principales qui sont la partie théorique et la partie application.

Ces deux parties sont structurées comme suite :

II Partie théorique

Cette dernière est composée de trois chapitres :

Chapitre 1 : (Concerne la modélisation de données)

En premier lieu, ce chapitre donne la notion générale de la modélisation de données toute en précisant la signification des aspects suivants : Mesure, données, information, modèle et l'exploitation d'un modèle.

Ensuite la relation entre statistique et la modélisation de données a été mise en évidence en montrant l'importance de l'Estimation et l'inférence statistique ou la théorie des tests.

En fin de ce chapitre les difficultés de la modélisation de données ont été évoquées au travers de : la définition du problème, la qualité des Données, le choix de la technique de modélisation, le choix des variables qui entreront dans le modèle et la sélection de modèle.

Chapitre 2 : (Concerne les réseaux de neurones artificiels)

Ce chapitre est constitué de trois sections :

1er Section :

En premier lieu ; cette section illustre l'historique des réseaux de neurones artificiel, leurs succès, et donne aussi les notions de base de ces derniers et leurs les modes de fonctionnement (Mode parallèle, séquentiel et mixte).

Après ; la structure de base d'un réseau de neurone artificiel a été abordé.

En fin, on trouve la notion d'architectures des réseaux avec les structures d'interconnexions qui existent, ces dernières se devisent en deux groupes :

a- Les réseaux non bouclés : Les plus répandus des réseaux non bouclés sont : les réseaux multicouches et les réseaux à connexions locales.

b- les réseaux récurrents : Les plus répandus des réseaux récurrents sont : les Réseaux à connexions récurrentes et les Réseaux à connexions complète.

2eme Section :

Au début de cette deuxième section la définition de la propagation de l'activation a été donnée, et les deux phénomènes associer a cette dernière ont été définis, ces deux derniers sont : la Coopération et la Compétition. Le résultat d'interactions de ces deux phénomènes est traduit par les deux propriétés suivantes : Détection de traits et Mémoire associative.

Ensuite, vient la définition des deux modes d'apprentissage : Apprentissage supervisé et Apprentissage non supervisé, puis l'illustration des deux sources d'inspiration de règles d'apprentissage:

- Source biologique qui comprend la règle de Hebb.
- Source mathématique qui comprend : La règle de Widrow-hoff (règle delta) et La règle delta généralisé (l'algorithme de rétro propagation), avec l'existence d'autres règles d'apprentissage : L'apprentissage compétitif, L'apprentissage de Boltzmann et Apprentissage par renforcement.

La définition de la procédure de validation croisée a été abordée, et elle sera utilisée pour le test d'arrêt.

En fin, cette section illustre les différents modèles des réseaux de neurones les plus utilisés et leurs architectures avec les règles d'apprentissage :

Le Perceptron

Les réseaux multicouches avec l'aspect pratique de l'algorithme de rétro propagation du gradient, toute en passant par la définition du problème des minima locaux.

3em e Section :

Dans cette section on trouve la définition des réseaux à fonction de base radiale qui sont un cas particulier des réseaux multicouche, leurs caractéristiques est comme suite :

L'architecture du réseau : constituer d'une couche d'entrée, couche cachée et une couche de sortie qui comprend un seul neurone.

La règle d'apprentissage de ce type de réseau est composée de trois parties :

1- Estimation des centres des noyaux.

2- Estimation de la largeur des noyaux.

3- Estimation des coefficients synaptiques.

On trouve aussi la définition des réseaux de neurones à fonction de base radiale généralisée, et la règle d'apprentissage de ce dernier est basée sur la méthode de descente de gradient.

Ces deux derniers réseaux seront utilisés dans les applications qui seront illustré dans la partie application ci-dessous.

Chapitre 3 : (Concerne la complexité des algorithmes)

Au premier lieu, ce chapitre donne la notion d'algorithme et la définition du langage de description d'algorithme.

Apres on a donné une représentation de l'algorithme avec la définition des parties qui le constituent : L'en-tête, Les déclarations, Le corps et Les commentaires avec une définition général de type de donnés.

En suite on trouve la définition spatiale et temporelle de la complexité des algorithmes et la définition de la notation « grand O, Θ et Ω », avec les principaux types de complexité.

Puis les règles de calcul de la complexité ont été abordées et qui sont comme suite :

1-Complexité en pire des cas :

.Règle de la Somme.

.Règle du Produit.

.Grandes lignes de l'analyse d'un algorithme :

Règle-1 : Enchaînement

Règle-2 : Conditionnelle

Règle-3 : Itération (Tant Que)

Règle-4 : Fonctions et Procédures non récursives

2-Complexité en moyenne

III Partie Application

L'application est composée de deux parties principales : la première est une application basée sur le principe du mode d'apprentissage supervisé existant. La deuxième partie de l'application est basée sur la nouvelle approche du mode d'apprentissage supervisé.

L'application a été portée sur les réseaux de neurones à fonction de base radiale (RBF).

La particularité des réseaux RBF est que leur fonction noyau est décrite par deux paramètres : la position de son centre c_i, et la largeur σ_i du champ récepteur, et la fonction noyau la plus utilisée est la fonction gaussienne qui est de la forme : $\phi(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right)$

III.1 La 1ere partie de l'application :

III.1.1 1er Cas d'application:

Dans ce première cas d'application les paramètres de la fonction noyau sont fixe une fois pour toute dans la phase d'apprentissage par la méthode des treillis régulier qui consiste à placer les centres des fonctions noyau à égales distances (entre le minimum et le maximum de l'espace des données), et les valeurs des rayons sont fixées par la distance entre centres voisins, c'est-à-dire que dans ce premier cas l'apprentissage se fera uniquement par la modification des poids.

Pour effectuer le travail deux algorithmes ont été utilisés :

1- Algorithme basé sur la règle de Widdrow-Hoff.

2- Algorithme basé sur la règle de rétro propagation du gradient.

III.1.2 2eme Cas d'application:

Dans ce deuxième cas d'application les paramètres de la fonction noyau ne sont pas fixes durant la phase d'apprentissage, ils seront variables.

Donc l'apprentissage ne se fera non seulement sur la modification des poids mais aussi sur la modification des paramètres de la fonction noyau la position du centre et de la largeur du noyau.

Pour effectuer le travail, deux algorithmes ont été utilisés :

1- Algorithme basé sur : la règle de Widdrow-Hoff pour la modification des poids, et la méthode de descente de gradient pour la modification des paramètres (centres et rayons).

2- Algorithme basé sur : la règle de rétro propagation du gradient pour la modification des poids, et la méthode de descente de gradient pour la modification des paramètres (centres et rayons).

La 2eme partie de l'application :

Dans cette deuxième partie de l'application, on reprendra les 4 algorithmes qui ont étés décrits ci-dessus dans la première partie de l'application mais cette fois ci selon la nouvelle approche sur le principe du mode de l'apprentissage supervisé.

La nouvelle approche:

Dans un premier lieu, le principe de l'apprentissage supervisé est basé sur l'erreur déterminer en comparant la sortie calculé par le réseau a la sortie désiré, et l'erreur due a cette comparaison sera utilisé pour calculé la modification des poids de l'itération suivante (exemple : par la méthode rétro propagation du gradient ou la règle de Widdrow-Hoff) qui vise a réduire cette erreur au file d'itérations, c'est à dire utilisé l'erreur issue de l'itération **(n-1)** pour pouvoir calculé la modification des poids qui sera utiliser dans l'itération **(n)**.

La nouvelle approche est comme suite :

C'est au lieu d'utiliser l'erreur commise à l'itération **(n-1)** pour calculer la modification des poids qui sera utilisé dans l'itération **(n),** on utilisera l'erreur issue de l'itération **(n-2)** pour calculer la mise à jour des poids pour l'itération **(n)**.

IV La complexité des algorithmes :

Après chaque application d'un algorithme on détermine la complexité temporelle de ce dernier qui va permettre d'évaluer quel va être la rapidité d'exécution de cet algorithme.

Cette complexité sera déterminée par le nombre d'opérations élémentaires (affectations, comparaisons, opérations arithmétiques) effectué par l'algorithme. Ce nombre s'exprime en fonction de la taille n des données.

En fin, la comparaison se portera sur les résultats issus de l'application des différents algorithmes avec leurs complexités, et aussi la comparaison des résultats qui ont été obtenus dans les deux modes d'apprentissages supervisés.

Modélisation de données

V La modélisation de données :

La Modélisation de Données est l'art d'extraire des informations utiles d'un ensemble de données obtenues par des mesures, et de condenser cette information dans un modèle exploitable. Tous ces termes méritent d'être expliqués. [18]

V.1 Mesures

Ce terme, issu du monde de la technique, doit être pris dans un sens général. Dans son sens le plus simple, "Mesurer", c'est représenter une grandeur d'un objet par un nombre, comme dans l'expression "Mesurer la longueur d'une barre de fer". Mais le résultat de la mesure peut prendre d'autres formes, comme un rang ("Un électeur très sûr de son choix"), ou l'appartenance à un groupe ("Cette personne est un artisan"). [18]

V.2 Données

Le plus souvent, plusieurs caractéristiques d'un même objet sont mesurées de façon à augmenter la quantité d'information disponible sur cet objet. Traditionnellement, les résultats de ces mesures sont regroupés sur une même ligne. [18]

Les caractéristiques mesurées sont souvent appelées "variables", et les objets sur lesquels ces caractéristiques sont mesurées des "individus".

V.3 Information

La collecte de ces données est effectuée dans le but de pouvoir répondre à certaines questions générales sur la population des individus, ou sur certains individus particuliers.

Bien qu'il existe une infinité de questions susceptibles de recevoir des réponses par le biais d'un tableau de données, nous ne retiendrons que la distinction entre deux catégories de questions [18] :

Celles qui cherchent à décrire, de façon condensée, la population dans ses grande Lignes (**Modélisation Descriptive**).

Celles qui cherchent à mettre en évidence des **liens** entre les différentes variables du tableau. Par exemple, l'analyse du tableau pourrait mettre en évidence une relation du type Tension = *f* (Sexe, Âge, Poids). [18]

Lorsqu'ils sont découverts, ces liens ont deux utilités [18]:

- D'une part, ils mettent en évidence une relation possiblement causale entre grandeurs. Si la nature causale de cette relation est confirmée, il devient alors possible, en modifiant les causes, d'orienter les effets dans des directions favorables.

- D'autre part, le lien découvert se traduit par une équation qui permet, si se présente un individu dont la tension artérielle n'a pas été mesurée, de **prédire** la valeur de cette grandeur grâce à cette équation.

C'est pour cette dernière raison que ce type de modélisation s'appelle la **Modélisation Prédictive**.

V.4 Modèle

Le Modèle est une représentation mathématique condensée et à valeur opérationnelle du grand tableau de données initial. [18]

V.5 Exploitable

La Modélisation Prédictive fournit un exemple d'exploitation directe d'un modèle pour la prédiction de grandeurs non mesurées sur de nouveaux individus. Ne figurant pas dans l'échantillon. [18]

VI Modélisation des données et statistique :

Il est souvent possible, moyennant certaines hypothèses, de juger de la crédibilité du modèle (construit sur l'échantillon) en tant que représentant des propriétés de la population totale. C'est le rôle de la Statistique. [18]

La Statistique comporte deux branches principales, l'Estimation et l'Inférence (ou Théorie de Tests).

VI.1 L'Estimation :

La Théorie de l'Estimation permet d'affirmer que non seulement la moyenne de l'échantillon est une bonne estimation de la moyenne de la population (Estimation ponctuelle), mais qu'elle est également la meilleure possible (dans un sens très précis). Aucune autre grandeur calculée sur l'échantillon ne pourrait recevoir une plus grande confiance comme estimation de la moyenne de la population. [18]

Moyennant certaines hypothèses, il est même possible de quantifier cette confiance (Estimation par Intervalle de Confiance).

Le rôle de l'Estimation est donc [18] :

- D'identifier les grandeurs calculables sur l'échantillon les plus représentatives des grandeurs réelles de la population. Les valeurs obtenues sont alors appelées des estimations des grandeurs (inaccessibles) relatives à la population totale.
- Et d'évaluer l'incertitude qui pèse sur la qualité de ces estimations.

VI.2 L'inférence ou Théorie des Tests :

L'inférence statistique est un ensemble de méthodes permettant de tirer des conclusions fiables (hypothèse concernant la population) à partir de données d'échantillons statistiques. [18]

VII Les difficultés de la modélisation de données

Les principales difficultés de la Modélisation sont [18] :

VII.1 La définition du problème :

La collecte et la mise en forme des données, la construction, la validation et l'interprétation d'un modèle sont des tâches longues et coûteuses. Le temps passé à définir clairement l'objectif de l'étude, les critères permettant de savoir si celui-ci a ou non été atteint, l'identification de la nature et du volume des données nécessaires à la construction d'un modèle adéquat, est toujours du temps gagné, jamais du temps perdu. [18]

VII.2 La qualité des Données :

Celles-ci ont la fâcheuse habitude d'être rares, chères, mal formatées, incomplètes, non synchronisées, entachées d'erreurs et biaisées. Les données sont pourtant le carburant de la modélisation, et doivent recevoir toute l'attention requise pour atteindre le niveau de qualité requis par l'application, sous peine de rendre vains tous les efforts de modélisation. [18]

VII.3 Le choix de la technique de modélisation (laissé à l'appréciation de l'analyste) :

Le développement considérable de la Modélisation de Données met à la disposition du praticien un grand nombre de techniques différentes permettant, théoriquement, d'atteindre un objectif donné. Mais chaque technique a ses avantages et ses inconvénients, et le choix de la technique la plus appropriée est une des conditions essentielles du succès de la modélisation.

Malheureusement, en dehors de considérations d'ordre général, seul une longue pratique permet d'éviter les choix malheureux. [18]

VII.4 Le choix des variables qui entreront dans le modèle (et donc de celles qui ne seront pas prises en compte) :

Pour des raisons fondamentales de Statistique (et souvent ignorées des praticiens), il est indispensable de procéder à une sélection rigoureuse (et difficile) des variables parmi les variables disponibles dans le tableau de données, voire de procéder à une Réduction de Dimensionnalité avant toute modélisation. Si trop de variables sont incluses dans le modèle,

celui-ci devient exagérément sensible à de petites modifications de l'échantillon, et le modèle obtenu est alors peu crédible (compromis biais variance). [18]

Des objectifs différents, ou même des techniques différentes mais ayant le même objectif, conduiront à des ensembles optimaux de variables différents.

VII.5 Sélection de modèle :

Tout modèle contient des paramètres (même les modèles dits "non paramétriques"). Les valeurs de ces paramètres sont :

Soit calculées par un algorithme d' "apprentissage", en général destiné à minimiser une quantité calculée à partir des données,

Soit définies arbitrairement (pensez au "Nombre de cases" d'un histogramme).
Dans les deux cas, le choix du nombre de paramètres du modèle est laissé à l'analyste. Même en Régression Linéaire Multiple, le nombre de paramètres est imposé par le nombre de prédicateurs retenus dans le modèle. [18]

On se convainc facilement qu'une augmentation du nombre de paramètres du modèle augmente sa souplesse, et donc sa capacité à rendre compte des données d'apprentissage. Mais on découvre également qu'à partir d'un certain point, une augmentation du nombre de paramètres conduit à une dégradation des performances du modèle sur les données nouvelles, les seules performances importantes. Cette situation rappelle celle décrite au paragraphe précédent (Sélection de variables), et relève en fait de la même problématique : le "compromis biais variance". [18]

Donc, le modèle une fois construit, il convient d'estimer ses performances réelles sur des données nouvelles (et non pas sur les données qui ont servi à le construire). Ceci se fait en soumettant plusieurs modèles candidats à une série de tests de validation. Le modèle finalement retenu est celui qui aura obtenu les meilleurs résultats lors de ces tests (et ce ne sera vraisemblablement pas celui qui aura obtenu les meilleurs résultats sur les données d'apprentissage). [18]

Réseaux de neurones artificiels

VIII Introduction :

Copier le cerveau humain restera pour longtemps une ambition exagérée, mais vouloir s'inspirer des architectures et des fonctions du système nerveux n'est pas un rêve inaccessible, en effet le cerveau a fait l'objet de nombreuses études dans le domaine de la neurobiologie, qui cherche à expliquer son comportement et à le modéliser par des outils mathématiques et algorithmiques, l'un des fruits de ces études a été l'apparition des réseaux de neurones qui sont des modèles très puissants. Ils ont pour origine un modèle du *neurone biologique*, dont ils ne retiennent qu'une vision simplifiée.

VIII.1 Historique :

Le fait d'utiliser des outils mathématiques ou de la physique pour modéliser le fonctionnement des systèmes nerveux ne date pas d'aujourd'hui.

Sous le terme de réseaux de neurones, on regroupe aujourd'hui un certain nombre de modèles dont l'intention est d'imiter certaines des fonctions du cerveau humain en reproduisant certaines de ces structures de base. Historiquement, les origines de cette discipline sont très diversifiées. [19, 5 et 2]

1890 : W. James, Introduit le concept de mémoire associative.

1943 : Mc Culloch et Pitts ont modélisé la cellule nerveuse ou neurone par un automate à seuil et lui ont donné le nom de neurone formel.

1949 : D. Hebb propose la loi de modification de propriétés de connexions entre neurones.

Les premiers succès :

1957 : F. Rosenblatt décrivit le premier modèle opérationnel de réseaux de neurones, mettant en œuvre les idées de Hebb, Mc Culloch et Pitts.

1960 : B. Widrow, un automaticien, développe le modèle Adaline (Adaptive Linear Elément).

L'ombre :

1969 : Deux mathématiciens, Minsky et Papert, démontrèrent les limites théoriques du perceptron pour les modèles non linéaires (le perceptron aide à résoudre seulement les problèmes dits linéairement séparables).

Les réseaux de neurones ont été délaissés pendant presque 20 ans, toutes les recherches ne sont, bien sur, pas interrompues. De grands noms travaillent durant cette période telle que : S. Grossberg, T. Kohonen, ….

Le renouveau :

1982 : avec l'approche de Hopfield et l'algorithme de retro-propagation les différents problèmes liés à l'approche symbolique, sont absents dans les réseaux de neurones.

La levée des limitations :

1983 : La machine de Boltzmann est le premier modèle connu apte à traiter de manière satisfaisante les limitations recensées dans le cas du Perceptron.

1985 : La rétro propagation de gradient apparaît. C'est un algorithme d'apprentissage adapté aux réseaux de neurones multicouches.

IX Notions de base sur les réseaux de neurones :

IX.1 Définition:

Les réseaux de neurones artificiels sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle. Chaque processeur élémentaire calcul une sortie unique sur la base des informations qu'il reçoit. Toute structure hiérarchique de réseaux est évidemment un réseau. [19]

IX.2 Les modes de fonctionnement :

La première hypothèse faite est que le temps est discrétisé en pas d'horloge. Les trois principaux modes de fonctionnement sont :

IX.2.1 Mode parallèle :

À chaque top d'horloge, tous les neurones calculent leurs nouvelles activations, leurs sorties et les transmettent aux neurones auxquels ils sont connectés en tant que neurone émetteur. Le calcul se fait en fonction de leurs entrées au top d'horloge précèdent.

IX.2.2 Mode séquentiel :

À chaque top d'horloge, un seul neurone calcule sa fonction d'activation, sa sortie et la transmette aux neurones qui lui sont connectés. Ce mode peut se faire suivant une liste cyclique prédéfinie ou suivant un choix aléatoire du neurone qui exécutera ce calcul.

IX.2.3 Mode mixte :

C'est une combinaison des deux modes séquentiel et parallèle.

X Modélisations :

X.1 Structure de base d'un réseau de neurone artificiel :

Les réseaux connexionnistes peuvent être définis comme un ensemble d'unités (ou automate), réalisant des calculs élémentaires, structurés en couches successives capables d'échanger des informations au moyen de connexions qui les relient.

Chaque unité effectue un traitement local de l'information. Il existe plusieurs types d'unités :

Des unités d'entrée : auxquelles sont transmises les données a traiter, en provenance des sources externes au réseau. Ces unités n'effectuent aucun traitement sur les données en entrée.

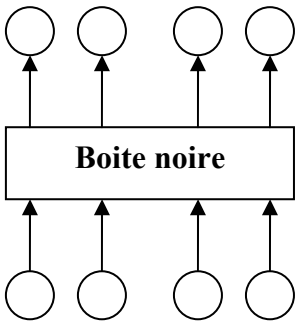
Des unités cachées : dont les entrées et les sorties sont reliées aux autres unités du réseau, et donc non visibles pour des systèmes extérieurs. Ces unités cachées servent à traiter, de façon interne au système, la structure des formes présentées à l'entrée.

Des unités de sortie : qui contiennent l'information traitée et utilisable par d'autres systèmes connectés au réseau.

Chaque unité est considérée comme un automate possédant plusieurs entrées X_i et une sortie S. cette dernière est une fonction des entrées appelées fonction de transfert ou fonction d'activation.

La valeur du poids d'une connexion entre deux unités reflet la relation entre ces deux unités. Si cette valeur est positive, la connexion est dite excitatrice, sinon elle est dite inhibitrice. Il existe plusieurs façons de relier les unités d'un réseau. La plus simple est la connectivité totale : chaque unité est connectée à toutes les unités de la couche suivante.

La figure ci-dessous montre que les neurones d'entrée sont connectés aux neurones de sortie à travers une « boite noire ». Seul le modèle exact du réseau, qui dépend de son application, déterminera la nature de cette boite noire. Cette disposition est commune à plusieurs taches, mais elle n'est pas la seule. Certains modèles connectent directement les entrées aux sorties. Pour d'autres modèles certains neurones d'une même couche sont connectés entre eux. [25]



X.2 Architecture des réseaux :

La topologie des réseaux de neurones peut être très variée. On peut concevoir plusieurs types de réseaux seulement en modifiant les règles de connexions**.**

Un réseau de neurone peut être vu comme des graphes orientés dans lesquels les neurones sont les sommets, et les arcs (pondérés) sont les connexions entre sortie de neurones et entrées. Les réseaux de neurones peuvent se distinguer par : l'architecture et le mode d'apprentissage.

L'architecture du réseau de neurone décrit la manière dont sont interconnectées les cellules. Elle est spécifiée par (Ces paramètres sont généralement fixés a priori):

- Le nombre de cellules.
- La nature des cellules (leurs fonctions de transfert sont en général, identique pour toutes les cellules).
- La relation entre le réseau et l'extérieur.
- Le graphe d'interconnexion des cellules.

XI Structures d'interconnexion :

En fonction du type de connexions (architecture), les réseaux de neurones artificiels sont regroupés en deux catégories : Les réseaux non bouclés et les réseaux récurrents. Nous allons définir les différentes architectures des réseaux, du moins les plus connues :

XI.1 Réseau non bouclés :

Les plus répandus des réseaux non bouclés sont :

XI.1.1 Les réseaux multicouches : (figure 5)

Les neurones sont organisés en couche, avec des connexions inter couche mais pas de connexions intra couche. Chaque couche reçoit des signaux de la couche précédente et transmette le résultat de ses traitements à la couche suivante. [19]

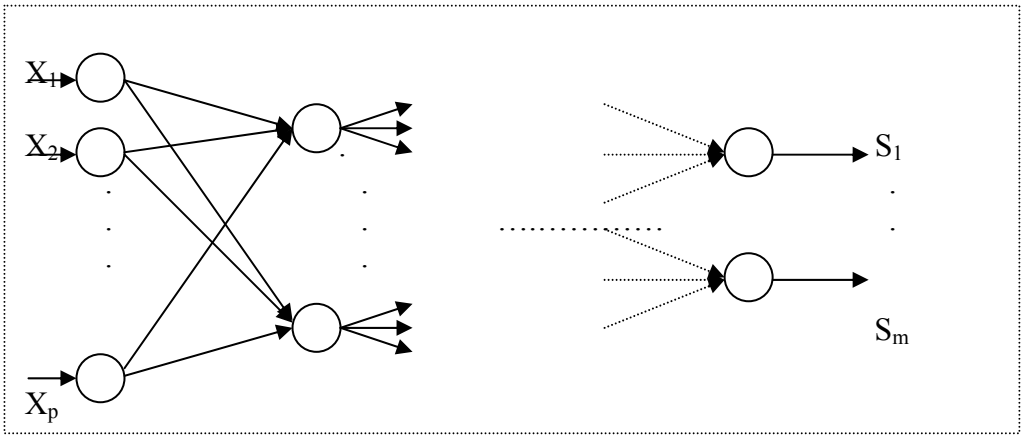
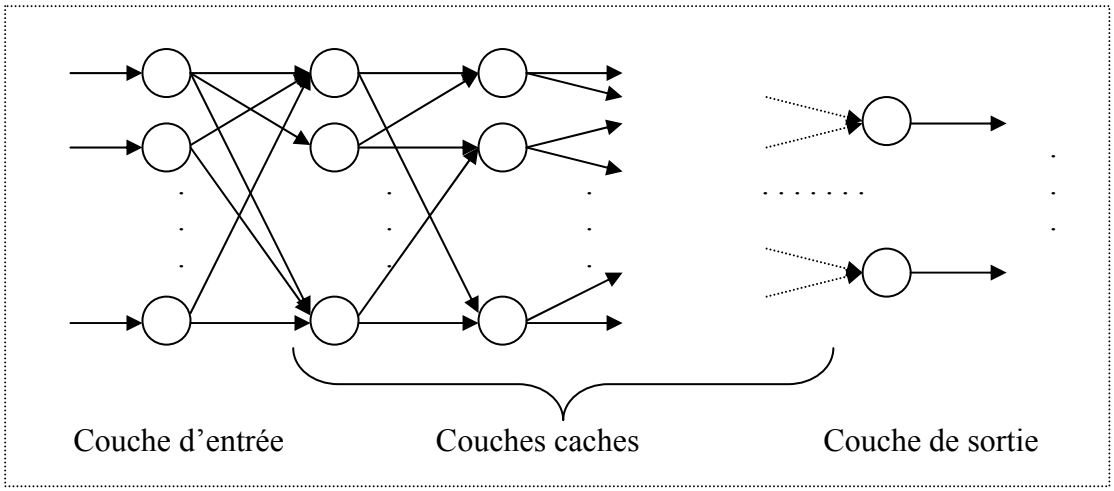


Figure 6 : Architecture d'un réseau multicouche a n niveaux.

XI.1.2 Réseau à connexions locales : (Figure 6)

Il s'agit d'une structure multicouche, mais qui conserve une certaine topologie. Chaque neurone entretien des relations avec un membre réduit et localise de neurones de la couche avale. Les connexions sont donc moins nombreuses que dans le cas d'un réseau multicouches classique. [19]



XI.2 Réseaux récurrents :

L'architecture des réseaux récurrents peut aller d'une connectivité totale (où tous les neurones sont reliés entre eux), à une connectivité locale (où les neurones ne sont reliés qu'à leurs plus proches voisins). La particularité de ces réseaux est qu'ils sont justement récurrents : ce sont des graphes avec circuits (du fait de la présence d'arcs de rétroaction). Un neurone peut posséder un retour sur lui-même. [5, 2 et19]

XI.2.1 Réseaux à connexions récurrentes : (Figure 7)

Les connexions récurrentes ramènent l'information en arrière par rapport au sens de propagation défini dans un réseau multicouche. Ces connexions sont le plus souvent locales. [19].

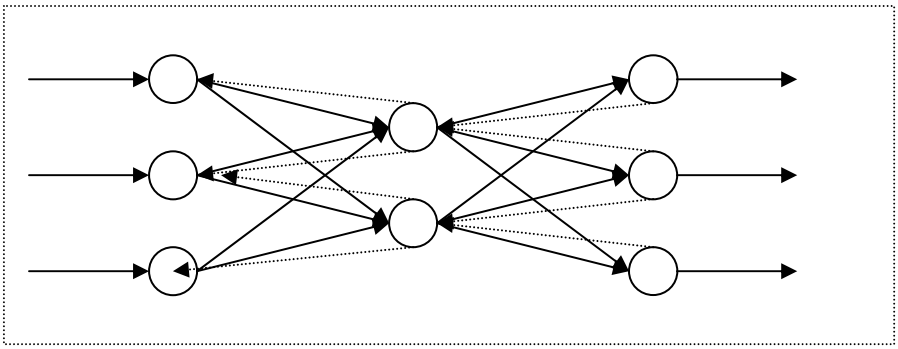


Figure 7

XI.2.2 Réseau à connexion complète : (Figure 7.1)

C'est la structure d'interconnexion la plus générale, chaque neurone est connecté à tous les neurones du réseau (et à lui-même). [19]

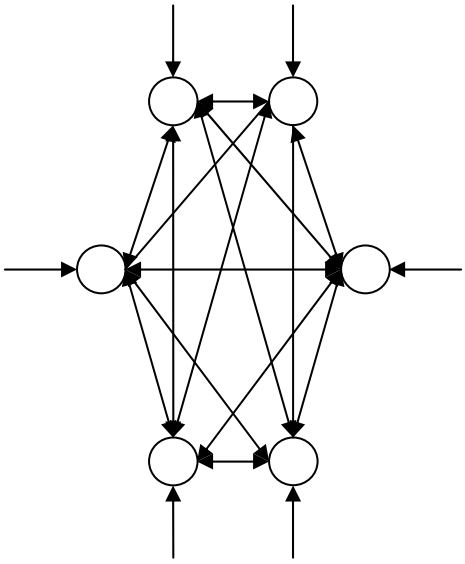


Figure 7.1

<u>Remarque</u> :

Les réseaux de type non bouclés sont statiques dans le sens où ils ne produisent qu'une seule réponse en fonction des entrées fournies. Ils sont sans mémoire, puisque les réponses fournies ne prennent pas en compte l'état antérieur du réseau. Au contraire, les réseaux récurrents sont dynamiques et dépendent de l'état antérieur.

Ces différentes architectures, nécessitent différent types d'algorithmes d'apprentissage.

XIILa propagation de l'activation :

Nous avons vu que le comportement d'un neurone dépendait essentiellement du choix de sa fonction d'activation. L'activation calculée par cette fonction affecte, par l'entremise des connexions, d'autres neurones du réseau ; on dit que l'activation se propage d'un neurone à l'autre. C'est la propagation d'activation au niveau du réseau entier qui est la source de son comportement collectif. [2]

On appelle cycle d'activation le calcul d'activation, en temps discret, de tous les neurones du réseau. Il est important de remarquer que la séquence d'états traversés le réseau à partir d'un état initial donné (sa trajectoire) peut être affectée par l'ordre dans lequel on choisit les neurones durant le cycle d'activation [2] :

- Dans une ***propagation synchrone,*** tous les neurones sont mis à jour en même temps. Ceci implique que chaque neurone calcule son activation à partir de l'activation que les autres neurones affichaient au pas de temps précédent. [2]
- En revanche, dans une ***propagation d'activation asynchrone,*** les neurones sont mis à jour un par un. Un neurone donné calcule son activation à partir d'activations qui peuvent ou non avoir été fraîchement recalculées. Il est clair, dans ces conditions, que l'ordre de mise à jour des neurones peut avoir un effet important sur le comportement du réseau. Une propagation est *séquentielle* quand la mise à jour s'effectue selon un ordre fixe, ou aléatoire quand les neurones sont choisis dans un ordre aléatoire. [2]

XIII Phénomènes connus de la propagation d'activation :

Parmi les phénomènes collectifs qui s'observent dans un réseau neuromimétique dés lors que l'on permet à l'activité de se propager parmi les neurones, deux sont d'une importance particulière [2] :

XIII.1 Coopération :

Dans un réseau, deux neurones reliés par des liens excitateurs s'entre stimulent. Dans un tel cas, une entrée, même faible, peut servir d'amorce et produire dans ces neurones un effet de rétroaction croissant qui se termine en leur pleine co-activation. De cette façon, un neurone peut entraîner l'autre par son activation. [2]

XIII.2 Compétition :

Ce phénomène est le contraire de la coopération. Ici, les neurones sont reliés par des liens inhibiteurs : leur activité est antagoniste. [2]

Dans un réseau neuromimétique, coopération et compétition agissent de concert pour donner des propriétés d'interaction utiles ; nous citons trois d'entre elles [2] :

XIII.2.1 Détection de traits :

Une forme particulière de compétition, nommée « on-center, off-surround » est particulièrement utile et apparaît notamment dans le système visuel des mammifères. Dans cette structure, le neurone (et éventuellement ses proches) est excité, tandis que les neurones plus distants sont inhibes. Cette connectivité peut provenir des neurones même, ou dépendre de leurs entrées. [2]

XIII.2.2 Mémoire associative :

Plusieurs réseaux de neurones sont capables de reconstituer une information complète à partir d'une trace dégradée (auto-assocation), ou de retrouver une information à partir d'indices associes (hetero-association). Cette faculté dépend à la fois de la coopération entre les neurones codant des informations associées, et de la compétition entre les neurones codant des informations distincts. [2]

Il est clair que plus la mémoire est chargée, plus il est difficile de maintenir séparées les différentes informations qui y sont entreposées. L'interaction parasite entre informations pose des contraintes très sévères non seulement à la mémoire, mais aussi dans le degré de similarité que l'on peut permettre entre les informations. Dans certains modèles, ce problème est ci délicat qu'il souffre « d'oubli catastrophique » ; si l'on surcharge ce réseau, il perd subitement sa faculté de reconstituer toutes les informations qu'il a retenues. [2]

XIII.3 Satisfaction de contraintes :

La satisfaction d'un réseau d'un ensemble de contraintes est un problème que l'on sait être très difficile ; sa difficulté émane du fait que les contraintes sont en général interdépendantes. [2]

Pour amorcer un réseau, on distribue une activation aléatoire aux neurones. Ensuite, il suffit de propager l'activation ce que l'état du réseau cesse d'évoluer. Le réseau se trouve alors dans un état où chaque neurone satisfait au mieux les contraintes dont il a localement connaissance, et qui est interprété comme une solution possible au problème.

Ainsi, deux phénomènes fondamentaux, la coopération et la compétition, se déclinent de plusieurs façons pour produire des comportements utiles dans les réseaux neuromimétique. [2]

XIV Les modes d'apprentissage :

La capacité à apprendre est une des caractéristiques fondamentales de l'intelligence, Bien qu'une définition précise de ce qu'est l'apprentissage soit difficile à fournir, un processus d'apprentissage dans le cadre des réseaux de neurones artificiels (RNA) peut être vu comme le problème d'initier le réseau à évoluer d'un état initial vers un état final. Cette évolution se traduit par la modification des poids des connexions entre les neurones du réseau. Ceci permet de trouver un réseau optimal pour résoudre le problème considéré ou encore de généraliser une fonction décrite par un ensemble de couples entrées-sorties appelés exemples. En général, un réseau de neurone doit apprendre les poids de ces connexions à partir d'un ensemble de données d'entraînement. [22, 5, 2]

L'efficacité des réseaux est améliorée en modifiant de façon itérative les poids du réseau. La capacité des réseaux de neurones artificielle à apprendre automatiquement à partir d'exemple les rend extrêmement attractifs.

Les RNA apprennent les règles sous-jacents (telle que la relation entre entrées et sorties) a partir d'un ensemble représentatif d'exemples, c'est là l'un des avantages majeurs de cette approche. Pour apprendre/comprendre ou définir un processus d'apprentissage, il est nécessaire de modéliser l'environnement dans lequel opéra le réseau, c'est à dire qu'il faut savoir quel type d'information sera fournie au réseau. Cette modélisation est le paradigme d'apprentissage. Ensuite, il faut comprendre comment le réseau met à jour les poids de ses connexions, c'est-à-dire quelles sont les règles d'apprentissage qui gouvernent la mise à jour. Un algorithme d'apprentissage est la procédure dans la quelle les règles d'apprentissage sont utilisées en vue de l'ajustement des poids. [22, 5, 2]

On Peut considérer qu'il existe deux principaux paradigmes d'apprentissage : supervisé et non supervisé :

XIV.1 Apprentissage supervisé :

Comme son nom l'indique, c'est un apprentissage guidé et suivi par un superviseur ou professeur, qui détermine la famille ou la classe de chaque échantillon donné en entrée. Ce mode d'apprentissage date des années 50 et se déroule de la manière suivante **:**

Un exemple est présenté aux neurones d'entrée, puis l'activation propagée à travers le réseau, la réponse des neurones de sortie au passage de l'exemple est alors comparée aux valeurs désirées. Ceci détermine l'erreur du réseau pour l'exemple donné. Il s'agit alors de répartir cette erreur à chaque poids du réseau en fonction de la part qu'il a joué dans la

production de la réponse erronée. On procède alors à une modification des poids qui vise à réduire l'erreur ainsi calculée. [22, 5, 2]

XIV.2 Apprentissage non supervisé :

L'apprentissage non supervisé est également appelé apprentissage sans professeur, contrairement à l'apprentissage précèdent, aucune information sur la sortie attendue n'est fournie au réseau. Le réseau évolue librement jusqu'à un point d'équilibre et les calculs sont moins complexes en comparaison avec l'apprentissage supervisé.

Ce mode d'apprentissage consiste à fournir à un réseau autonome une quantité suffisante d'exemples pour que celui-ci en dégage les irrégularités spontanément. Cet apprentissage entre dans le cadre des méthodes d'apprentissage automatique qui sont fréquemment utilisées dans la compression des données. [22]

XV Les règles d'apprentissage :

Il existe deux grandes familles de règles qui différent par leur source d'inspiration : la première source est biologique, la deuxième est mathématique.

XV.1 Source biologique :

L'évolution du système nerveux est due à l'interaction entre l'environnement extérieur et le programme génétique. Cette évolution se traduit par l'évolution des synapses qui soient se dégénèrent soient se stabilisent. Cette idée d'un mécanisme synaptique de couplage avait était proposée par Hebb en 1949. [2, 5, 19, 21 et 23]

Règle de Hebb :

Le point fondamental de l'observation de Hebb sur ses résultats expérimentaux est que le renforcement synaptique intervient lorsqu'il y a activation conjointe du neurone présynaptique (cause) et du neurone post-synaptique (effet).

La règle de Hebb prévoit exclusivement le renforcement des efficacités synaptiques, c'est-à-dire que le poids de la synapse ne peut qu'augmenter, ce qui conduirait en l'absence de phénomènes limitants à la saturation des réseaux. [2, 5, 19, 21 et 23]

La règle de Hebb a été complétée par les travaux de Rauschecker et Singer qui ont proposés les trois règles suivantes [2, 5, 19, 21 et 23]:

- L'efficacité des synapses excitatrices augmente à chaque fois que les éléments pré- et post-synaptiques sont actifs simultanément.
- L'efficacité des synapses excitatrices décroît si le neurone post-synaptique est actif, tandis que le neurone présynaptique est silencieux.

- L'efficacité des synapses décroît lentement indépendamment de l'activité présynaptique si le neurone post-synaptique est au repos.

La variation de l'efficacité synaptique (poids synaptique) est donnée par l'expression mathématique proposée par Sutton en 1981. [2, 5, 19, 21 et 23]

$$W_{ij}(t+1) = W_{ij}(t) + \mu \ A_i A_j$$

A_i : est l'activation du neurone i.

A_j : est l'activation du neurone j.

μ : est le paramètre d'intensité d'apprentissage $(\mu \succ 0)$.

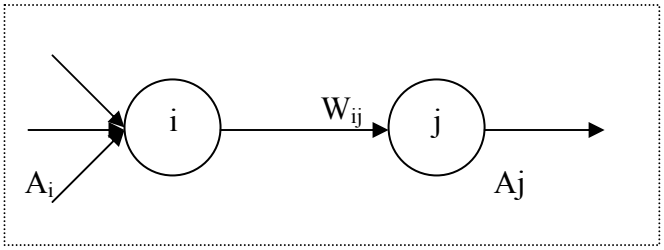


Figure 8 : Connexion renforcée

XV.2 Source mathématique :

Partant d'un réseau de neurones formels, on cherche à imposer au réseau une certaine fonctionnalité, par exemple reconnaître des mots.

Le réseau est considéré comme une fonction de transfert des entrées qui lui sont présentées. Il reste à calculer les bons paramètres de cette fonction de transfert. Cette démarche a abouti à des algorithmes comme celui du perceptron, la règle delta pour les réseaux monocouche et la règle delta généralisée (rétro propagation) pour les réseaux multicouches. [21]

XV.2.1 La règle de Widrow-hoff (règle delta) :

La loi conçue par les deux chercheurs Bernard Widrow et Marcian hoff s'applique uniquement aux réseaux à une couche cachée. L'apprentissage se fait à partir d'une paire d'entraînement, formée d'un vecteur d'entrée x et un vecteur de sortie S désirée. [24, 5 et 19]

Plus précisément le vecteur d'entrée x correspond aux valeurs des connexions d'entrées du réseau. Quand au vecteur de sortie S, il correspond aux valeurs des sorties du réseau que l'on désire obtenir à partir de x. Donc, si S_{di} est la valeur de la sortie désirée du neurone i, où le neurone i est un neurone de sortie du réseau. Les vecteurs d'entrée du réseau doivent être linéairement indépendants. L'apprentissage par la règle de delta comporte 4 phases :

- Assigner des poids non nuls aux connexions avant de commencer l'apprentissage.

- Prendre au hasard une des paires d'entraînement possibles, propager x dans le réseau pour obtenir la valeur de sortie calculée de chaque neurone, ces valeurs sont mises dans y.
- Pour chaque neurone de sortie i, générer le signale d'erreur δ_i en comparant la sortie calculée y_i à la sortie espérée S_{di} ;
- Modifier les poids w_{ji} des connexions en fonction des δ_i .

Répéter les 3 dernières phases jusqu'à ce que le réseau répond de façon satisfaisante aux différentes valeurs d'entrée.

δ_i Est appelée aussi delta, elle correspond à la différence entre y_i et S_i . L'apprentissage se fait en modifiant chaque poids associé à une connexion qui relie un neurone j à un neurone i, de façon à réduire la différence entre y_i et S_i . La modification du poids se fait comme suit [24, 5 et 19]:

$$\delta_{ji} = \eta\left(y_{pi} - S_{pi}\right)x_{pj}$$

Où η : est le taux d'apprentissage. Ce taux correspond au temps nécessaire pour apprendre. Plus ce taux augmente, plus les risques d'un comportement oscillatoire augmentent.

δ_{ji} **:** La différence entre w_{ji} à l'instant (t+1) et w_{ji} à l'instant (t).

y_{pi} **:** La sortie réelle du neurone i, i est le neurone de sortie et p est l'exemple courant.

S_{pi} **:** Sortie désirée du neurone i.

x_{pj} **:** Entrée du réseau.

XV.2.2 La règle delta généralisé (l'algorithme de rétropropagation) :

Cette règle que l'on distingue couramment par « back propagation » est une généralisation de la règle de Widrow-hoff pour un réseau multicouche.

L'idée simple qui est à la base de cet algorithme, et qui permet de lever la difficulté du « credit-assignement problem » est l'utilisation d'une fonction dérivable (fonction sigmoïde) en remplaçant la fonction à seuil utilisée dans les neurones linéaires à seuil. Mathématiquement, cet algorithme utilise simplement les règles de dérivation composées et ne présente aucune difficulté particulière.

L'apprentissage de BPN (Back Propagation Network) fonctionne par le même principe que les règles simples du perceptron ou de Widrow-Hoff. On dispose d'un ensemble

d'exemples qui sont des couples (entrées, sorties désirées). À chaque étape un exemple est présenté à l'entrée du réseau. Un signal est propagé de proche en proche à travers chaque couche supérieure à la couche d'entrée jusqu'à ce qu'une sortie soit générée.

Cette sortie est alors comparée à la sortie désirée et un signal d'erreur (somme quadratique des erreurs sur chaque cellule de sortie) est calculé. Ce signal d'erreur est ensuite rétro propagé de la couche de sortie vers chaque cellule de la couche intermédiaire qui contribue directement à la sortie. Ce processus est répété, couche par couche, jusqu'à ce que chaque cellule du réseau ait reçu le signal d'erreur.

En parallèle, les poids des connexions sont alors mis à jour pour chaque cellule, et cela pour permettre au réseau de converger vers un état qui permettra à tous les modèles d'apprentissage d'être codés.

Ce processus est répété, en présentant successivement chaque exemple. Si pour tous les exemples, l'erreur est inférieure à un seuil choisi, on dit alors que le réseau a convergé.
Le sens de ce processus est que durant l'apprentissage du réseau, les cellules des couches intermédiaires s'organisent de manière à ce qu'elles apprennent à reconnaître les différentes caractéristiques de tout l'espace d'entrée.

Ce qui est important est le fait que le réseau doit trouver une représentation interne qui lui permettra de générer la sortie désirée quand on lui présente les entrées apprises.

XV.3 Autres règles d'apprentissage :

XV.3.1 L'apprentissage compétitif :

Où les neurones de sortie du réseau ‘‘décident’’ lequel d'entre eux sera actif. Il n'existe donc qu'un seul neurone de sortie actif à chaque cycle. [23]

XV.3.2 L'apprentissage de Boltzmann :

Qui repose sur un algorithme stochastique issu de la thermodynamique et de la théorie de l'information (car on va chercher un maximum de vraisemblance). [5]

XV.3.3 Apprentissage par renforcement :

Lorsque l'information du gradient n'est pas disponible, il faut recourir à des mécanismes d'apprentissage par renforcement. Ces mécanismes fonctionnent sur la base d'une exploration judicieusement organisée de toutes les actions de commandes possibles et déduisent les variations correspondantes des poids. En d'autres termes, l'apprentissage nécessite seulement une évaluation qualitative de la performance désirée, qui est

généralement spécifiée sous forme d'une fonction de coût à minimiser. Cette fonction est directement liée à la tache que doit accomplir le réseau de neurones dans l'environnement où il est intégré. [23, 19]

XVI La procédure de validation croisée (Test d'arrêt) :

La validation croisée est une méthode d'entraînement fort utile qui permet d'estimer les performances en généralisation d'un réseau après apprentissage. Elle consiste d'abord à mesurer l'évolution des performances du réseau sur son corpus d'apprentissage. En général, ces performances s'améliorent constamment au cours de l'apprentissage. [2]

Cependant, cette mesure ne fournit pas à elle seule une évaluation objective des performances réelles du réseau. En effet, il est possible que le réseau apprenne à traiter spécifiquement les informations contenues dans le corpus d'apprentissage, au détriment de ses performances sur le problème général. Ce phénomène survient souvent en fin d'apprentissage, et se nomme surapprentissage, ou apprentissage par coeur. [2]

Pour éviter ce problème, l'on mesure l'évolution des performances du réseau sur un second corpus, distinct du corpus d'apprentissage (le corpus test, ou corpus de généralisation). Dans la mesure où le réseau apprend effectivement à traiter le problème, ses performances s'amélioreront à la fois sur le corpus d'apprentissage et sur le corpus test. [2]

S'il commence à apprendre par cœur, ses performances en test se dégraderont. Ainsi, le moment judicieux pour interrompre l'apprentissage est celui où les performances en test sont optimales :(figure9). Le point « A » indique le moment où il serait judicieux d'interrompre l'apprentissage. [2]

Notons que si le corpus test n'a pas été appris par le réseau, il a néanmoins été employé pour interrompre l'entraînement. [2]

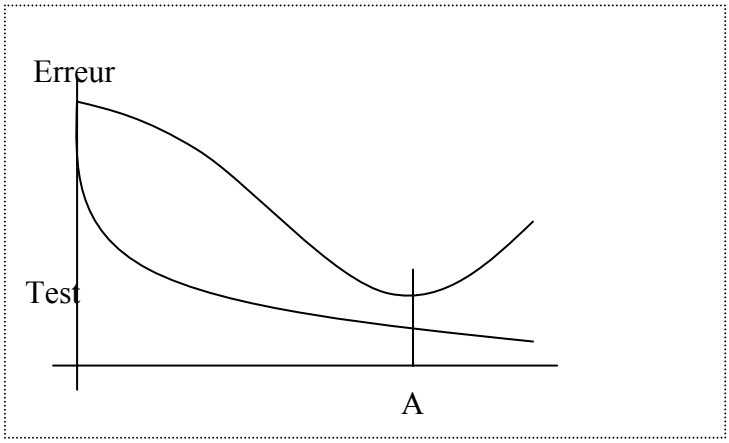


Figure : 9

XVII Les différents modèles :

On distingue 4 modèles principaux, mais on va illustrer que deux réseaux (le Perceptron et les réseaux multicouches) car ces derniers vont aider à comprendre notre travail :

- Les réseaux de Hopfield
- Les réseaux de Kohonen
- Le Perceptron
- Les réseaux (ou perceptron) multicouches

XVII.1 Le Perceptron :

XVII.1.1 Généralité :

En 1957, au Cornell Aeronautical Laboratory Frank Rosenblatt et ces collaborateurs fût conçu le célèbre << perceptron >>, une machine auto- adaptative capable d'apprendre. Cette découverte donna aux sciences cognitives une nouvelle tournure, et mis le connexionnisme en avance par rapport au cognitivisme. Ce modèle a pour propriétés :

- D'être spécifié en termes suffisamment précis pour permettre le test des performances annoncées.
- D'être suffisamment complexe pour que l'on puisse espérer des comportements intéressants.
- D'être suffisamment simple pour que ses performances puissent être prédites et analysées.
- Enfin et surtout d'être en accord avec les faits biologiques.

Le perceptron est un réseau à trois couches présenté par Rosenblatt, chaque couche joue un rôle bien précis, cela est montré dans la figure suivante :

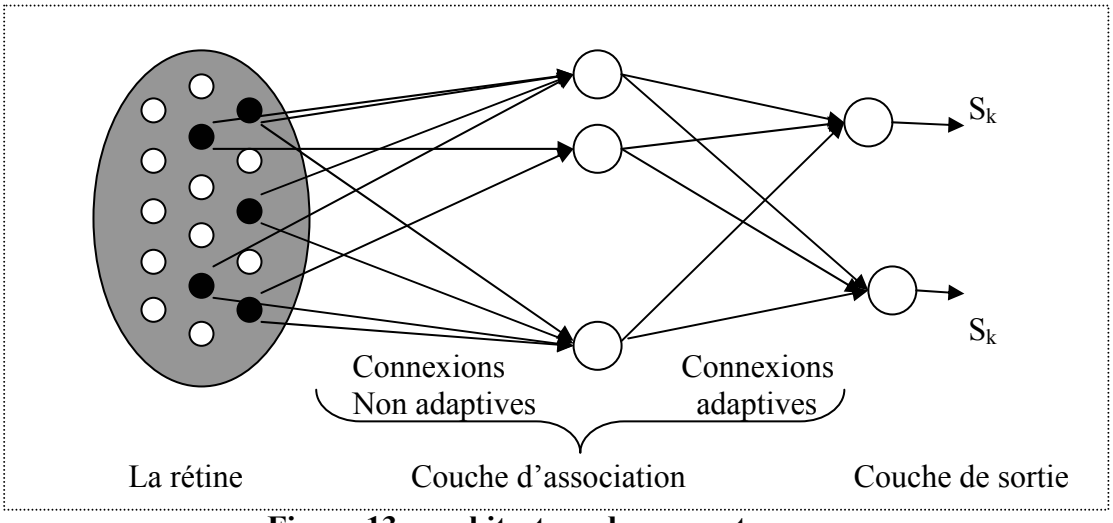


Figure 13 : architecture du perceptron

- **La rétine :** première couche. Elle contient les cellules d'entrées. Chaque cellule se contente de recopier la valeur qu'elle reçoit de l'extérieure sur sa sortie. [21]
- **La couche d'association :** (ou deuxième couche), elle est composée de cellules associatives. Chaque cellule a des connexions entrantes pouvant provenir de toutes ou d'une partie des cellules de la rétine. Les fonctions de transfert de ces cellules sont fixées à priori et sont en général différentes d'une cellule à une autre. [21]
- **La couche de la sortie :** Elle est composée d'automates à seuil. Chaque automate est connecté à toutes les sorties de la couche précédente. [21]

-Pour permettre l'évolution du réseau suivant les principes énoncés par Hebb, chaque connexion entre les cellules d'association et les cellules de décision elle est affectée d'un poids.

XVII.1.2 L'apprentissage :

L'apprentissage du perceptron est un apprentissage supervisé qui se fait correction d'erreurs. Dans ce modèle la fonction de transfert utilisée est linéaire, elle est de la forme : F(x)= Ech$(x-\theta)$. [5]

Où Ech est la fonction échelon (voir Figure 14). [5]

$$F(x)=\begin{cases}1 & si \quad x \succ 0\\ 0 & \sin on\end{cases}$$

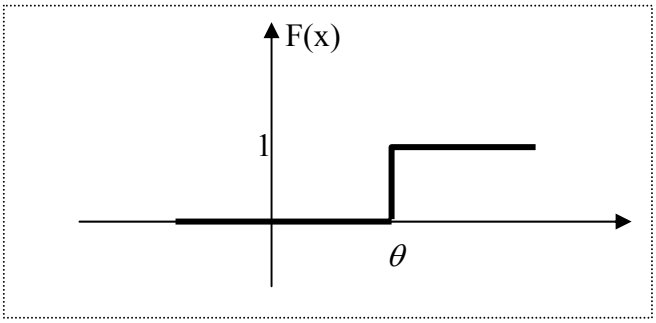


Figure 14 : fonction de transfert du perceptron.

Les poids sont mis à jour suivant la règle suivante [21] :

$$w_{ij}(t+1) = w_{ij}(t) - \alpha(y_i - d_i).x_j \qquad (5)$$

Où : $w_{ij}(t+1)$: Le nouveau poids de connexion entre les neurones i et j.

$w_{ij}(t)$: Le poids actuel de la connexion entre les neurones i et j.

α : Un paramètre d'adaptation de l'apprentissage (α >0).

x_j : L'entrée au neurone j.

y_i : La sortie du neurone i.

d_i : La sortie désirée du neurone i.

XVII.1.3 Limitation du perceptron :

En 1960 les deux chercheurs Minsky et Papert démontrèrent qu'avec un perceptron, on ne pouvait séparer que des données linéairement séparables et que pour des opérations plus générales de classifications il fallait mettre plusieurs couches. [2]

Puis la limitation a été généralisée pour tous les réseaux monocouches dont les neurones sont linéaires. [2]

Exemple de séparation linéaire :

Soient les deux classes A et B : on dit que les deux classes sont linéairement séparables si on arrive à les séparer par une ligne droite coupant le plan en deux (voir la Figure 15). [22]

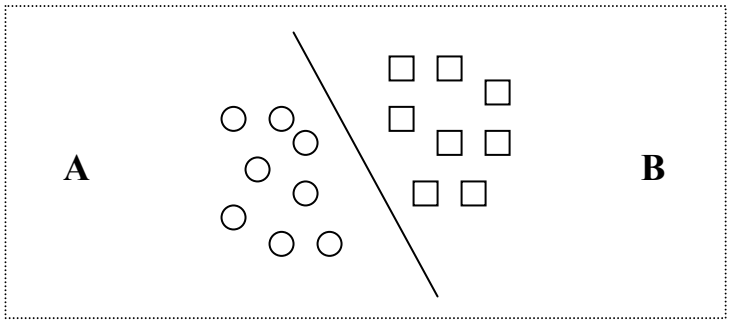


Figure 15 : la séparation linéaire entre la classe A et la classe B

Avec les réseaux multicouches le problème est résolu, car ces derniers peuvent résoudre toutes sorte de problèmes qu'ils soient linéairement séparables ou non. [22]

XVII.2 Les réseaux multicouches :

Ce sont des réseaux contenant 3 types de couches :

- Une couche en entrée qui représente les entrées du réseau.
- Une ou plusieurs couches cachées qui effectuent le traitement spécifique du réseau.
- Une couche en sortie qui délivre les résultats.

Le perceptron multicouche a connu son heure de gloire au début des travaux sur la reconnaissance de formes. Il s'agit d'un réseau constitué de processeurs élémentaires qui apprennent à connaître et à classer des formes.

Les neurones sont des éléments simples organisés en une seule couche. Ces couches sont complètement connectées, ils sont relies entre eux par des connexions pondérées, et il n'existe aucune connexion intra couche. (Chaque neurone est connecté à l'ensemble des neurones de la couche suivante, par des connexions dont les poids sont des nombres réels quelconques).

Ce sont les poids de connexions qui gouvernent le fonctionnement du réseau et « programment » une application de l'espace des entrées vers l'espace des sorties à l'aide d'une transformation non linéaire.

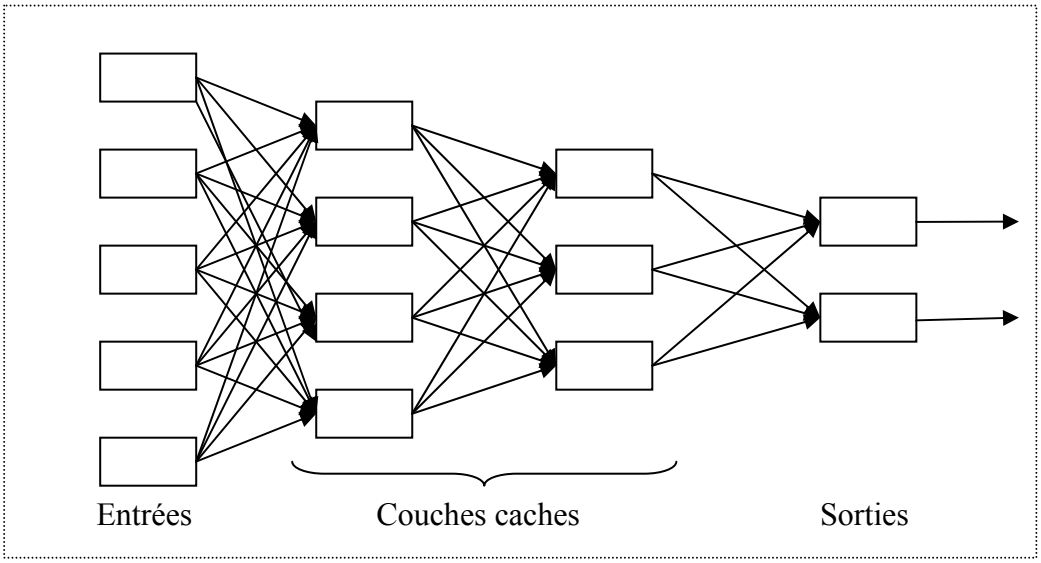


Figure 16 : réseau multicouche (2 couches cachées)

XVII.2.1 L'apprentissage :

XVII.2.1.1 L'algorithme de rétro propagation du gradient :

Cet algorithme est le plus utilisé pour l'apprentissage supervisé. Il permet d'ajuster les poids synaptiques du réseau en commençant par les dernières couches jusqu'aux premières.

Le neurone utilisé est fondamentalement de même nature que le neurone linéaire à seuil du perceptron. Il applique une fonction à la somme pondérée de ses entrées. Cette fonction est une version « lissée » de la fonction seuil. On utilise en général une fonction sigmoïde qui s'écrit [5]:

$$y = \sigma(p - \theta) = \frac{1}{1 + \exp[-k(p - \theta)]}$$

Où

$$y = \sigma(p - \theta) = \tanh\left[k(p - \theta)\right]$$

Dans la suite, les calcul seront conduit de façon générale avec une fonction N (.) arbitraire notée σ simplement supposée dérivable. [5]

On reprend ici la règle de Widrow-Hoff en l'adaptant au cas où la fonction neurone est non linéaire. On considère un réseau constitué de n neurones recevant des vecteurs à p composant (figure .1). Les p entrées x_k du réseau sont distribuées sur tous les neurones. La sortie du neurone i vaut [5]:

$$y_i = \sigma(p_i - \theta) = \sigma\left[\sum_{j=1}^{p} w_{i\,j} x_j - \theta\right] \quad (1)$$

A un vecteur d'entrée x, on veut associer un vecteur de sorite yd. Si les poids wik ont des valeurs quelconques ce qui est le cas initialement et avant la fin de l'apprentissage), le vecteur de sortie y observé est a priori différent de yd. On peut associer à cette différence l'erreur quadratique [5]:

$$\mathrm{E} = \frac{1}{2}\sum_{j=1}^{n}(y_i - yd_j)^2 \quad (2)$$

Calculons le gradient de cette erreur par rapport à w_{ik} :

$$\frac{\partial \mathrm{E}}{\partial w_{i\,k}} = \sum_{j=1}^{n}(y_j - yd_j)\frac{\partial(y_j - yd_j)}{\partial w_{i\,k}} \quad (3)$$

La dérivée partielle du membre de droite est nulle sauf pour j=i, car seule le sortie y_i est fonction du poids w_{ik}. De plus, la sortie désirée, fournie par le superviseur, ne dépend d'aucun poids. On arrive finalement à [5]:

$$\frac{\partial \mathrm{E}}{\partial w_{i\,k}} = (y_i - yd_i)\frac{\partial y_i}{\partial w_{i\,k}} = (y_i - yd_i)x_k\sigma'(p_i - \theta) \quad (4)$$

Où σ' est la dérivée de σ .

En posant $\delta_i = (y_i - yd_i)x_k\sigma'(p_i - \theta)$ le gradient de l'erreur quadratique E s'écrit [5] :

$$\frac{\partial \mathrm{E}}{\partial w_{i\,k}} = \delta_i x_k \tag{5}$$

Et la mise à jour du poids, selon le principe du gradient, s'écrit :

$$\Delta w_{i\,k} = -a\delta_i x_k \tag{6}$$ où a est un gain d'adaptation positif.

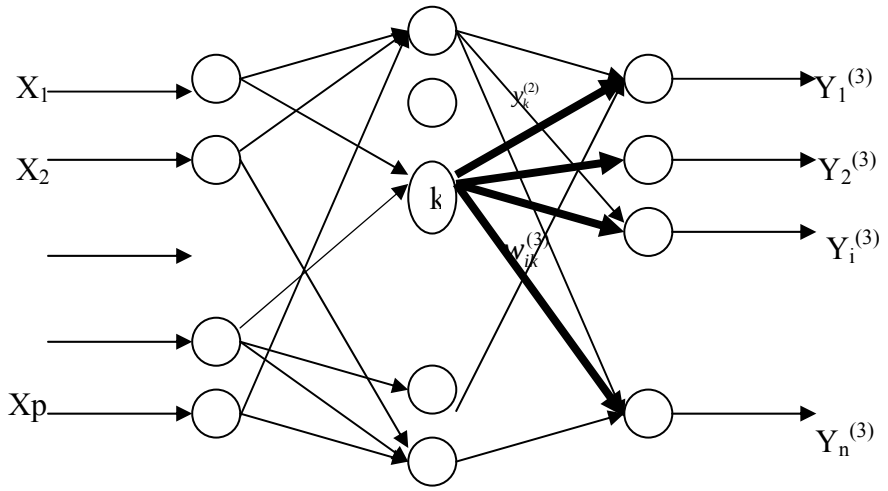


Figure.2. Réseau de neurones multicouches. Le numéro de la couche (des poids et des sorties) est indiqué en indice situé entre parenthèses et en haut. [5]

XVII.2.1.2 Rétro propagation du gradient, principe

Considérons maintenant le réseau à 3 couches de la figure 2, par convention la couche d'entrées ne fait que transmettre les entrées : le réseau ne comporte que deux couches réelles de traitement. La couche interne ou couche cachée et la couche de sortie. On veut comme précédemment associer un vecteur de sortie yd à un vecteur d'entrée x. En réalité, compte tenu des valeurs des poids des connexions, la sortie observée est y ≠ yd. On peut y associer le même terme d'erreur quadratique E qu'au paragraphe précédent. [5]

Nous allons montrer que l'on peut modifier les poids du réseau en minimisant cette erreur quadratique, d'une façon analogue à celle développée dans le paragraphe précédent.

Cependant, si cette méthode est directement applicable pour ajuster les poids de la dernière couche, elle ne l'est pas pour ceux des couches internes. En effet, on ne connaît pas la sortie désirée pour ces couches, et par conséquent on ne connaît pas directement le terme d'erreur associé à chaque couche interne. Il faut donc tenter d'exprimer l'erreur à la sortie de chaque neurone d'une couche quelconque à partir de l'erreur de la dernière couche, seule erreur directement mesurable. [5]

Ceci est possible par un calcul différentiel simple et l'on peut montrer que l'erreur δ_k^2 du neurone k de la couche 2 s'exprime par [5] :

$$\delta_k^{(2)} = \sigma'(p_k^{(2)}) \sum_{i \in couche\ 3} w_{i\,k}^{(3)} \delta_i^{(3)} \tag{7}$$

D'une façon plus générale, on peu calculer l'erreur associé à un neurone k d'une couche quelconque j à partir des erreurs des neurones de la couche suivante (j+1) pondérées par les poids $w_{ik}^{(j+1)}$ des connexions partant du neurone k vers les neurones de la couche suivante (en gras sur la figure 2) et la relation (7) peut se généraliser [5] :

$$\delta_k^{(j)} = \sigma'(p_k^{(j)} \sum_{i \in couche\ j+1} w_{ik}^{(j+1)}\ \delta_i^{(j+1)} \tag{8}$$

En appliquant la méthode du gradient, on propose une modification des poids selon la règle [5]:

$$\Delta w_{kh}^{(j)} = -a\delta_k^j y_h^{(j-1)} = -a\left[\sigma'(p_k^j) \sum_{i \in couche\ j+1} w_{ik}^{(j+1)} \delta_i^{(j+1)}\right] y_h^{(j+1)} \tag{9}$$

La mise à jour de l'ensemble des poids d'une couche nécessite la connaissance des erreurs associées à chaque neurone de la couche suivante. On appliquera l'algorithme d'ajustement des poids en partant de la dernière couche (pour laquelle les erreurs sont connues) vers la première. D'ou l'appellation de cet algorithme : algorithme de rétropropagation du gradient de l'erreur, ou simplement "back propagation algorithm" dans la littérature en langue anglaise. Cet algorithme a été développé indépendamment par différents chercheurs entre 82 et 85. [5]

XVII.2.1.3 Calcul détaillé

Dans ce paragraphe, nous détaillons les calculs qui permettent d'obtenir la règle (9) de modifications des poids selon la méthode de rétro propagation du gradient de l'erreur. En reprenant l'équation (2), l'erreur quadratique sur la sortie s'écrit [5]:

$$E = \frac{1}{2}\sum_{j=1}^{n}(y_i - yd_j)^2$$

Calculons le gradient de cette erreur par rapport au poids $w_{kh}^{(2)}$ associé à la connexion venant du neurone h de la couche j vers le neurone k de la couche 2 (l'indice de la couche cible est indiqué en haut) [5]:

$$\frac{\partial E}{\partial w_{kh}^{(2)}} = \sum_{i \in couche\ 3} \frac{\partial E}{\partial p_i^{(3)}} \frac{\partial p_i^{(3)}}{\partial w_{kh}^{(2)}} \quad (10)$$

Calculons d'abord le premier terme [5]:

$$\frac{\partial E}{\partial p_i^{(3)}} = \frac{\partial E}{\partial y_i^{(3)}} \frac{\partial y_i^{(3)}}{\partial p_i^{(3)}}$$

De la relation $y_i^{(3)} = \sigma\left[\sum_{j \in couche\ 2} w_{ij}^{(3)} y_j^{(2)}\right]$ on tire, en tenant compte l'expression de E :

$$\frac{\partial E}{\partial p_i^{(3)}} = (y_i^{(3)} - yd_i)\sigma'\left[p_i^{(3)}\right] \quad (11)$$

Calculons maintenant le second terme de (10). Le potentiel du neurone i de la couche 3 est la somme des sorties $y_j(2)$ de la couche 2 pondérées par les poids $w_{ij}(3)$ des neurones de la couche 3. On peut donc écrire [5]:

$$\frac{\partial p_i^{(3)}}{\partial w_{kh}^{(2)}} = \frac{\partial}{\partial w_{kh}^{(2)}}\left[\sum_{j \in couche\ 2} w_{ij}^{(3)} y_j^{(2)}\right] \quad (12)$$

Le second membre de cette expression, le seul terme de type $y_j(2)$ dépendant $w_{hk}(2)$ est $y_k(2)$, c'est-à-dire celui obtenu pour j=k. La relation (12) se donc à [5]:

$$\frac{\partial p_i^{(3)}}{\partial w_{kh}^{(2)}} = w_{ik}^{(3)} \frac{\partial}{\partial w_{kh}^{(2)}}\left[y_k^{(2)}\right] \quad (13)$$ dans laquelle on a pu sortir le poids $w_{jk}(3)$

En remplaçant y_k (2) (sortie du neurone k de la couche 2) par son expression, on arrive à [5]:

$$\frac{\partial p_i^{(3)}}{\partial w_{kh}^{(2)}} = w_{ik}^{(3)} \frac{\partial}{\partial w_{kh}^{(2)}}\left[\sigma\left(\sum_{m \in couche\ 1} w_{km}^{(2)} y_m^{(1)}\right)\right] \quad (14)$$

Le seul terme de la somme sur m dont la dérivée est nom nulle est celui qui contient $w_{kh}(2)$, c'est-à-dire celui correspond à m=h. En calculant la dérivée, on a finalement [5]:

$$\frac{\partial p_i^{(3)}}{\partial w_{kh}^{(2)}} = w_{ik}^{(3)} \sigma'(p_k^{(2)}) y_h^{(1)} \quad (15)$$

En reprenant maintenant les relations (11) et (15), on peut réécrire le gradient de l'erreur. [5]

$$\frac{\partial \mathrm{E}}{\partial w_{kh}^{(2)}} = \sum_{i \in couche\ 3} (y_i^{(3)} - yd_i) \sigma'(p_i^{(3)}) w_{ik}^{(3)} \sigma'(p_k^{(2)}) y_h^{(1)}$$

C'est-à-dire, en posant $\delta_i^{(3)} = (y_i^{(3)} - yd_i)\sigma'(p_i^{(3)})$ et en sortant les termes $y_h(1)$ et $\sigma'(p_k^{(2)})$ de la somme sur i [5]:

$$\frac{\partial \mathrm{E}}{\partial w_{kh}^{(2)}} = \left[\sum_{i \in couche\ 3} \delta_i^{(3)} w_{ik}^{(3)} \right] \sigma'(p_k^{(2)}) y_h^{(1)} \quad (16)$$

On remarque que cette opération est formellement identique au gradient de E par rapport à un poids de la dernière couche, si l'on pose [5]:

$$\delta_k^{(2)} = \left[\sum_{i \in couche\ 3} \delta_i^{(3)} w_{ik}^{(3)} \right] \sigma'(p_k^{(2)}) \quad (17)$$

Cette relation s'interprète en considérant que l'erreur affectée à un neurone k de la couche 2 est égale à la somme des erreurs des neurones de la couche suivante 3 pondérées par le poids reliant le neurone k à ces neurones de la couche 3. Cette relation est matérialisée par les connexions en traits gras sur la figure 2.

A partir de relations (16) et (17), on peut proposer finalement la règle d'adaptation suivante pour les connexions des neurones de la couche 2

$$\Delta w_{kh}^{(2)} = -a \delta_k^{(2)} y_h^{(1)} \quad (18)$$

Les trois dernières relations se généralisent à n'importe quelle couche cachée d'un réseau multicouche l'incrément d'un poids $w_{kh}(j)$ d'une couche j est le produit de l'entrée $y_h(j-1)$ provenant de la couche précédente (j-1) par l'erreur attribuée au neurone k. Pour la

dernière couche, l'erreur sur chaque neurone est donnée par le superviseur. Pour un neurone k d'une couche interne j, l'erreur $\delta_k^{(j)}$ est la somme des erreurs $\delta_k^{(j+1)}$ associées aux neurones i de la couche suivante (j+1) pondérées par le poids $w_{ik}^{(j+1)}$ entre ces dernières et le neurone k. On a alors [5] :

(19) $$\Delta w_{kh}^{(j)} = -a\delta_k^{(j)} y_h^{(j-1)}$$

Où l'erreur $\delta_k^{(j)}$ vaut:

$$\delta_k^{(j)} = y_k^{(j)} - yd_k$$

Pour les neurones de la dernières couche, et :

(20) $$\delta_k^{(j)} = \left[\sum_{i \in couche\ j+1} \delta_i^{j+1} w_{ik}^{(j+1)} \right] \sigma'(p_k^{(j)})$$ pour les neurones d'une couche interne. [5]

XVII.2.2 Aspect pratique de l'algorithme :

XVII.2.2.1 Lissage de la règle d'adaptation :

L'algorithme proposé au paragraphe précèdent est rarement utilisé tel quel en pratique. En effet, c'est un algorithme de type Gradient Stochastique, fondé sur la minimisation d'une erreur instantanée et non pas d'une erreur calculée sur toute la base d'apprentissage. La vitesse de convergence est donc assez lente et nécessite un pas d'adaptation petit pour éviter l'instabilité. Tous les raffinements des méthodes de moindres carrées peuvent être utilisés en tenant compte de l'aspect non linéaire du modèle. [5]

L'amélioration la plus couramment utilisée consiste à ajouter un terme de filtrage sur les incréments d'adaptation. Ce terme est souvent appelé "momentum" dans la littérature. En effet, l'adjonction de ce terme correspond à la minimisation d'un critère E approximativement égal à la somme des erreurs quadratiques pondérées exponentiellement. En effet, soit [5]:

(21) $$\mathrm{E}(n) = \frac{1}{2} \sum_{i=1}^{n} \gamma^{n-i} \left\| y - yd \right\|^2$$

Où γ est un facteur d'oubli inférieur à l.

En dérivant ce terme par rapport à w_{jk} (n) on obtient [5]:

(22) $$\frac{\partial \mathrm{E}(n)}{\partial w_{jk}(n)} = \sum_{i=1}^{n} \gamma^{n-i} x_k(i)(y_j(i) - yd_j(i))\sigma'(p_j(i))$$

Où $x_k(i)$ est l'entrée k à l'instant i (connectée au neurone j par la connexion w_{jk}), $y_j(i)$ est la sortie calculée du neurone j à l'instant i, et $y_{dj}(i)$ est la sortie désirée du neurone j à l'instant i. [5]

L'expression (22) peut encore se mettre sous la forme [5] :

$$\frac{\partial E(n)}{\partial w_{jk}(n)} = x_k(n)(y_j(n) - yd_j(n))\sigma'(p_j(n)) + \gamma \frac{\partial E(n-1)}{\partial w_{jk}(n)} \quad (23)$$

Si l'on admet l'hypothèse [5]:

$$\frac{\partial E(n-1)}{\partial w_{jk}(n)} = \frac{\partial E(n-1)}{\partial w_{jk}(n-1)}$$

À partir de la relation d'adaptation écrite à l'instant (n-1) [5]

$$w_{jk}(n) = w_{jk}(n-1) - a\frac{\partial E(n-1)}{\partial w_{jk}(n-1)} \quad (24)$$

On arrive à la relation [5] :

$$\frac{\partial E(n-1)}{\partial w_{jk}(n)} = \frac{\partial E(n-1)}{\partial w_{jk}(n-1)} = \frac{w_{jk}(n-1) - w_{jk}(n)}{a} \quad (25)$$

En tenant compte de (25), on peut déduire de la relation (23) la règle d'adaptation à l'instant n+1 [5]:

$$w_{jk}(n+1) = w_{jk}(n) - a\left[\delta_j(n)x_k(n) + \gamma\frac{w_{jk}(n-1) - w_{jk}(n)}{a}\right] \quad (26)$$

C'est-à-dire, en développant [5]:

$$w_{jk}(n+1) = w_{jk}(n) - a\delta_j(n)x_k(n) + \gamma\left[w_{jk}(n-1) - w_{jk}(n)\right] \quad (27)$$

En posant [5] :

$$w_{jk}(n) - w_{jk}(n-1) = \Delta w_{jk}(n)$$

On arrive à la règle d'apprentissage bien connue [5]:

$$w_{jk}(n+1) = w_{jk}(n) - a\delta_j(n)x_k(n) + \gamma\ \Delta w_{jk}(n) \quad (28)$$

On voit clairement, en reprenant la définition du nouveau terme d'erreur (21), que l'oubli γ correspond à la pondération exponentielle des erreurs. En pratique, on prend γ inférieur et voisin l. On peut aussi choisir un oubli variable, faible initialement et tendant asymptotiquement vers l : comme en automatique, ce choix peut s'avère pertinent dans le cas où les premières données sont douteuses. [5]

XVII.2.2.2 Choix de l'architecture du réseau :

La question à posée est combien doit-il y'avoir de couches cachées et combien doit-il y'avoir de neurones sur les couches cachées ?

L'idée générale pour dimensionner correctement un réseau en fonction du problème à résoudre, est d'utiliser un maximum de cellules dans la couche cachée. Si le réseau ne converge pas vers une solution satisfaisante, il est possible que plus de cellules cachées soient nécessaires.

En pratique, la plupart des problèmes peuvent être résolus par un réseau à une couche cachée. Pour le nombre de neurones cachés, il peut être estimé par une règle dite « géométrique pyramide ». Cette règle, constate que le nombre de neurones suit une forme pyramidale, le nombre de neurones est décroissant de l'entrée vers la sortie.

Le nombre de neurones dans chaque couche suit une progression géométrique. Donc si nous avons un réseau à (03) couches avec (n) entrées et (m) sorties, la couche cachée comprendra $\sqrt{n \times m}$ neurones.

XVII.2.2.3 Choix de l'ensemble d'apprentissage :

- la sélection des paires de vecteurs d'apprentissage du réseau doit être suffisante et convenable pour l'entraînement.
- l'ordre de représentation de ces exemples est important dans l'algorithme de rétropropagation (une disposition aléatoire des vecteurs d'apprentissage serait la meilleure solution).
- En général, on choisit de représenter chaque exemple une fois, et de répéter cette séquence.
- Pour que le réseau soit efficace, il faut que le BPN soit suffisamment entraîné sur des valeurs d'entrée d'une classe particulière, sinon l'identification des membres de cette classe serait peu faible. Il faut s'assurer donc que les données d'entraînement couvrent tout l'espace des entrées attendues.
- Il est à noter que l'on ne doit pas entraîner le réseau complètement avec un ensemble d'entrée appartenant à la même classe, puis basculer vers une autre classe car le réseau oubliera l'entraînement original.

XVII.2.2.4 Valeurs initiales des poids :

Les valeurs initiales des poids doivent être différentes de zéro, pour que le réseau de neurone ne dérive pas dans la phase d'apprentissage. En pratique, les poids sont initialisés aléatoirement par des valeurs entre [-0.5, 0.5].

XVII.2.2.5 Le bais :

Le terme du bais « θ » devra traiter comme tout autre poids provenant de la sortie d'une unité imaginaire, dont la valeur de sa sortie est égale a 1.

XVII.2.2.6 Choix du taux d'apprentissage :

Comme dans tout algorithme de type gradient, le pas d'adaptation doit être choisi avec soin si l'on veut une vitesse de convergence suffisante et un réseau performant. Le pas d'adaptation est généralement compris entre 0.05 et 0.25 pour assurer que le réseau atteint une solution.

Une petite valeur de ce dernier signifie que l'apprentissage du réseau sera très lent (un grand nombre d'itérations avant de converger). Cependant, s'il prend une valeur trop grand, le réseau risquera de ne pas converger (rebondir très loin du minimum).

XVII.2.2.7 Problème de minimaux locaux :

Ce problème concerne la possibilité de converger vers un minimum local dans l'espace des poids. Une fois que le réseau se stabilise sur un minimum, celui-ci peut être local ou global, l'apprentissage s'arrête ainsi. Si un minimum local est atteint, l'erreur au niveau de la sortie du réseau peut être grande, voir inacceptable, heureusement, ce problème ne cause pas trop de difficultés en pratique. Si le réseau s'arrête d'apprendre avant d'atteindre une solution acceptable, alors on peut régler le problème par un changement dans le nombre de neurones cachés ou bien dans les paramètres d'apprentissage, on peut aussi recommencer l'apprentissage avec un ensemble différent de poids initiaux.

Remarque :

La plupart des applications courantes utilisent une architecture « perceptron multicouches », dont les fonctions d'activation à chaque nœud sont de type sigmoïdal. La possibilité et la facilite d'une mise en ouvre du réseau et de son mode d'apprentissage en font une architecture certainement très attrayante, surtout si l'on envisage d'utiliser le réseau pour une application en temps réel. [4]

En outre, ce réseau entraîne des difficultés d'apprentissage dont les plus importantes sont [4] :

La présence de minima locaux dans la fonction de l'erreur dont l'origine s'explique, entre autre, par le fait que les paramètres (poids des connexions) apparaissent non linéairement dans l'expression de la sortie du réseau. Le fait que l'on puisse permuter indifféremment les nœuds des couches cachées (et les poids correspondants) tout en

produisant le même comportement à la sortie complique encore les choses, en multipliant le nombre d'optima ; les méthodes d'ajustement de type « gradient » peuvent donc conduire à un apprentissage sous optimale, tandis que les méthodes à caractère stochastique s'avèrent plus pesantes. [4]

La possibilité de paralysie durant l'apprentissage à cause de la saturation introduite par les sigmoïdes.

La possibilité de désapprentissage, du fait de caractère introduit par la combinaison de sigmoïdes telle qu'elle est réalisée dans les réseaux. En effet, l'ajustement des paramètres suite à une expérience particulière peut effacer l'effet d'un ajustement précèdent pour une expérience complètement différente, ce qui peut entraîner certaine inefficacité lors de l'apprentissage. En fait, rien ne garantit à l'avance que la généralisation obtenue est appropriée pour un problème donné. [4]

On peut dés lors songer à adopter un autre type d'architecture qui puisse éviter ces problèmes. Le réseau à fonction de base radiale ou bien radial basis function (RBF) répond en grande partie à ce désir, en proposant une structure de réseau préservant le caractère local de l'apprentissage des expériences.

Il est naturel d'imposer un certain nombre de contraintes sur le comportement d'un réseau. Parmi les contraintes possibles, certaines ont un intérêt tout particulier [4] :

- Contrainte de capacité d'approximation universelle.
- Contrainte de couverture : mathématiquement, elle s'exprime par le fait que, pour chaque sortie y_i associer à un vecteur d'entrée, il existe au moins un paramètres w_j tel que la valeur $\left| \frac{\partial\, y_i(u;w)}{\partial\, w_j} \right|$ (appelée « fonction de sensibilité ») soit suffisamment grande dans le voisinage de u.
- Contrainte de généralisation locale : si la valeur $\left| \frac{\partial\, y_i(u;w)}{\partial\, w_j} \right|$ est grande dans le voisinage de u, elle doit relativement être proche de zéro une fois que l'on s'éloigne de ce voisinage.

Les réseaux de neurones à fonction de base radiale (RBF) ou encore fonction noyau, peuvent aisément et naturellement satisfaire à ces contraintes. [4]

XVIII Réseaux de neurones à fonction de base radiale :

XVIII.1 Définition :

La méthode RBF a comme particularité que ses fonctions noyaux sont locales,

C'est-à-dire qu'elles ne donnent de réponses significatives que pour un domaine de valeurs restreint, leur champ récepteur. Ce champ est défini autour d'un point, le noyau (ou centre). [26]

La réponse de fonction noyau est maximale au centre du noyau, et décroît généralement de façon monotone avec la distance. Ainsi chaque fonction noyau est décrite par deux paramètres : la position de son centre c_i, et la taille σ_i du champ récepteur.

La fonction noyau la plus utilisée est la gaussienne qui est de forme :

$$\phi(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right)$$ [26].

XVIII.2 Architecture des réseaux à fonction de base radiale :

Ils sont constitués d'une couche d'entrée et de sortie, et comprennent une couche cachée de n unités qui ne réagissent significativement qu'à une partie restreinte de l'espace d'entrée suivant une fonction d'activation de type gaussien. [4]

Ainsi pour un réseau à une seule sortie, en utilisant un réseau à base radiale normalisée, la valeur de la sortie est donnée par [4]:

$$y = \sum_{j=1}^{n} w_j \frac{\rho_i(u)}{\sum_{j=1}^{n} \rho_j(u)} \qquad (\sum_{j=1}^{n} \rho_j(u) \text{ Est le terme de normalisation})$$

Ou : $\rho_j(u) = \exp\left(-\frac{1}{2}\sum \frac{(u_k - c_{k.j})^2}{\sigma_{k.j}^2}\right) = \exp\left(-\frac{1}{2}(u - c_j)^t Q_j (u - c_j)\right)$

Les vecteurs c_j (j=1,…, n) sont appelés « centre des gaussiennes » (ils ont la même dimension que le vecteur d'entrée u). Les matrices Q_j sont les matrices de rayons choisies le plus souvent diagonales. [4]

Dans le cas où les centres sont disposés en treillis régulier, les valeurs des rayons sont fixées par la distance entre centre voisins afin d'assurer un certain degré de recouvrement entre les différentes gaussiennes (champ d'influence des différents centres). Ce degré de recouvrement résulte d'un compromis entre les propriétés de localité et de généralisation. L'étendue de la généralisation est donc contrôlable. [4]

Remarque : Les réseaux RBF forment une classe particulière de réseaux multicouches.

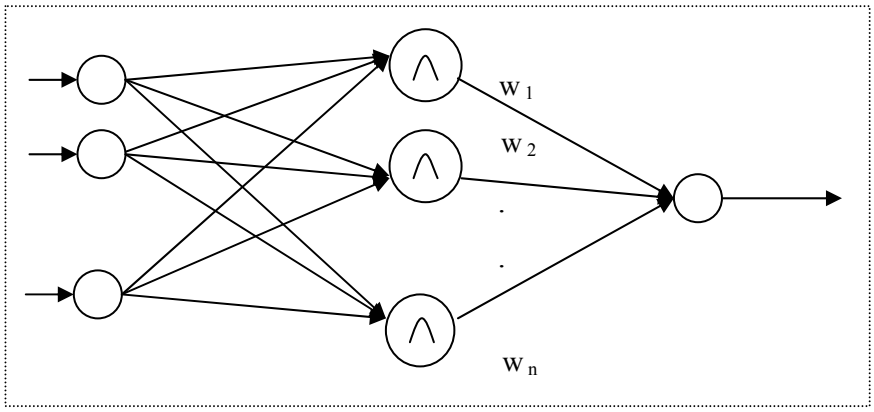


Figure 17 : structure d'un réseau RBF à une sortie

XVIII.3 Apprentissage :

L'apprentissage d'un réseau RBF est composé de 3 étapes :

XVIII.3.1 Estimation des centres des noyaux :

Dans les applications courantes des réseaux à base radiale, notamment en classification, les centres des gaussiennes sont choisis par des techniques de regroupement de données, le but étant de choisir les centres comme points représentatifs de zones de grande concentration de données. Néanmoins, lorsqu'il s'agit de travailler en ligne et en boucle fermée, ces méthodes de placement de centres s'avèrent peu efficaces ou peu utilisables. [4]

On peut alors penser a disposé les centres en treillis régulier pour couvrir uniformément la partie utile de l'espace d'entrée. [4]

XVIII.3.2 Estimation de la largeur des noyaux :

Déterminer la largeur σ_i des noyaux revient à déterminer comment les noyaux vont se recouvrir. Dans cet esprit, Moody et Darken (1989) ont proposés de fixer les σ_i en minimisant une fonction d'erreur définie afin de tenir compte d'un paramètre de recouvrement Q :

$$\varepsilon_Q = \frac{1}{2}\sum_{r=1}^{k}\left[\sum_{s=1}^{k}\exp\left(-\left(\frac{\left\| c_s - c_r \right\|}{\sigma_r}\right)^2\right)\left(\frac{\left\| c_s - c_r \right\|}{\sigma_r}\right)^2 - Q\right]$$

Les paramètres σ_i sont trouvés par une minimisation de cette fonction, par exemple à travers une méthode de descente de gradient.

Dans le cas où les centres c_i sont disposés en treillis régulier, les valeurs des rayons sont fixées par la distances entre centres voisins afin d'assurer un certain degré de recouvrement entre les différentes fonctions noyaux (champ d'influence des différents centres). [4]

XVIII.3.3 Estimation des coefficients synaptiques :

La règle de Widrow-hoff définie ci-dessus sera utilisée pour la modification des poids synaptiques, et qui s'applique uniquement aux réseaux à une couche cachée.

L'apprentissage se fait à partir d'une paire d'entraînement, formée d'un vecteur d'entrée x et un vecteur de sortie S désirée. Plus précisément le vecteur d'entrée x correspond aux valeurs des connexions d'entrées du réseau. Quand au vecteur de sortie S, il correspond aux valeurs des sorties du réseau que l'on désire obtenir à partir de x.

XIX Réseaux de neurones à fonction de base radiale généralisée :

La plupart des applications courantes utilisant les fonctions à base radiale disposent de centres et de rayons fixés une fois pour toutes. Les seuls paramètres ajustables sont les poids wj associés à chacune des n unités cachées (et donc à chacun des n centres). [4]

La sortie du réseau étant linéaire en ces paramètres, il est tout a fait concevable d'utiliser les techniques classiques d'identification paramétrique (par exemple, les moindre carrés ou toutes les variantes possibles) et de profiter ainsi de toute l'efficacité de ces méthodes pour obtenir un apprentissage rapide et précis. Même si l'approche à des avantages certains, d'autres facteurs peuvent déprécier son utilisation. [4]

- La plausibilité contestable d'une mise en œuvre biologique d'un algorithme de type " moindre carrés récurrents " ou " filtre de Kalman " ;
- La perte du parallélisme dans l'ajustement des paramètres :il est difficile de "paralléliser" efficacement les algorithmes à cause de couplages entre paramètres, il existe néanmoins des versions simplifiées des algorithmes d'identification rétablissant en partie le parallélisme au niveau de chaque unité (neurone) du réseau, ces simplifications se basent sur une approximation de la matrice hessienne inverse $\mathbf{H^{-1}}$ intervenant dans les algorithmes d'identification par une matrice presque diagonale, garantissant le découplage de l'ajustement des paramètres au niveau de chaque unité .
- La complexité des algorithmes requis.

. En outre, la disposition des centres a priori s'avère quelquefois impraticable. En effet, le nombre de centres requis est par fois trop élevé (dans le cas où l'on décide d'adopter une disposition en treillis régulier). [4]

Ce que nous proposons est d'utiliser alors un nombre beaucoup plus restreint de centres et de laisser libre la disposition de ces centres. Dans ce cas, l'algorithme d'apprentissage devra ajuster, outre les poids wj, les positions des centres ainsi que les valeurs des rayons associées. Idéalement, le réseau s'auto organisera pour placer un nombre élevé de centre la où la fonction est accidentée et espacer les centres dans les régions où la fonction est plus lisse.

Ce faisant, la sortie n'est plus linéaire en les paramètres (centres et rayons) et le risque de minima locaux apparaît par la même occasion. Les réseaux permettant non seulement l'apprentissage des poids des connexions w, mais aussi celui des centres et des rayons seront appelés « *réseaux à base radiale généralisés* ». [4]

Notons que la perte de la linéarité rend l'emploi des algorithmes d'identification comme le filtre de Kalman moins justifiable, voire inutile .Le problème de minima locaux n'est sans doute pas aussi critique que dans le cas des réseaux sigmoïdaux car l'introduction de connaissances *a priori* sous la forme d'une distribution initiale des centres, rayons et poids permet de disposer de valeurs "pas trop éloignées "des valeurs optimales. [4]

L'ajustement des centres et des rayons d'un réseau à base radiale généralisé peut être aisément effectué au moyen d'un algorithme de type "descente de gradient". Ce faisant, comme le montreront les équations ci-dessous, le parallélisme au niveau de chaque neurone est conservé (algorithme facilement parallélisable); autrement dit, chaque neurone j de la couche cachée est capable d'ajuster les poids wj, ainsi que les centres c_j et rayons σ_j, indépendamment des autres neurones de la couche cachée, car il n'utilise que ses propre paramètres (en plus de l'erreur sur la sortie et du facteur de normalisation) pour calculer les corrections. Pour p=1(une seule sortie).

En utilisant un réseau à base radiale normalisée et en se référant aux équations exprimant la sortie y du réseau en fonction des entrées u, les valeurs des gradients sont données par les équations suivantes [4]:

Soit : $J = \frac{1}{2}\left(y - y^*\right)^2$ le critère à minimiser. [4]

Posons : R (u) = $\sum_{i=1}^{n} \rho_i(u)$ (facteur de normalisation).

$$\frac{\partial J}{\partial w_j} = (y - y^*)\frac{\rho_j(u)}{R(u)}$$

$$\frac{\partial J}{\partial c_{j,k}} = \frac{\partial J}{\partial \rho_j}\frac{\partial \rho_j}{\partial c_{j,k}}$$

$$\frac{\partial J}{\partial \sigma_{j,k}} = \frac{\partial J}{\partial \rho_j} \frac{\partial \rho_j}{\partial \sigma_{j,k}}$$

$$\frac{\partial J}{\partial \rho_j} = \left(y\_y^*\right) \left[\frac{w_j\ R(u) - \sum_{i=1}^{n} \rho_i(u) w_i}{R^2(u)} \right]$$

$$\frac{\partial \rho_j}{\partial c_{j,k}} = \rho_j . \frac{u_k - c_{j,k}}{\sigma_{j,k}^2}$$

$$\frac{\partial \rho_j}{\partial \sigma_{j,k}} = \rho_j . \frac{\left(u_k - c_{j,k}\right)^2}{\sigma_{j,k}^3}$$

Les corrections s'obtiennent par [4] :

$$\Delta w_j = -\eta_w \frac{\partial J}{\partial w_j}$$

$$\Delta c_{j,k} = -\eta_c \frac{\partial J}{\partial c_{j,k}}$$

$$\Delta \sigma_{j,k} = -\eta_\sigma \frac{\partial J}{\partial \sigma_{j,k}}$$

Notons qu'il y a maintenant (1+2m)*n paramètres ajustables au lieu des n précédents (m est le nombre d'entrées). [4]

Il est quelquefois nécessaire de calculer la dérivée de J par rapport à l'entrée u du réseau (« rétropropagation étendue aux entrées »). Cette dérivée s'obtient aisément par les relations [4] :

$$\frac{\partial J}{\partial\ u_k} = \frac{\partial J}{\partial \rho_j} \frac{\partial \rho_j}{\partial\ u_k}$$

$$\frac{\partial \rho_j}{\partial\ u_k} = -\rho_j \frac{u_k - c_{j,k}}{\sigma_{j,k}^2}$$

L'algorithme de correction des paramètres d'un réseau à base radiale généralisé se transpose tout naturellement à l'adaptation de certains paramètres intervenant dans la modélisation de données. [4]

Remarque :

Le problème des minima locaux n'est sans doute pas aussi critique que dans le cas des réseaux sigmoïdaux car l'introduction de connaissances a priori sous la forme d'une

distribution initiale des centres, rayons et poids permet de disposer de valeurs « pas trop éloignées » des valeurs optimales.

XX Propriétés et limites des réseaux de neurones :

XX.1 Les propriétés :

L'intérêt porté aujourd'hui aux réseaux de neurones tient sa justification dans les propriétés suivantes :

XX.1.1 Le parallélisme :

Cette notion se situe à la base de l'architecture des réseaux de neurones considérés comme ensembles d'unités élémentaires qui travaillent simultanément.
L'intérêt d'une telle conception du traitement des données a été mis en évidence par l'échec des méthodes séquentielles pour traiter des problèmes qui nécessitent des quantités énormes de données. Le parallélisme permet une rapidité de calcul supérieur mais exige de penser et de poser différemment les problèmes à résoudre.

XX.1.2 Capacité d'adaptation :

Celle-ci manifeste tout d'abord dans les réseaux de neurones par la capacité d'apprentissage qui permet au réseau de tenir compte de nouvelles contraintes ou nouvelles données du monde extérieur. De plus, elle se caractérise dans certains réseaux par leur capacité d'auto organisation qui assure leur stabilité en tant que système dynamique.

XX.1.3 La mémoire distribuée :

Dans les réseaux de neurones, la mémoire correspond à une carte d'activation des neurones. L'intérêt de cette distribution de la mémoire sur plusieurs unités est la résistance au bruit.

XX.1.4 Capacité de généralisation :

Cette capacité se manifeste dans le fait que les réseaux de neurones peuvent à partir d'ensemble d'exemples, apprendre à trouver les règles sous-jacentes, ou à mimer les comportements qui permettent de résoudre les problèmes.

XX.2 Limites des réseaux de neurones :

En plus de quelques difficultés surmontables déjà exposées notamment dans le choix des paramètres du réseau, on peut citer d'autres :

- La plupart des réseaux sont simules sur des machines séquentielles. Ce qui entraîne rapidement des temps de calculs importants dés que la taille du problème devienne conséquent.
- L'un des principaux reproches fait aux modèles connexionnistes tient en leur incapacité à expliquer les résultats qu'ils fournissent.

XXI Les caractéristiques d'une bonne application :

On peut déterminer quelques-unes des caractéristiques des problèmes bien adaptés à une résolution par les réseaux de neurones.

Les règles qui permettent de résoudre le problème sont inconnues ou très difficiles à expliquer ou à formaliser. Cependant, on dispose d'un ensemble d'exemples qui correspondent à des entrées du problème et aux solutions qui leur sont données par des experts.

- Le problème fait intervenir des données bruitées.
- Le problème peut évoluer, par exemple en faisant varier son champ de conditions initiales.
- Le problème nécessite une grande rapidité de traitement, il doit par exemple être traité en réel.
- Il n'existe pas de solutions technologiques courantes.

On peut donc dresser la liste des domaines d'applications privilégie :

- Traitement de signal.
- Vision, parole.
- Reconnaissance des formes.
- Prévision et modélisation.
- Aide à la décision.
- Robotique.

Ces quelques domaines possèdent pratiquement toutes les caractéristiques exposées précédemment, c'est pourquoi ils constituent le noyau des applications des réseaux de neurones.

Complexité des algorithmes

XXII Introduction :

Un problème informatique peut être résolu de plusieurs manières. Toutes ces manières ne se sont pas équivalentes. Au-delà des critères subjectifs de distinction des algorithmes, on peut différencier les divers programmes par le moyen des critères suivants [7]:

- L'efficacité de l'algorithme en terme de durée d'exécution. Un algorithme est dit plus efficace qu'un autre si pour les mêmes données, il s'exécute en laps de temps plus court.
- L'efficacité de l'algorithme en espace mémoire de stockage. Un algorithme sera dit plus efficace qu'un autre si pour résoudre le même problème, il utilise moins d'espace mémoire.
- La fiabilité de l'algorithme qui est le degré d'absence de bug. Plus un programme est complexe, plus il y a des risques d'existence de bugs. Les bugs sont ces erreurs plus ou moins évidentes qui se manifestent lors de la mise en exploitation du programme. Un programme sera jugé plus fiable ou plus stable qu'un autre s'il présente moins de bugs «connus ».
- La robustesse de l'algorithme qui mesure son degré de tolérance aux erreurs des utilisateurs et sa résistance aux attaques des pirates. Un programme sera plus robuste qu'un autre s'il résiste mieux aux erreurs de manipulations des utilisateurs plus ou moins attentionnés notamment dans ses composantes entrées/sorties.

Comment développer un algorithme efficace pour résoudre un problème donné ? Parmi plusieurs algorithmes résolvant un même problème, lequel choisir ? Pour illustrer l'importance de ces questions, considérer le problème du calcul du déterminant d'une matrice. Un algorithme classique, dû à Gauss et Jordan, peut calculer le déterminant d'une matrice 20x20 en moins d'un dixième de seconde sur un ordinateur contemporain. Un autre algorithme tout aussi classique, basé sur la définition récursive du déterminant, prendrait dix millions d'années pour arriver au même résultat sur le même ordinateur [7].

En plus de ces critères de qualité d'un programme et qui font l'unanimité au sein de la communauté des programmeurs il existe d'autres critères de qualité plus ou moins subjectifs et surtout souvent source de polémique entre programmeurs sur la manière de les réaliser [7] :

- La beauté d'un algorithme, à savoir sa concision, sa modularité, le choix des noms de identificateurs, l'indentation des instructions, etc.
- L'inter-opérabilité d'un programme c'est-à-dire sa capacité à s'interfacer avec programmes avec plus ou moins de bonheur et de facilité.
- La réutilisabilité du code, c'est-à-dire la facilité de maintenance du code ou de son intégration à d'autres projets de développement.
- Absence d'effet de bords. On appelle effet de bord toute communication entre le programme et son environnement en dehors de ses paramètres d'entrées/sorties. Toute d'entrée/sortie par une variable globale, par un accès à un fichier standard ou pas ou à une base de données sont considérés comme des effets de bord. Un programme sans effet de bord posera moins de problème lors de son débogage et lors de sa maintenance.

Un algorithme est supposé être une boite noire faisant correctement le travail qui lui est demandé en consommant ses entrées et en produisant des sorties sans aucun effet de bord, sans bug, et sans trous de sécurité. On n'est pas obligé de regarder dans le code de l'algorithme pour l'utiliser. La connaissance des ses entrées/sorties suffit pour l'exploiter sans risque [7].

XXIII Définitions :

XXIII.1 Notion d'algorithme :

L'algorithme est une science apparue, il y a très longtemps, bien avant l'idée même d'ordinateur [8].

- Citons, vers 1800 avant J-C, les babyloniens de l'époque d'Hammurabi formulant des règles précises pour la résolution de certains types d'équations.
- Plus tard, au 3^{éme} siècle avant J-C, chez les grecs, fleurissent un grand nombre de procédés de calcul, dont le célèbre « algorithme d'Euclide ».
- En perse, au 9 ^{éme} siècle après J-C, on trouve l'origine du mot « algorithme », qui provient du nom de Abu Ja'fa Mohammed Ibn Mûsâ Al-Khowâ-rismi, qui écrit un ouvrage d'arithmétique utilisant les règles de calcul sur la représentation décimale des nombres, et montra l'inutilité des tables et abaques. Il eut une influence capitale pendant plusieurs siècles.

- Au fil du temps, la signification du mot s'élargit et finalement vient à désigner tout procédé de calcul systématique, voire automatique, mais sans référence nécessaire à une machine.
- Côté informatique du 21^{éme} siècle et en première approche, on peut dire qu'un algorithme est un « mode d'emploi pour résoudre un problème ».

XXIII.2 Définition 1 d'un algorithme :

Un algorithme est un procédé de calcul automatique composé d'un ensemble fini d'étapes, chaque étape étant formée d'un nombre fini d'étapes élémentaires, qui permet de résoudre le problème en donnant la sortie requise. Chaque étape élémentaire est [8] :

- Définie de façon rigoureuse et non ambiguë.
- Effective, c à d, pouvant être réalisé par une machine.

XXIII.3 Définition 2 d'un algorithme :

C'est un ensemble de règles opératoires rigoureuses, ordonnant à un processeur d'exécuter dans un ordre déterminé un nombre fini d'opérations élémentaires ; il oblige à une programmation structurée [9].

Un algorithme est écrit en utilisant un langage de description d'algorithme (LDA).

L'algorithme ne doit pas être confondu avec le programme proprement dit.

XXIII.4 Le langage de description d'algorithme :

Ce langage utilise un ensemble de mots clés et de structures permettant de décrire de manière complète, claire l'ensemble des opérations à exécuter sur des données pour obtenir des résultats ; on n'hésite donc pas à agrémenter l'algorithme de nombreux commentaires.

L'avantage d'un tel langage est de pouvoir être facilement transcrit dans langage de programmation structuré (pascal, C,…) [9].

XXIV Structure d'un algorithme :

XXIV.1 Représentation :

- L'en-tête algorithme nom de l'algorithme ;
- Les déclarations de constantes, variables, structures
  - const liste des constantes;
  - var liste des variables;
  - struct liste des structures;

- Les déclarations de fonctions et procédures $\begin{cases} \underline{fonc}\ liste\ des\ fonctions; \\ \underline{proc}\ liste\ des\ procédures; \end{cases}$

- Le corps de l'algorithme $\begin{cases} \underline{début} \\ action\ 1; \\ action\ 2; \\ \vdots \\ action\ n; \\ \underline{fin}\ \underline{algorithme} \end{cases}$

Tous les mots clés sont soulignés est écrits en miniscules.

Une marque de terminaison (;) est utilisée entre chaque action. [9]

XXIV.2 Définitions :

XXIV.2.1 L'en-tête : Il permet tout simplement d'identifie un algorithme.

XXIV.2.2 Les déclarations : C'est une liste exhaustive des objets, grandeurs utilisés et manipulés dans le corps de l'algorithme ; cette liste est placée en début de l'algorithme.

XXIV.2.3 Le corps : Dans cette partie de l'algorithme, sont placées les taches (instructions, opérations,…) à exécuter.

XXIV.2.4 Les commentaires : Pour mettre une interprétation aisée de l'algorithme, l'utilisation de commentaires est vivement conseiller. [9]

De plus, l'algorithme doit toujours terminer après un nombre fini d'opération, quelle que soit la donnée en entrée, et fournir un résultat.

Les algorithmes que nous considérons sont déterministes : étant donné un algorithme toute exécution de cet algorithme sur les mêmes données donne le même résultat. [8]

XXIV.3 Types de données :

Pour pouvoir d'écrire un algorithme, la spécification du problème doit préciser les structures de données stockant entrées et sorties. Il n'est pas souvent nécessaire de décrire les structures de données en détail. Pour décrire l'algorithme, il suffit souvent d'une structure de données abstraite, pour laquelle on définit un type abstrait et des opérations abstraites. [8]

Un type de données est un ensemble de valeurs muni d'opérations permettant de faire des calculs sur ces valeurs.

Un type abstrait spécifie des ensembles de toutes les valeurs possibles, alors qu'une classe : correspond à un ensemble fini d'objets, qui peuvent être crées ou détruits, et auxquels on peut affecter des valeurs.

Les structures de données abstraites permettent la modularité dans la conception des algorithmes : on étudie d'un coté, étant donné une structure de données abstraites, on étudie ses implémentations possibles et de l'autre, étant donné un problème, on le résout par un algorithme qui accède aux données uniquement par l'intermédiaire des opérations abstraites, sans se préoccuper de l'implémentation des ces opérations. [8]

Exemple de type abstrait [8]: type booléen, ensemble de deux valeurs muni des opérations de négation, files de priorité, conjonction, disjonction.

Structures de données abstraites fréquemment utilisées : piles, files, dictionnaires, files de priorité, arbres, etc.

La conception d'un algorithme élaboré se fait en plusieurs étapes : par raffinements successifs.

Démarche descendante : on se donne la définition des types de données (leur spécification, indépendamment de toute implémentation) et on conçoit l'algorithme à ce niveau. On donne ensuite une représentation concrète des types et des opérations (qui peuvent encore être abstraite), puis en descendant les niveaux d'abstractions successives, on arrive à un programme exécutable dans un langage. [8]

XXV Complexité des algorithmes :

Quand on tente de résoudre un problème, la question se pose souvent du choix d'un algorithme. Quels critères peuvent guider ce choix ? Deux besoins contradictoires sont fréquemment en présence : l'algorithme doit [10]:

- Etre simple à comprendre, à mettre en œuvre et à mettre au point.
- Mettre intelligemment à contribution des ressources de l'ordinateur et plus précisément, il doit s'exécuter le plus rapidement possible.

Si un algorithme ne doit servir qu'un petit nombre de fois, le premier critère est le plus important. Par contre, s'il doit être employé souvent, le temps d'exécution risque d'être un facteur plus déterminant que le temps passé à l'écriture. [10]

Ce qui nous intéresse, étant donné un problème, est de trouver le meilleur algorithme. Pour cela, on va comparer des algorithmes qui ont la même spécification. Ce que l'on veut pouvoir dire : « Sur toute machine, quel que soit le langage de programmation, l'algorithme A1 est meilleur que l'algorithme A2 pour les données de grand taille. » [8]

XXV.1 Définition :

La complexité est une mesure de la « difficulté » du calcul d'un algorithme en fonction de la taille n des données [12]. C'est une évaluation du coût d'exécution d'un algorithme en termes de temps (complexité temporelle) ou d'espace (complexité spatiale). [11]

XXV.1.1 Complexité spatiale :

Elle permet d'évaluer l'occupation mémoire (le nombre de cases mémoire nécessaires) que va nécessiter un algorithme en fonction (toujours) de certains paramètres. [13]

XXV.1.2 Complexité temporelle :

Elle permet d'évaluer quel va être la rapidité d'exécution d'un algorithme en fonction d'un ou plusieurs paramètres dépendant de la ou des données fournies en entrée. [13]

C'est le nombre d'opérations élémentaires (affectations, comparaisons, opérations arithmétiques) effectuer par un algorithme. Ce nombre s'exprime en fonction de la taille n des données. [14]

Si l'intérêt de la complexité temporelle semble évident pour étudie la vitesse, les performances, ou la capacité à supporter la charge, je ne vous apprendrai rien en vous disant que de nos jours la mémoire n'est plus un paramètre réellement limitant. D'une part, le prix de la RAM a beaucoup baissé durant les vingt dernières années et d'autre part, la mémoire virtuelle est également une réponse à ce problème. Nous allons donc délaisser la complexité spatiale des algorithmes pour nous pencher que sur leur complexité temporelle. [13]

XXV.2 Mesure de la complexité :

Evaluer le temps pris par un algorithme, fort bien, mais sur la base de quelle unité ? Imaginons que nous choisissions la seconde. Ce n'est pas un bon choix! En effet, la vitesse des microprocesseurs ne cesse d'évoluer, donc une mesure de temps fait aujourd'hui n'aurait plus aucun sens dans quelques années à peine!

Le temps d'exécution d'un algorithme dépend des facteurs suivants [10] :

- Les données entrant dans le programme.
- La qualité du code généré par le compilateur.

- La nature de la vitesse d'exécution des instructions du microprocesseur.
- La complexité algorithmique du programme.

En général, la longueur des données est une unité de mesure adéquate. Il est alors habituel de parler d'un temps d'exécution T (n) pour un programme portant sur des données de taille n. A titre d'exemple, les programme peuvent avoir une complexité de $T(n) = cn^2$ où c est une constante. Les unités choisies pour T (n) restent à définir mais il est possible de se représenter T (n) comme le nombre d'instructions « élémentaires » exécutées par une machine formelle [10].

Donc nous allons pouvoir exprimer nos complexités. Mais si nous avons le choix entre deux algorithmes ayant deux complexités différentes, comment savoir lequel est le meilleur ? C'est là que l'outil mathématique intervient. Il va nous permettre de comparer les complexités. [13]

XXV.3La notation « grand O, Θ et Ω » :

Lors de l'étude d'une suite ou d'une fonction dont la nature est compliquée, certaines questions ne nécessitent que des renseignements d'ordre qualitatif tels que $f(x) \to 0$ ou $f(x) \to +\infty$ pour $x \to +\infty$. D'autres exigent un contrôle quantitatif très précis, défini par des inégalités explicites. Les comportements asymptotiques relèvent d'une préoccupation intermédiaire : dans de nombreux problèmes, on remplace la quantité étudiée par une autre plus simple sans que « à la limite », le résultat en soit modifier. [10]

Pour décrire les croissances asymptotiques des fonctions, on utilise la notation de Landau « O ». D'une manière analogue, quand on parle d'une complexité algorithmique en $O(n^2)$ pour un programme donné, on veut dire qu'il existe deux constantes c et $n_0>0$, telles que pour $n >= n_0$ on a $T(n) <= cn^2$. Il doit être clair que cette notation est ensembliste, et que la fonction T est décrite par la formulation précédente comme appartenant à la classe de toutes les fonctions satisfaisant cette condition. Par abus de langage on dit plus souvent que : T (n) est en $O(n^2)$ voire T (n)=O (n). [10]

Par la suite, nous supposerons que l'ensemble des fonctions de complexité est défini sur l'ensemble N des entiers naturels et qu'elles prennent leur valeur dans R^+, l'ensemble des nombres réels positifs ou nuls. On dit alors que T (n) est en O (f (n)) s'il existe deux constantes c et n_0 telles que $T(n) <= c*f(n)$ chaque fois que $n >= n_0$. Un programme dont la complexité est O (f (n)) est dit appartenir à la classe de complexité O (f (n)) ou encore avoir une croissance en complexité de f (n). [10]

XXV.3.1 Définition 1 :

Soient f et g deux fonctions de N dans R^{+*}, s'il existe un réel c>0 et un entier positif (un rand) n_0 tel que : pour toute n> n_0, f (n) <= c*g (n) on dit que f est en O (g) (f est asymptotiquement dominée par g). [11]

XXV.3.2 Définition 2 :

Soient f et g deux fonctions de N dans R^{+*}, on dit qu'une fonction f (n) est en Θ (g (n)) s'il existe deux constante strictement positives c_1 et c_2 et un n_0 tels que :

0< = c_1* g (n) <= f (n) <= c_2* g (n) pour toute n>= n_0, autrement dit : f est en O (g) et g est en O (f) (f et g sont de même ordre de grandeur asymptotique).La notation Θ borne donc une fonction entre 2 facteurs constants. [11]

XXV.3.3 Définition 3 :

On dit qu'une fonction f (n) est en Ω (g (n)) s'il existe une constante strictement positive c et un n_0 tels que f (n)>= c*g (n) pour tout n>= n_0. [10]

En pratique f représente une quantité à étudier (temps, nombre d'opérations) et g fait partie d'une échelle de fonctions simples (n, $n\log_2$ (n), n2, etc.…) destinée à informer sur le comportement asymptotique de f. [11]

Ainsi avec un grand O, on sait à quoi s'attendre dans le pire des cas alors qu'avec un grand Θ, on sait à quoi s'attendre tout le temps. En général, on utilise surtout le grand O. [13]

XXV.4Les principales classes de complexité :

Habituellement la plupart des algorithmes admettent un paramètre de base n qui correspond grosso modo à la taille du problème à résoudre. Cela peut être le degré d'un polynôme, le nombre d'enregistrement dans une base de données ou encore le nombre d'éléments dans un tableau. L'analyse de la complexité d'un algorithme reviendra à étudier l'incidence de l'augmentation de n sur la durée d'exécution du programme ou l'espace mémoire. [7]

Grâce à notre nouvel outil « O » nous allons pouvoir comparer les complexités entre elles[13]. De plus, on utilise la notation « O » pour indiquer l'efficacité de l'algorithme [7].

Ci-dessous les complexités d'algorithmes les plus classiques habituellement rencontrés [7]:

XXV.4.1 Complexité constante (O (1)):

On rencontre cette complexité quand toutes les instructions sont exécutées une seule fois qu'elle que soit la taille n du problème.

XXV.4.2 Complexité logarithmique (O (ln (n)) (ou ln^k (n), k>1)):

La durée d'exécution croit légèrement avec n. ce cas de figure se rencontre quand la taille du problème est divisée par une entité constante à chaque itération. Par exemple, la recherche dichotomique dans un vecteur trié à n composantes.

XXV.4.3 Complexité linéaire (O (n)) :

C'est typiquement le cas d'un programme avec une boucle de 1 à n et le corps de la boucle effectue un travail de durée constante et indépendante de n. Par exemple, la recherche linéaire dans vecteur.

XXV.4.4 Complexité quasi linéaire (O (n*ln (n))):

Se rencontre dans les algorithmes où à chaque itération la taille du problème est divisée par une constante avec à chaque fois un parcours linéaire des données. Un exemple typique de ce genre de complexité est l'algorithme de tri « quick sort » qui, de manière récursive, divise le tableau à trier en deux morceaux par rapport à une valeur particulière appelée pivot, trie ensuite la moitie du tableau puis l'autre moitie etc.… s'il n'y avait pas cette opération de division l'algorithme serait l'algorithmique puisqu'on divise par 2 la taille du problème à chaque étape, le fait de reconstituer à chaque fois le tableau en parcourant séquentiellement les données ajoute ce facteur n au ln(n). Noter que la complexité n-logarithmique est tributaire du bon choix du pivot.

XXV.4.5 Complexité polynomiale (O (n^k) (k>1)):

Pour k=2 c'est la complexité quadratique. Typiquement c'est le cas d'algorithmes avec deux boucles imbriquées. Par exemple, le coût d'un tri par insertion d'un vecteur de n éléments est un O (n^2).

Pour k= 3, c'est la complexité cubique. Idem quadratique mais avec ici par exemple trois boucle imbriquées.

XXV.4.6 Complexité exponentielle (O (a^n) pour a>1):

Les algorithmes de ce genre sont dits « naïfs » car ils sont inefficaces et inutilisable dés que n dépasse 50. Par exemple, la prévision des n coûts possibles dans une partie d'échecs ou la recherche de toutes les combinaisons pour le remplissage d'un carré magique d'ordre n.

Remarque :

Noter que les complexités « lourdes » du type exponentiel ou pire sont parfois inévitables pour résoudre certains problèmes particuliers. Ainsi, les problèmes qui appartiennent à une catégorie que l'on nome NP-complets ne peuvent satisfaire au mieux que d'algorithmes à complexité exponentielle. Dans ce genre de situation peu importe la machine, peu importe le langage, peu importe les optimisations géniales du programmeur, il suffit d'augmenter légèrement n pour écouler n'importe quelle machine sous la charge de calcul!

A l'inverse, le top du top, c'est la complexité logarithmique. Quand vous avez un algorithme de ce genre, c'est merveilleux. Vous avez beau augmenté n dans des proportions ahurissantes, votre machine tiendra la charge! [13]

XXV.5Règles de calcul de complexité :

XXV.5.1 Définition :

La notion de coût : On définit pour une donnée d, le coût de l'algorithme A pour cette donnée d : $cout_A(d)$.On définit les trois mesures de complexités suivantes [17] :

- Complexité dans le pire des cas : elle est définie par :

$$Sup_A(n)=\sup\{cout_A(d)/\ d : \text{donnée de taille } n\}.$$

- Complexité dans le meilleur des cas s : elle est définie par :

$$Inf_A(n)=\inf\{cout_A(d)/\ d \text{ donnée de taille } n\}.$$

- Complexité en moyenne : elle est définie par :

$$Moy_A(n) = \sum_{d\ de\ taille\ n} p(d) * cout(d)$$

Où P (d) est la probabilité d'avoir en entrée une instance d parmi toutes les données de taille n.

XXV.5.2 Complexité en pire des cas :

Le calcul précis de la complexité d'un algorithme peut se révéler être un problème mathématique assez complexe. Cependant, dans la pratique, on se contente d'utiliser quelques principes fondamentaux [10] :

XXV.5.2.1 Règle de la Somme :

Supposons que deux modules P_1 et P_2 aient des complexités T_1 (n) et T_2 (n) et que T_1 (n) est en O (f (n)) et T_2 (n) est ne O (g (n)) alors T_1 (n) +T_2 (n), la complexité de P_1 suivi de P_2 est en O (max (f (n), g (n))). [10]

XXV.5.2.2 Règle du Produit :

Si T_1 (n) et T_2 (n) sont en O (f (n)) et O (g (n)) alors T_1 (n)*T_2 (n) est en O (f (n)*g (n)). [10]

XXV.5.2.3 Grandes lignes de l'analyse d'un algorithme :

- La complexité de toute opération de lecture ou d'écriture peut en général se mesurer par O (1).
- La complexité d'une suite d'itération est déterminée par la règle de la somme. Autrement dit, elle est égale à un constant multiplicatif pré, à la complexité la plus forte parmi toutes les instructions de la suite.
- La complexité d'une instruction conditionnelle (if…then…else…) est égale à celle des instructions exécutées dans la condition plus celle de l'évaluation de la condition. On prendra toujours la plus grande entre celle dans le cas où la condition est vraie et celle où la condition est fausse. [10]

Nous essayons ici de fixer des règles pour aider à l'évaluation de la complexité en temps ou en nombre d'opérations d'un algorithme.

Notons T_A (n) le temps ou le nombre d'opération correspondant à la suite d'actions A, ou au calcul de l'expression A. une suite d'action est considérée ici comme une action élémentaire. [11]

XXV.5.2.3.1 Règle-1 : Enchaînement

Soient deux suites d'action A_1 et A_2, et A_1+A_2, la suite « A_1 suivi de A_2 » alors : T_{A1+A2} (n)= T_{A1} (n) + T_{A2} (n). [11]

XXV.5.2.3.2 Règle-2 : Conditionnelle

Soit une action A de la forme « Si C alors A_1 Sinon A_2 Fin Si » alors : T_A (n)= T_C (n) + Max (T_{A1} (n), T_{A2} (n)). [11]

XXV.5.2.3.3 Règle-3 : Itération (Tant Que)

Soit une action A de la forme « Tant Que C faire A_1 Fin Tant Que » alors :

$$T_A(n) = \sum_{i=1}^{niter(n)} \left(T_C(i,n) + T_{A1}(i,n)\right) + T_C\left(niter(n)+1, n\right) [11]$$

Cependant cette règle est trop abstraite et ce qu'il faut en retenir est surtout le lien entre les structures répétitives et la sommation, comme nous l'avons vu précédemment. En particulier il est bon de lier la variable indice de la sommation avec une variable « compteur » de la structure répétitive lorsqu'elle existe. Il est cependant intéressant de noter que [11] :

- La condition C est exécutée dans le « Tant Que » une fois de plus que le « corps » A_1 de la boucle.
- T_c n'est pas toujours une constante, mais peut dépendre de n et du rang i de l'itération, qui est souvent lié à une variable.

XXV.5.2.3.4 Règle-4 : Fonctions et Procédures non récursives

On évalue d'abord les fonctions et les procédures qui ne contiennent pas d'appels à d'autres fonctions et procédures, puis celles qui contiennent des appels aux précédentes, etc….. [11]

Remarque importante :

Lorsqu'on fait un appel, il faut en toute rigueur compter l'appel lui-même comme une opération particulière, mais aussi compter les opérations correspondant au passage de l'argument.

Plus précisément : lors d'un passage par valeur quelles sont les opérations mises en jeu ? Pour chaque argument passé il faut évaluer l'argument (par exemple l'addition pour factorielle (n+1), et affecter cette valeur à une nouvelle variable (locale à la fonction). On néglige souvent cette dernière opération, cependant si l'argument passé est un tableau de taille N, alors l'affectation correspond à N affectations élémentaires, et ce coût n'est plus négligeable. C'est en particulier une des raisons pour lesquelles on évite souvent de passer par valeur un tableau même si sa valeur ne doit pas être modifiée par la procédure (ou fonction).

En effet un passage par adresse ne correspond pas à N affectations élémentaire puisque seule l'adresse en mémoire du tableau est fournie à la procédure lors de l'appel. [11]

Application

XXVI Modélisation

La plus part des applications courantes utilisent une architecture «réseau multicouche» dont les fonctions d'activation sont de type sigmoïdale. Ce type d'architecture entraîne des difficultés d'apprentissage dont les plus importantes :

- La présence de minima locaux.
- La possibilité de paralysie durant l'apprentissage.
- La possibilité de désapprentissage.
- La difficulté d'introduire de l'information à priori. (ce qui est dû au caractère distribué et non locale de la représentation).
- Grand temps de calcul.

A cet effet, et pour éviter ces problèmes dus à l'apprentissage, on a utilisé les réseaux de neurones à fonction de base radiale (Rbf) avec une fonction noyau Gaussienne qui est décrite par deux paramètres qui sont:

La position de son centre c_i, et la taille σ_i de la largeur du noyau.

$\Phi(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right)$ Fonction noyau.

$\phi_{pj}(t) = \frac{\Phi_j(\|x_p - c_j\|)}{\sum_{j=1}^{k} \Phi_j(\|x_p - c_j\|)}$: La sortie des neurones de la couche cachée à l'instant t.

$S_p(t)$: La sortie du réseau à l'instant t, $S_p(t) = \sum_{j=1}^{k} w_j \frac{\Phi_j(\|x_p - c_j\|)}{\sum_{j=1}^{k} \Phi_j(\|x_p - c_j\|)}$.

p : représente l'exemple courant.

Donc pour effectuer notre travail (Prévision), on a construit les modèles d'architecture de réseaux suivants :

- Le premier réseau : est un Rbf, contenant un neurone d'entré (x_t), une couche cachée contenant k (k fonctions gaussiennes) neurones et un neurone de sortie (x_{t+1}).
  Mathématiquement, ce réseau est équivalent au modèles : $x_{t+1} = f(x_t) + \varepsilon_{t+1}$.
- Le deuxième réseau : est un Rbf contenant deux neurones d'entrées (x_t, x_{t-1}), une couche cachée à (n) neurones et un neurone de sortie x_{t+1}.
  Mathématiquement, ce réseau est équivalent au modèles : $x_{t+1} = f(x_t, x_{t-1}) + \varepsilon_{t+1}$.

- Le troisième réseau : est un Rbf contenant trois entrées (x_t, x_{t-1}, x_{t-2}), une couche cachée contenant (c) neurones et un neurone de sortie x_{t+1}.

  Mathématiquement, ce réseau est équivalent au modèle : $x_{t+1}= f(x_t, x_{t-1}, x_{t-2}) + \varepsilon_{t+1}$.

:

- Le nièmes réseau : est un Rbf contenant n entrées (x_t, x_{t-1}, x_{t-2},…, x_t), une couche cachée contenant (c) neurones et un neurone de sortie x_{t+1}.

  Mathématiquement, ce réseau est équivalent au modèle : $x_{t+1}= f(x_t, x_{t-1}, x_{t-2},\ldots, x_t) + \varepsilon_{t+1}$.

XXVII Prévision :

La prévision de X_t au pas h, notée $\hat{X}_{t+h}$ est donnée par :

$$\hat{X}_{t+h} = E(x_{t+h} / x_{t+h-1}, x_{t+h-2}, ...)$$

- Rbf à une entrée : $\hat{X}_{t+h} = f(x_{t+h-1})$
- Rbf à deux entrées : $\hat{X}_{t+h} = f(x_{t+h-1}, x_{t+h-2})$
- Rbf à trois entrées : $\hat{X}_{t+h} = f(x_{t+h-1}, x_{t+h-2}, x_{t+h-3})$

:

:

- Rbf à n entré : $\hat{X}_{t+h} = f(x_{t+h-1}, x_{t+h-2}, x_{t+h-3}, ..., x_{t+h-n})$

XXVIII Etude de la complexité des algorithmes :

La complexité est une mesure de la « difficulté » du calcul d'un algorithme en fonction de la taille n des données. C'est une évaluation du coût d'exécution d'un algorithme en termes de temps (complexité temporelle) ou d'espace (complexité spatiale).
Dans notre cas on s'intéresse seulement a étudie la complexité temporelle qui est le nombre d'opérations élémentaires (affectations, comparaisons, opérations arithmétiques) effectuer par un algorithme. Ce nombre s'exprime en fonction de la taille n des données.
Dans l'algorithme les boucles utilisées sont la boucle **For,** la boucle **While**, et la condition **if.**
Règles du calcul de la complexité « en pire des cas » :

XXVIII.1 Règle-1: Conditionnelle

Soit une action A de la forme « if C alors a_1 Sinon a_2 Fin if » alors :

$T_A(n)= T_C(n) + \max(T_{a1}(n), T_{a2}(n))$ ou a_1= action 1 et a_2= action 2.s cas,

En effet, dans le pire des cas, c'est toujours la plus coûteuse des deux actions qui s'exécute.

XXVIII.2 Règle-2: Itération (While)

Soit une action A de la forme « While C faire A_1 Fin While » alors :

$T_A(n) = \left(\sum_{i=1}^{n} \left(T_C(i,n) + T_{A1}(i,n)\right) \right) + T_C\left(niter(n)+1, n\right)$ Tel que :

T_c : est le nombre de test de la condition C, cette dernière sera exécuté une fois de plus que le corps A1 dans la boucle (test de continuation).

XXVIII.3 Règle-3: Itération (For)

Soit une action A de la forme « For C faire A_1 Fin For » alors :

$T_A(n) = \sum_{i=1}^{n} \left(T_C(i,n) + T_{A1}(i,n)\right)$ Tel que:

T_c : est mis pour l'affectation de l'indice.

Première partie de l'application selon Le principe classique du mode d'apprentissage supervisé

Premiere Méthode

On sait que la fonction noyau du réseau à fonction de base radial (Rbf) est caractérisée par ces deux paramètres (la position de son centre et la largeur du noyau) donc :
Dans cette première méthode de la 1^{ere} partie de l'application les paramètres (centres et rayons) de la fonction noyau (fonction Gaussienne) seront fixés une fois pour toute durant tout le processus d'apprentissage, et une fois les paramètres de la fonction noyau (centres et rayons) sont fixés, l'opération de l'apprentissage se fera uniquement par la modification des poids de connexions entre les neurones.

XXIX Procédure de fixation des centres et des rayons :

XXIX.1 Choix des paramètres des gaussiennes :

Pour couvrir uniformément la partie utile de l'espace d'entrée et respecter ainsi les contraintes de couverture et de généralisation localisée expliquée dans la partie théorique, on a fixé les paramètres des gaussiennes (centres et rayons) une fois pour toutes, de la manière suivante :

XXIX.2 Les centres sont fixés par la méthode des treillis :

Cette méthode consiste à placer les centres des fonctions noyau à égales distances (entre le minimum et le maximum de l'espace des données).

Pour le premier centre : $c_1=\min(x)$

Pour le deuxième centre : $c_2=c_1+a$

:

Pour le n^{ième} centre : $c_n=c_{n-1}+a$, tel qu'a est la distance entre deux centres voisins.

XXIX.3 La largeur des noyaux :

Les valeurs des rayons sont fixées par la distance entre centres voisins (dans notre cas on a fixé la largeur de chaque noyau égale à deux fois la distance entre deux centres voisins), afin d'assurer un certain degré de recouvrement entre les différentes gaussiennes (champs d'influence des différents centres).

La distance entre deux centres successifs est fixée par :

$$a = \frac{d}{\textit{nombre de neurones dans la couche cachée}}$$ Tel que : d= max(x)-min(x).

XXIX.4 L'apprentissage des poids de connexions :

Un processus d'apprentissage dans le cadre des réseaux de neurones artificiels peut être vu comme le problème d'initier le réseau à évoluer d'un état initial vers un état final, cette évolution se traduit par la modification des poids de connexions entre les neurones du réseau. L'apprentissage est supervisé et il se fait par correction d'erreurs, c'est-à-dire :

Minimisation d'une erreur quadratique $E=\frac{1}{2}\sum e_t^2$ où : e= (s (t)-y (t)).

Pour cela on a utilisé deux règles de modification de poids :

XXIX.5 1ere Cas : La règle de Widrow-Hoff

Les poids sont mis à jour par la règle de Widrow-Hoff suivante : $\delta_{ji} = \eta(y_{pi} - S_{pi})x_{pj}$.

Dans notre cas cette règle est équivalente à : $w_{ji}(t+1) = w_{ji}(t) - \alpha(S_{pi}(t) - y_{pi}(t))\ \phi_{pj}(t)$

Où : $w_{ji}(t+1)$: Le poids reliant le neurone j à i à l'instant t+1.

$w_{ji}(t)$: Le poids reliant le neurone j à i à l'instant t.

α : Un paramètre d'adaptation de l'apprentissage (α >0).

$y_{pi}(t)$: La sortie désirée à l'instant t.

p : représente l'exemple courant.

XXIX.5.1 Le choix d'architecture de réseau à fonction de base radiale (Rbf) :

Le but de notre travail est d'utiliser les réseaux de neurones pour calculer la prévision. Pour cela on a choisi l'architecture qui donne une erreur minimale au sens de SCR_p (la somme des carrés des résidus de la prévision).

Donc d'après les expériences qui ont été menées sur plusieurs types d'architectures (voir un exemple dans le tableau ci-dessous) :

	n^{bre} de neurones dans la couche cachée	SCR	SCR_P
Rbf à 2 entrées	**2**	**147.95**	**5.2426**
	3	**138.72**	**4.9699**
	4	**144.68**	**4.8112**
Rbf à 3 entrées	**2**	**6806.1**	**3.241**
	3	**55793**	**1.7576**
	4	**171.94**	**5.0661**
Rbf à 4 entrées	**2**	**30499**	**2.6887**
	3	**136.51**	**4.1552**
	4	**311.95**	**0.74106**

Le réseau Rbf à quatre entrées et a quatre neurones cachés donne le meilleur résultat (SCR_p = **0.74106**).

XXIX.5.2 Réseau à fonction de base radiale à 4 entrées

Taille de la Base d'apprentissage : 144

Taille de la Base de test : 6

Nombres d'itérations : 12

Nombres de neurones dans la couche cachée : 4

Les figures (1, 2,3) illustrent l'apprentissage qui consiste à chercher la configuration de poids qui minimise l'erreur ; ces figures représentent les graphes des fonctions suivantes :

- la série et son ajustement en utilisant les poids de la dernière itération.
- la somme des carrés des résidus de l'ajustement (SCR).
- la somme des carrés des résidus de la prévision (SCRp).

Deuxième itération :

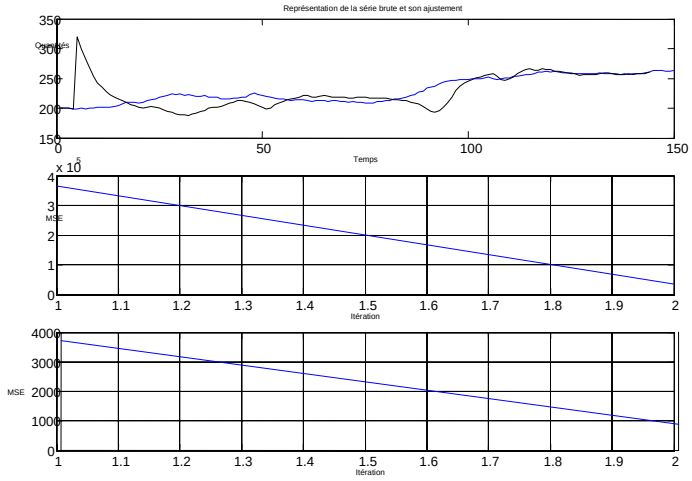


Figure 1

$$SCR = \frac{1}{2}\sum_{t=1}^{144} e_t^2 = 36031$$

$$SCR_p = \frac{1}{2}\sum_{t=145}^{150} e_t^2 = 873.85$$

Itération intermédiaire :

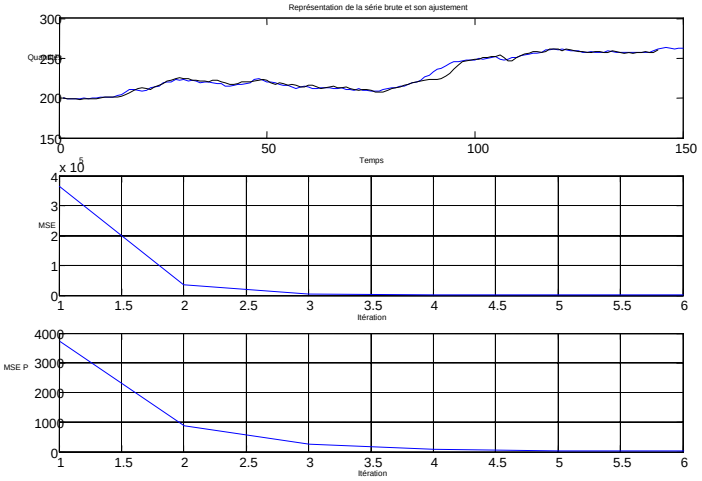


Figure 2

SCR= 649.53

SCRp=9.6029

Dernière itération :

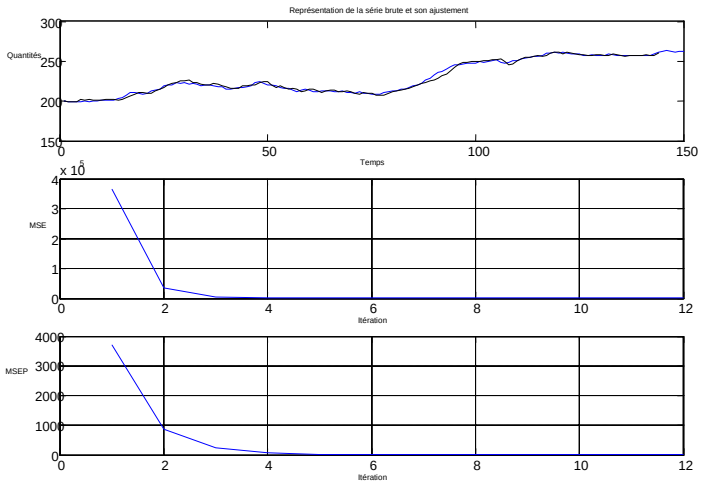


Figure 3

SCR=311.95

SCRp=0.74106

XXIX.5.3 Test d'arrêt :

Théoriquement : Sur la base d'apprentissage, l'erreur diminue constamment, alors que sur la base de test elle passe par un minimum. Si l'apprentissage se prolonge au delà, les performances en test diminuent.

D'après le tableau ci-dessous on remarque que :

L'erreur quadratique de l'ajustement (SCR) diminue sans cesse au cours de l'apprentissage, par contre l'erreur quadratique de la prévision (SCR_P) passe par un minimum a **l'itération 12**, ce qui est conforme à ce qui a été écrit dans la partie théorique. Chose qu'on a pas pu voir graphiquement, ce qui peut être expliqué par :

- taille de l'échantillon (faible).
- la série chronologique est régulière.

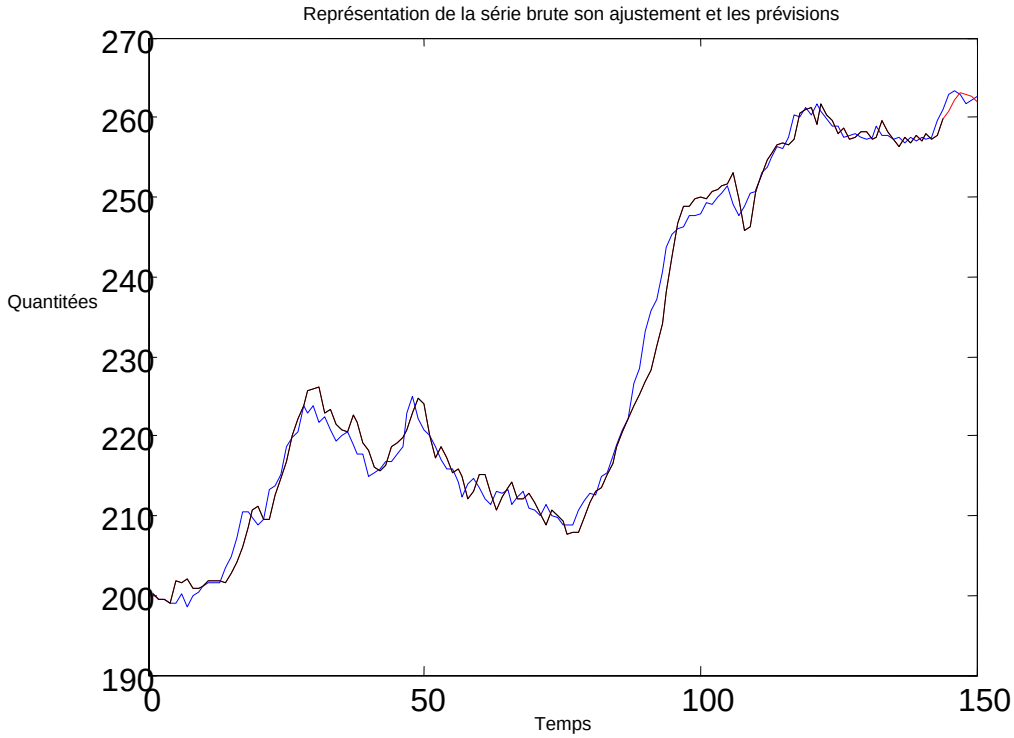
SCRP	SCR	ITERATION
26.395	912.18	5
9.6029	649.53	6
3.7346	506.26	7
1.6879	424.19	8
1.0076	375.88	9
0.80273	345.99	10
0.75026	326.17	11
0.74106	**311.95**	**12**
0.74634	300.96	13
0.76539	291.93	14
0.80514	284.17	15
0.87264	277.3	16
0.97271	271.07	17
1.1076	265.36	18

XXIX.5.4 Prévision :

BRUT	PREVI	ERRE
262.9	260.7446	2,1554
263.3	262.2354	1,0646
262.8	263.0655	-0,2655
261.8	262.9180	-1,118
262.2	262.6590	-0,459
262.7	261.8724	0,8276

D’après le tableau ci-dessus, on remarque que les écarts entres les prévisions obtenues par le réseau de neurones et les valeurs de série Brut sont faibles, d’ailleurs on peut le constater graphiquement (graphe ci-dessous).

On constat aussi d'après le graphe ci-dessous que l'écart entre la série d'ajustement et la série brut dans certaines région est faible par rapport à d'autres régions.



Graphe de la série avec son ajustement et la prévision.

XXIX.5.5 Etude de la complexité de cet algorithme :

Le programme qui a été utilisé dans l'application est constitué d'une fonction principale qui la fonction RBF, cette dernière et au cours de son exécution fait appelle à deux autres fonctions :

La fonction treillis qui sera exécuté une fois au début de l'exécution du programme, et la fonction prévision qui sera exécuté dans la boucle While qui existe dans la fonction RBF.

XXIX.5.5.1 Calcul de la complexité de la fonction prévision :

Cette fonction est constituée d'une boucle « For » a son intérieur une autre boucle « For » imbriquée, et des opérations indépendantes des boucles.

XXIX.5.5.1.1 Complexité de la boucle For :

Cette dernière contient à l'intérieur une boucle imbriquée, et d'autres opérations indépendantes de la boucle imbriquée, et la formule pour calculer sa complexité est :

$$T_{Al}(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)+774\right)$$ Ou :

T_1 : Est la complexité de la boucle « For » imbriquée.

T_c : Nombre d'affectation d'indice.

774 : Nombre d'opérations dans la boucle « For » indépendamment de la boucle imbriquée.

Donc : la complexité de la boucle imbriquée est :

$$T_1(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)\right)=\sum_{i=1}^{n}(1+19)=\sum_{i=1}^{n}20=20\text{n.}$$

La complexité de la boucle « For » est :

$$T_{Al}(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)+774\right)=T_{Al}(n)=\sum_{i=1}^{n}(1+20n+774)=\sum_{i=1}^{n}1+20n\sum_{i=1}^{n}1+774\sum_{i=1}^{n}1=$$

$$20n^2+775n$$

XXIX.5.5.1.2 La complexité totale de la fonction prévision :

On a : 35 est le nombre d'opérations en dehors de la boucle « For » donc la complexité total de la fonction prévision est :

Op_p (n)$=35+T_{Al}(n)=35+775n+20n^2$

XXIX.5.5.2 Calcul de la complexité de la fonction Treillis :

Cette fonction est constituée de plusieurs opérations indépendamment des boucles, une boucle « For » et d'une grande boucle « For » qui contiennent 4 autres boucles « For » imbriquées comme suite :

La boucle n°4 est imbriquée dans la boucle n°3 qui est imbriquée dans la boucle n°2 cette dernière est imbriquée dans la boucle n°1 et les quatre boucles sont imbriquées dans la grande boucle « For ».

Soit :

Op_1 (n) le nombre des opérations dans la grande boucle « For ».

T_1 (i, n) le nombre des opérations dans la boucle « For » n°1.

T_2 (i, n) le nombre des opérations dans la boucle « For » n°2.

T_3 (i, n) le nombre des opérations dans la boucle « For » n°3.

T_4 (i, n) le nombre des opérations dans la boucle « For » n°4.

XXIX.5.5.2.1 Calcul du nombre d'opérations dans la grande boucle « For » :

T_4 (i, n)= $\sum_{i=1}^{n} 6 = 6n$

T_3 (i, n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n} 6)+1+1) = \sum_{i=1}^{n}(6n+2) = n(6n+2) = 2n+6n^2$

T_2 (i, n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n} 6)+1+1))+1) = \sum_{i=1}^{n}(2n+6n^2+1) = n+2n^2+6n^3$

T_1 (i, n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n} 6)+1+1))+1))+1) = \sum_{i=1}^{n}(n+2n^2+6n^3+1) = n+n^2+2n^3+6n^4$

Donc le nombre d'opérations dans la grande boucle « For » est :

Op_1 (n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n} 6)+1+1))+1))+1))+1) = \sum_{i=1}^{n}(1+n+n^2+2n^3+6n^4)$

Op_1 (n)= $n+n^2+n^3+2n^4+6n^5$

XXIX.5.5.2.2 Calcul d'opérations en dehors de la grande boucle « For » :

Soit Op_2 (n) le nombre des opérations dans la fonction treillis en dehors de la grande boucle « For ».

T (n)= $\sum_{i=1}^{n}(T_C(i,n)+T(i,n))$ est le nombre d'opérations dans la boucle « For » qui est indépendante de la grande boucle « For ».

6 : est le nombre d'opérations indépendamment de toutes les boucle qui existent dans la fonction treillis.

Op_2 (n)= 6+T (n)=6+ $\sum_{i=1}^{n} 2 = 6+2n$

Donc le nombre d'opérations total dans la fonction treillis est :

Op_t (n)= Op_1 (n) + Op_2 (n)= $n+n^2+n^3+2n^4+6n^5+6+2n = 6+3n+n^2+n^3+2n^4+6n^5$

XXIX.5.5.3 Calcul de la complexité de la fonction RBF :

XXIX.5.5.3.1 Calcul du nombre d'opérations dans la boucle While :

Soit T_A (n) le nombre d'opérations dans la boucle While tel

que : $T_A(n) = (\sum_{i=1}^{n}(T_C(i,n)+T_{A1}(i,n)+T_{A2}(i,n)+T_{A3}(i,n)+300+op_p(n)+1)) + T_C(n+1,n)$

Tel que :

T_c : Nombre de test de la condition C.

T_{A1} : Nombre d'opérations dans la grande boucle For, cette dernière contient a l'intérieur deux boucles « For » imbriquées (T_1 et T_2).

T_{A2} : Nombre d'opération dans la condition « if ».

T_{A3} : Nombre d'opération dans la condition « if ».

296 : Nombre d'opérations dans la boucle While indépendamment des boucles imbriquées.

Op_p (n) : Nombre d'opération dans la fonction prévision.

1 : représente l'appel de la fonction prévision car l'appel est compté comme une opération.

On a :

$T_C(i,n) = 1.$

XXIX.5.5.3.2 Calcul du nombre d'opération dans la grande boucle « For » :

Cette dernière contient à l'intérieur deux boucles imbriquées, et d'autres opérations indépendantes de ces dernières, donc notre formule devient :

$$T_{Al}(n) = \sum_{i=1}^{n} \left(T_C(i,n) + T_1(i,n) + T_2(i,n) + 770\right) \text{ Ou :}$$

T_1 : Est le nombre d'opérations dans la première boucle For imbriquée.

T_2 : Est le nombre d'opérations dans la deuxième boucle For imbriquée.

T_c : Nombre d'affectation d'indice.

770 : Nombre d'opérations dans la grande boucle « For » indépendamment des deux autres boucles imbriquées.

Pour la 1ere boucle « For » imbriqué on a :

$$T_1(n) = \sum_{i=1}^{n} \left(T_C(i,n) + T_1(i,n)\right) = \sum_{i=1}^{n} (1+19) = \sum_{i=1}^{n} 20 = 20n.$$

Pour la 2ere boucle « For » imbriqué on a :

$$T_2(n) = \sum_{i=1}^{n} \left(T_C(i,n) + T_1(i,n)\right) = \sum_{i=1}^{n} (1+4) = \sum_{i=1}^{n} 5 = 5n.$$

Donc le nombre d'opération dans la grande boucle « **For** » est:

$$T_{Al}(n) = \sum_{i=1}^{n} \left(T_C(i,n) + T_1(i,n) + T_2(i,n) + 770\right) = \sum_{i=1}^{n} (1 + 20n + 5n + 770) =$$

$$\sum_{i=1}^{n} 1 + 20n \sum_{i=1}^{n} 1 + 5n \sum_{i=1}^{n} 1 + 770 \sum_{i=1}^{n} 1 \quad \text{donc} : T_{Al}(n) = n + 20n^2 + 5n^2 + 770n = 25n^2 + 771n$$

En dehors de la grande boucle « For » on trouve deux conditions « if » :

XXIX.5.5.3.3 Le nombre d'opérations dans la 1ere condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

$T_{A2}(n) = T_C(n) + \text{Max}(T_{a1}(n), T_{a2}(n))$.

Dans le cas de notre algorithme l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'affectations, donc notre équation devienne :

$T_{A2}(n) = T_C(n) + T_{a1}(n)$.

$T_{A2}(n) = 1+7=8$

XXIX.5.5.3.4 Le nombre d'opérations dans la 2eme condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

$T_{A3}(n) = T_C(n) + \text{Max}(T_{a1}(n), T_{a2}(n))$.

De même pour la deuxième boucle « if » l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'instructions, donc notre équation devienne :

$T_{A3}(n) = T_C(n) + T_{a1}(n)$.

$T_{A3}(n) = 1 +12=13$.

Donc le nombre total d'opérations dans la boucle While est :

$$T_A(n) = \left(\sum_{i=1}^{n} \left(T_C(i,n) + T_{A1}(i,n) + T_{A2}(i,n) + T_{A3}(i,n) + 298 + op_p(n) + 1 \right) \right) + T_C(n+1,n) =$$

$$\left(\sum_{i=1}^{n} (1 + 25n^2 + 771n + 8 + 13 + 296 + 35 + 775n + 20n^2 + 1)) \right) + 1 = \left(\sum_{i=1}^{n} (354 + 1546n + 45n^2)) \right) + 1$$

$$T_A(n) = 1 + 354n + 1546n^2 + 45n^3$$

Le nombre total d'opération **Op(n)** de la fonction principale RBF qui est aussi le nombre total d'opérations effectuées dans l'exécution de l'algorithme:

$Op(n) = (Op_t(n)+1)+T_A(n)+421$ tel que :

Op_t (n) : est le nombre d'opérations dans la fonction Treillis

T_A (n) : est le nombre d'opérations dans la boucle While de la fonction RBF.

421 : est le nombre d'opérations dans la fonction RBF indépendamment des boucle et des autres fonctions.

1 : est l'opération d'appel de la fonction Treillis dans la fonction RBF car la fonction treillis est appelé qu'une seule fois dans l'exécution de l'algorithme.

Donc :

Op(n)= $6+3n + n^2 + n^3 + 2n^4 + 6n^5 + 1 + 1 + 354n + 1546n^2 + 45n^3 + 421$

Op(n)= $429 + 359n + 1547n^2 + 46n^3 + 2n^4 + 6n^5$

On a:

$$\lim_{n\to\infty}(429+357n+1547n^2+46n^3+2n^4+6n^5)/n^5=6\neq 0\Rightarrow Op(n) \quad est\ en \quad \Theta(n^5)$$

XXIX.5.5.4 Conclusion:

D'après les résultats ci-dessus qui ont été obtenus, l'algorithme qui a été utilisé dans cette application est d'une complexité polynomiale d'ordre 5 c'est-à-dire que notre algorithme est en $O(n^5)$.

XXIX.6 2ere Cas : Algorithme de rétro propagation du Gradient

Dans cette section on utilisera l'algorithme de rétro propagation du Gradient pour la mise à jour des poids de connexions entre les neurones et cela est comme suite :

$w_{ji}(t+1)=w_{ji}(t)-\alpha(S_{pi}(t)-y_{pi}(t))\ \phi_{pj}(t)+\gamma\left[w_{ji}(t-1)-w_{ji}(t)\right]$ Tel que:

: $w_{ji}(t+1)$: Le poids reliant le neurone j à i à l'instant t+1.

$w_{ji}(t)$: Le poids reliant le neurone j à i à l'instant t.

$w_{ji}(t-1)$: Le poids reliant le neurone j à i à l'instant t-1.

α : Un paramètre d'adaptation de l'apprentissage (α >0).

$y_{pi}(t)$: La sortie désirée à l'instant t.

p : représente l'exemple courant.

γ **:** est un facteur d'oubli inférieur à 1.

XXIX.6.1 Le choix d'architecture de réseau à fonction de base radiale (Rbf) :

Le choix de l'architecture du réseau se fait de la même façon que la partie de l'application ci-dessus, c'est-à-dire qu'on choisi l'architecture qui donne une erreur minimale au sens de SCR_p (la somme des carrés des résidus de la prévision) car notre but est de calculé une prévision.

Donc d'après les expériences qui ont été menées sur plusieurs types d'architectures (voir un exemple dans le tableau ci-dessous) :

	n^{bre} de neurones dans la couche cachée	SCR	SCR_P
Rbf à 2 entrées	**2**	**3333.5**	**0.19667**
	3	**650.84**	**1.4327**
	4	**1455.6**	**1.3361**
Rbf à 3 entrées	**2**	**831.19**	**1.2386**
	3	**1075.2**	**0.5564**
	4	**397.88**	**1.8077**
Rbf à 4 entrées	**2**	**550.96**	**0.41191**
	3	**4308.7**	**0.48087**
	4	**265.9**	**2.3757**

Le réseau Rbf à deux entrées et a deux neurones cachés donne le meilleur résultat (SCR_p = **0.19667**).

XXIX.6.2 Réseau à fonction de base radiale à 2 entrées

Taille de la Base d'apprentissage : 144

Taille de la Base de test : 6

Nombres d'itérations : 22

Nombres de neurones dans la couche cachée : 2

Les figures (1, 2,3) illustrent l'apprentissage qui consiste à chercher la configuration de poids qui minimise l'erreur ; ces figures représentent les graphes des fonctions suivantes :

- la série et son ajustement en utilisant les poids de la dernière itération.
- la somme des carrés des résidus de l'ajustement (SCR).
- la somme des carrés des résidus de la prévision (SCRp).

Deuxième itération :

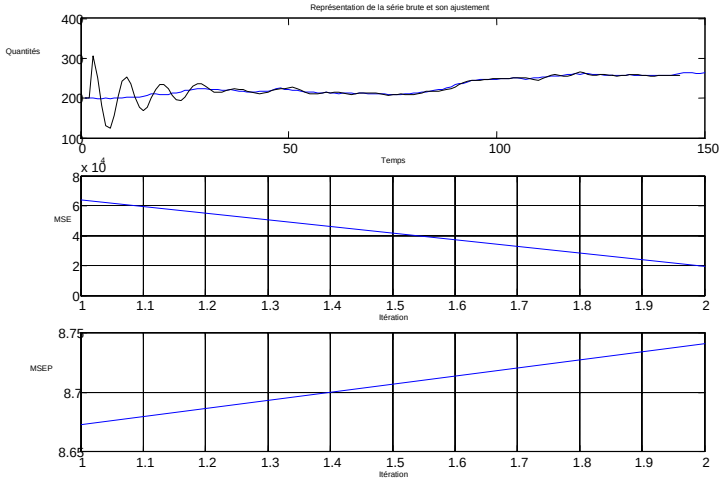


Figure 1

$$SCR = \frac{1}{2}\sum_{t=1}^{144} e_t^2 = 19506$$

$$SCR_p = \frac{1}{2}\sum_{t=145}^{150} e_t^2 = 8.741$$

Itération intermédiaire :

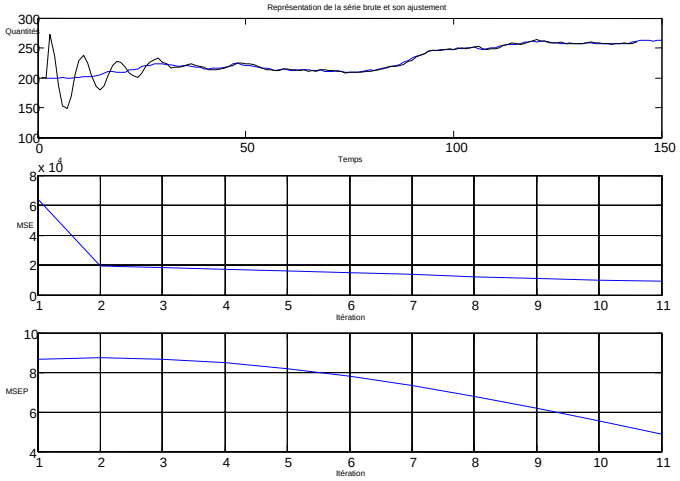


Figure 2

SCR= 9124.2

SCRp= 4.8901

Dernière itération :

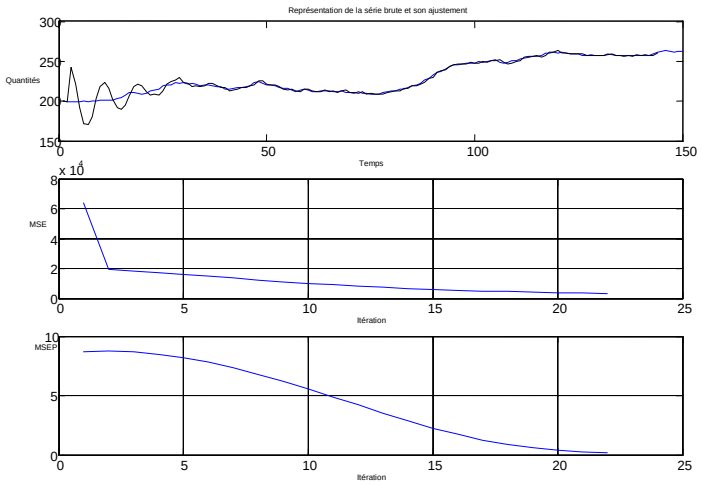


Figure 3

SCR=3333.5

SCRp= **0.19667**

XXIX.6.3 Test d'arrêt :

Théoriquement : Sur la base d'apprentissage, l'erreur diminue constamment, alors que sur la base de test elle passe par un minimum. Si l'apprentissage se prolonge au delà, les performances en test diminuent.

D'après le tableau ci-dessous on remarque que :

L'erreur quadratique de l'ajustement (SCR) diminue sans cesse au cours de l'apprentissage, par contre l'erreur quadratique de la prévision (SCR_P) passe par un minimum a **l'itération 22**, ce qui est conforme à ce qui a été écrit dans la partie théorique. Chose qu'on a pas pu voir graphiquement, ce qui peut être expliqué par :

- taille de l'échantillon (faible).
- la série chronologique est régulière.

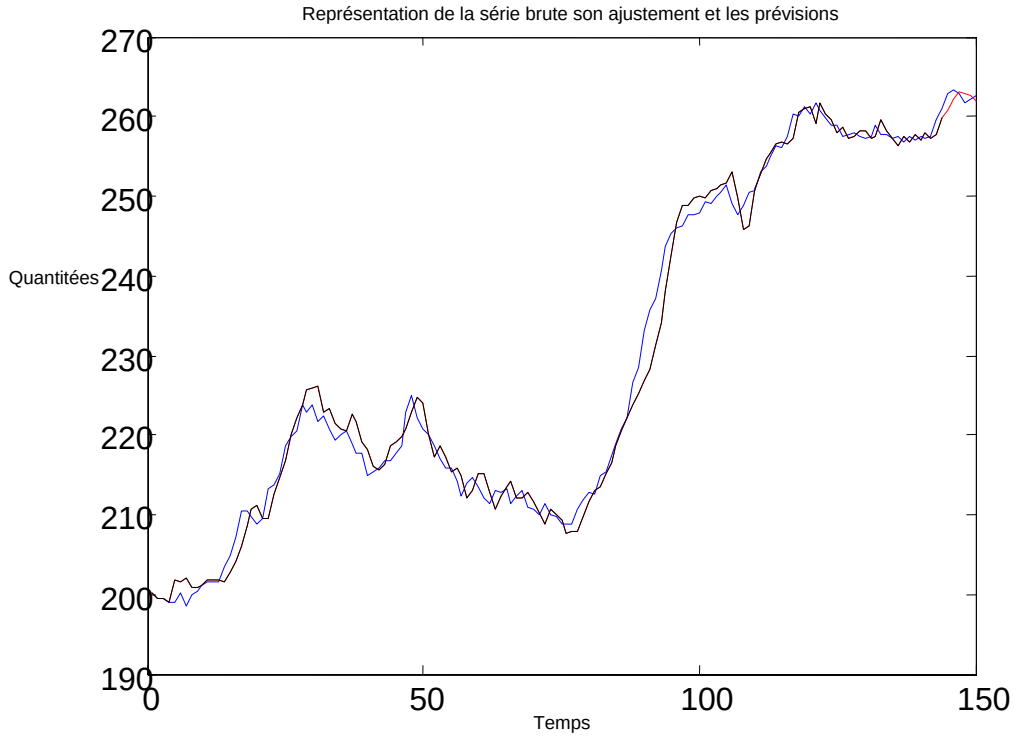
SCRP	SCR	ITERATION
1.7272	5463.9	16
1.262	4965.6	17
0.88133	4532.6	18
0.58833	4158.8	19
0.38126	3838.1	20
0.25394	3564.7	21
0.19667	**3333.5**	**22**
0.19747	3139.3	23
0.24321	2977.9	24
0.32079	2845.2	25
0.41796	2737.7	26
0.52405	2652.2	27
0.63026	2586.1	28
0.72987	2536.9	29

XXIX.6.4 Prévision :

BRUT	PREVI	ERRE
262.9	262.9443	-0,0443
263.3	263.3602	-0,0602
262.8	262.6724	0,1276
261.8	262.1364	-0,3364
262.2	262.0644	0,1356
262.7	262.2101	0,4899

D’après le tableau ci-dessus, on remarque que les écarts entres les prévisions obtenues par le réseau de neurones et les valeurs de série Brut sont très faibles, les valeurs sont presque confondues, d’ailleurs on peut le constater graphiquement (graphe ci-dessous).

On constat aussi d’après le graphe ci-dessous que l‘écart entre la série d’ajustement et la série brut dans certaines région est faible par rapport à d’autres régions.



Graphe de la série avec son ajustement et la prévision.

XXIX.6.5 Etude de la complexité de cet algorithme :

De la même manière que ci-dessus on détermine la complexité de cet algorithme.

XXIX.6.5.1 Calcul de la complexité de la fonction prévision :

Cette fonction est constituée d'une boucle « For » à son intérieur une autre boucle « For » imbriquée, et des opérations indépendantes des boucles.

XXIX.6.5.1.1 Complexité de la boucle For :

Cette dernière contient à l'intérieur une boucle imbriquée, et d'autres opérations indépendantes de la boucle imbriquée, et la formule pour calculer sa complexité est :

$$T_{Al}(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)+21\right)$$ Ou :

T_1 : Est la complexité de la boucle « For » imbriquée.

T_c : Nombre d'affectation d'indice.

21 : Nombre d'opérations dans la boucle « For » indépendamment de la boucle imbriquée.

Donc : la complexité de la boucle imbriquée est :

$$T_1(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)\right)=\sum_{i=1}^{n}(1+11)=\sum_{i=1}^{n}12=12\text{n}.$$

La complexité de la boucle « For » est :

$$T_{A1}(n)=\sum_{i=1}^{n}(T_C(i,n)+T_1(i,n)+21)=T_{A1}(n)=\sum_{i=1}^{n}(1+12n+21)=\sum_{i=1}^{n}1+12n\sum_{i=1}^{n}1+21\sum_{i=1}^{n}1=$$

$$n+12n^2+21n$$

XXIX.6.5.1.2 La complexité totale de la fonction prévision :

On a : 28 est le nombre d'opérations en dehors de la boucle « For » donc la complexité total de la fonction prévision est :

Op_p (n)=$30+T_{A1}(n)=30+22n+12n^2$

XXIX.6.5.2 Calcul de la complexité de la fonction Treillis :

Cette fonction est constituée de plusieurs opérations indépendamment des boucles, une boucle « For » et d'une grande boucle « For » qui contiennent 2 autres boucles « For » imbriquées comme suite :

La boucle n°2 est imbriquée dans la boucle n°1, et les deux boucles sont imbriquées dans la grande boucle « For ».

Soit :

Op_1 (n) le nombre des opérations dans la grande boucle « For ».

T_1 (i, n) le nombre des opérations dans la boucle « For » n°1.

T_2 (i, n) le nombre des opérations dans la boucle « For » n°2.

XXIX.6.5.2.1 Calcul du nombre d'opérations dans la grande boucle « For » :

T_2 (i, n)= $\sum_{i=1}^{n}4=4n$

T_1 (i, n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n}4)+1+1)=\sum_{i=1}^{n}(4n+2)=n(4n+2)=2n+4n^2$

Donc le nombre d'opérations dans la grande boucle « For » est :

Op_1 (n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}4)+1+1))+1)=\sum_{i=1}^{n}(2n+4n^2+1)=n+2n^2+4n^3$

Op_1 (n)= $n+2n^2+4n^3$

XXIX.6.5.2.2 Calcul d'opérations en dehors de la grande boucle « For » :

Soit Op_2 (n) le nombre des opérations dans la fonction treillis en dehors de la grande boucle « For ».

T (n)= $\sum_{i=1}^{n}(T_C(i,n)+T(i,n))$ est le nombre d'opérations dans la boucle « For » qui est indépendante de la grande boucle « For ».

6 : est le nombre d'opérations indépendamment de toutes les boucle qui existent dans la fonction Treillis.

Op_2 (n)= 6+T (n)=6+ $\sum_{i=1}^{n}3$ =6+ $3n$

Donc le nombre d'opérations total dans la fonction Treillis est :

Op_t (n)= Op_1 (n) + Op_2 (n)= $n+2n^2+4n^3+6+3n=6+4n+2n^2+4n^3$

XXIX.6.5.3 Calcul de la complexité de la fonction RBF :

XXIX.6.5.3.1 Calcul du nombre d'opérations dans la boucle While :

Soit T_A (n) le nombre d'opérations dans la boucle While tel que :

$T_A(n)=\left(\sum_{i=1}^{n}\left(T_C(i,n)+T_{A1}(i,n)+T_{A2}(i,n)+T_{A3}(i,n)+300+op_p(n)+1\right)\right)+T_C(n+1,n)$ Tel que :

T_c : Nombre de test de la condition C.

T_{A1} : Nombre d'opérations dans la grande boucle For, cette dernière contient a l'intérieur deux boucles « For » imbriquées (T_1 et T_2).

T_{A2} : Nombre d'opérations dans la condition « if ».

T_{A3} : Nombre d'opérations dans la condition « if ».

300 : Nombre d'opérations dans la boucle While indépendamment des boucles imbriquées.

Op_p (n) : Nombre d'opérations dans la fonction prévision.

1 : représente l'appel de la fonction prévision car l'appel est compté comme une opération.

On a :

$T_C(i,n)=$ 1.

XXIX.6.5.3.2 Calcul du nombre d'opération dans la grande boucle « For » :

Dans cet algorithme la grande boucle « For » est constituée d'une action conditionnelle de la forme : if (condition) action1 else action2 end if.

Donc d'après les formules illustrées ci-dessus le nombre d'opérations dans la grande boucle « For » est :

$T_{A1}(n)= \sum_{i=1}^{n}(T_C (n) + Ts)= \sum_{i=1}^{n}(T_C (n) + (1+ Max (Ta_1 (n), Ta_2 (n))))$ tel que :

Ta_1 : Est le nombre d'opérations dans l'action1 (a1).

Ta_2 : Est le nombre d'opérations dans l'action2 (a2).

T_c : Nombre d'affectation d'indice de la grande boucle « For ».

Ts : Est le nombre d'opérations dans la structure if (condition) action1 else action2 end if.

Soit :

$Ts=1+ Max (Ta_1 (n), Ta_2 (n))$

$Ta_1(n)= T_1 (n) +T_2 (n) +T_3 (n)$ tel que :

T_1 : est le nombre d'opération dans la première boucle de l'action 1.

T_2 : est le nombre d'opération dans la deuxième boucle de l'action 1.

T_3 : est le nombre d'opération en dehors des deux boucles «For » de l'action 1.

Donc :

$Ta_1(n) = (\sum_{i=1}^{n}(T_{C1}(i,n)+T_1(i,n)))+(\sum_{i=1}^{n}(T_{C2}(i,n)+T_2(i,n)))+T_3(i,n)$ Tel que :

$Ta_1=19+ \sum_{i=1}^{n}12 +\sum_{i=1}^{n}5 =19+12n+5n=19+17n.$

De même pour la deuxième action :

$Ta_2(n) = (\sum_{i=1}^{n}(T_{C1}(i,n)+T_1(i,n)))+(\sum_{i=1}^{n}(T_{C2}(i,n)+T_2(i,n)))+T_3(i,n)$

$Ta_2=19+ \sum_{i=1}^{n}12 +\sum_{i=1}^{n}8 =19+12n+8n=19+20n.$

Donc Ts=1+Max ((19+17n), (19+20n))=1+19+20n =20+20n

Ts=20+20n

Le nombre d'opérations dans la grande boucle « For » est :

$T_{A1}(n)= \sum_{i=1}^{n}(T_C (n) + (1+Max (Ta_1 (n), Ta_2 (n))))= \sum_{i=1}^{n}(1+ 20+20n)$

$T_{A1}(n)= \sum_{i=1}^{n}21+\sum_{i=1}^{n}20n = 21n + 20n^2$

En dehors de la grande boucle « For » on trouve deux conditions « if » :

XXIX.6.5.3.3 Le nombre d'opérations dans la 1ere condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

$T_{A2}(n) = T_C(n) + \text{Max}(T_{a1}(n), T_{a2}(n))$.

Dans le cas de notre algorithme l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'affectations, donc notre équation devienne :

$T_{A2}(n) = T_C(n) + T_{a1}(n)$.

$T_{A2}(n) = 1+7=8$

XXIX.6.5.3.4 Le nombre d'opérations dans la 2eme condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

$T_{A3}(n) = T_C(n) + \text{Max}(T_{a1}(n), T_{a2}(n))$.

De même pour la deuxième boucle « if » l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'instructions, donc notre équation devienne :

$T_{A3}(n) = T_C(n) + T_{a1}(n)$.

$T_{A3}(n) = 1 +12=13$.

Donc le nombre total d'opérations dans la boucle While est :

$$T_A(n) = \left(\sum_{i=1}^{n} \left(T_C(i,n) + T_{A1}(i,n) + T_{A2}(i,n) + T_{A3}(i,n) + 300 + op_p(n) + 1 \right) \right) + T_C(n+1,n) =$$

$$\left(\sum_{i=1}^{n} (1 + 21n + 20n^2 + 8 + 13 + 300 + 30 + 22n + 12n^2 + 1)) \right) + 1 = \left(\sum_{i=1}^{n} (353 + 43n + 32n^2)) \right) + 1$$

$$T_A(n) = 1 + 353n + 43n^2 + 32n^3$$

Le nombre total d'opération **Op(n)** de la fonction principale RBF qui est aussi le nombre total d'opérations effectuées dans l'exécution de l'algorithme est:

$Op(n) = (Op_t(n)+1)+T_A(n)+174$ tel que :

Op_t (n) : est le nombre d'opérations dans la fonction Treillis

T_A (n) : est le nombre d'opérations dans la boucle While de la fonction RBF.

174 : est le nombre d'opérations dans la fonction RBF indépendamment des boucle et des autres fonctions.

1 : est l'opération d'appel de la fonction Treillis dans la fonction RBF car la fonction treillis est appelé qu'une seule fois dans l'exécution de l'algorithme.

Donc :

Op(n)$= 1 + 6 + 4n + 2n^2 + 4n^3 + 1 + 353n + 43n^2 + 32n^3 + 174$

Op(n)$= 182 + 357n + 45n^2 + 36n^3$

On a :

$$\lim_{n\to\infty}(182+357n+45n^2+36n^3)/n^3=36\neq 0\Rightarrow Op(n) \quad est\ en \quad \Theta(n^3)$$

XXIX.6.5.4 Conclusion:

D'après les résultats ci-dessus qui ont été obtenus, l'algorithme qui a été utilisé dans cette application est d'une complexité polynomiale d'ordre 3, c'est-à-dire que notre algorithme est en $O(n^3)$.

Deuxieme Méthode

Dans cette deuxième méthode de la 1ere partie de l'application les paramètres (centres et rayons) qui caractérise la fonction noyau ne seront pas fixes durant le processus d'apprentissage, c'est-à-dire que l'apprentissage ne se fera pas seulement par la modification des poids de connexions entre les neurones mais il se fera aussi par la modification de la positions des centres et la modification de la largeur des noyaux.
Donc, dans cette méthode l'apprentissage se fera par l'ajustement de trois paramètres (W, C et σ) au lieu d'un seul paramètre (w) dans la première méthode.

XXX Apprentissage des centres (c_i) et des rayons (σ_i):

L'ajustement des centres et des rayons peut être aisément effectué au moyen d'un algorithme de type "descente de gradient" comme le montreront les équations ci-dessous :

Soit : $J=\frac{1}{2}\left(y-y^*\right)^2$ le critère a minimiser.

Posons : $R(u)=\sum_{i=1}^{n}\rho_i(u)$ (facteur de normalisation).

$$\frac{\partial J}{\partial w_j}=(y-y^*)\frac{\rho_j(u)}{R(u)}$$

$$\frac{\partial J}{\partial c_{j,k}}=\frac{\partial J}{\partial \rho_j}\frac{\partial \rho_j}{\partial c_{j,k}}$$

$$\frac{\partial J}{\partial \sigma_{j,k}}=\frac{\partial J}{\partial \rho_j}\frac{\partial \rho_j}{\partial \sigma_{j,k}}$$

$$\frac{\partial J}{\partial \rho_j}=\left(y\_y^*\right)\left[\frac{w_j\ R(u)-\sum_{i=1}^{n}\rho_i(u)w_i}{R^2(u)}\right]$$

$$\frac{\partial \rho_j}{\partial c_{j,k}}=\rho_j.\frac{u_k-c_{j,k}}{\sigma_{j,k}^2}$$

$$\frac{\partial \rho_j}{\partial \sigma_{j,k}}=\rho_j.\frac{\left(u_k-c_{j,k}\right)^2}{\sigma_{j,k}^3}$$

Les corrections s'obtiennent par:

$$\Delta w_j=-\eta_w\frac{\partial J}{\partial w_j}$$

$$\Delta c_{j,k}=-\eta_c\frac{\partial J}{\partial c_{j,k}}$$

$$\Delta\sigma_{j,k} = -\eta_{\sigma}\frac{\partial J}{\partial \sigma_{j,k}}$$

XXX.11ere Cas d'application :

Dans ce premier cas d'application l'apprentissage se fera sur les trois paramètres W, C et σ comme suite :

L'ajustement des Centres et des rayons (Ci, σ_i) s'effectue par la méthode de descente de gradient qui a été illustrée ci-dessus.

L'ajustement des poids W_i se fera par la règle de Widdrow-Hoff définie dans la première méthode ci-dessus.

XXX.1.1 Le choix d'architecture de réseau à fonction de base radiale (Rbf) :

Le choix de l'architecture du réseau se fait de la même façon que la première méthode de l'application ci-dessus, c'est-à-dire qu'on choisi l'architecture qui donne une erreur minimale au sens de SCR_p (la somme des carrés des résidus de la prévision) car notre but est de calculé une prévision.

Donc d'après les expériences qui ont été menées sur plusieurs types d'architectures (voir un exemple dans le tableau ci-dessous) :

	nbre de neurones dans la couche cachée	**SCR**	**SCR P**
Rbf à 2 entrées	**2**	**145.44**	**5.553**
	3	**919.35**	**1.3454**
	4	**1095.2**	**1.4972**
Rbf à 3 entrées	**2**	**3557.3**	**2.1573**
	3	**136.51**	**6.0837**
	4	**272.77**	**1.933**
Rbf à 4 entrées	**2**	**3336.4**	**0.65755**
	3	**177.47**	**6.0171**
	4	**360.41**	**4.9328**

Le réseau Rbf à deux entrées et a deux neurones cachés donne le meilleur résultat (SCR_p = **0.65755**).

XXX.1.2 Réseau à fonction de base radiale à 4 entrées

Taille de la Base d'apprentissage : 144

Taille de la Base de test : 6

Nombres d'itérations : 3

Nombres de neurones dans la couche cachée : 2

Les figures (1, 2,3) illustrent l'apprentissage qui consiste à chercher la configuration de poids qui minimise l'erreur ; ces figures représentent les graphes des fonctions suivantes :

- la série et son ajustement en utilisant les poids de la dernière itération.
- la somme des carrés des résidus de l'ajustement (SCR).
- la somme des carrés des résidus de la prévision (SCRp).

Deuxième itération :

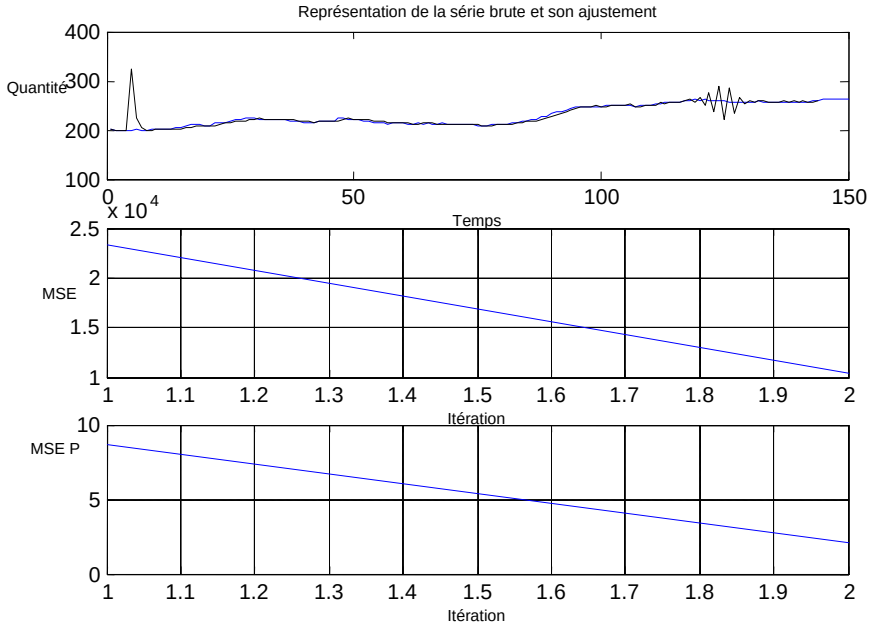


Figure 1

$$SCR = \frac{1}{2}\sum_{t=1}^{144} e_t^2 = 10468$$

$$SCR_p = \frac{1}{2}\sum_{t=145}^{150} e_t^2 = 2.0832$$

Itération intermédiaire :

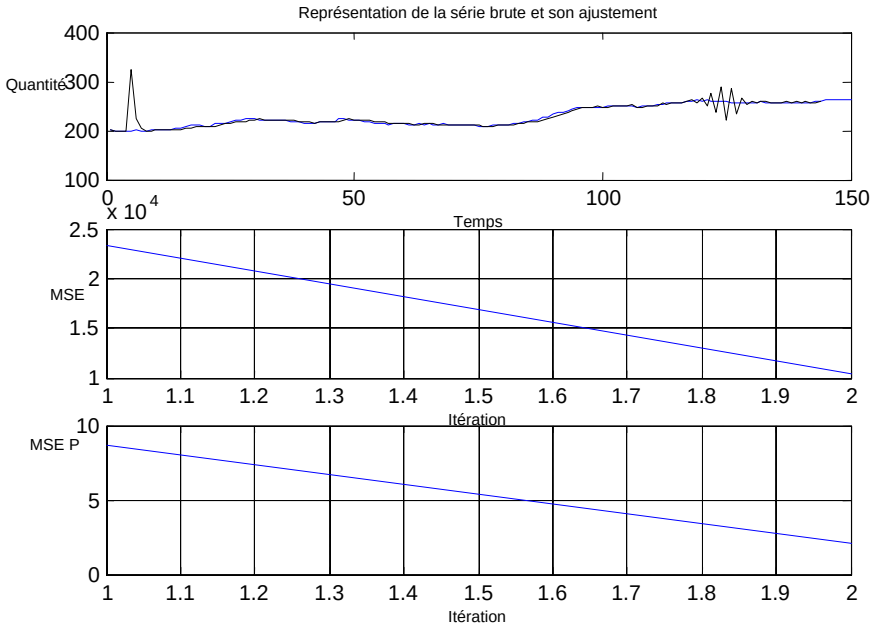


Figure 2

SCR= 10468

SCRp= 2.0832

Dernière itération :

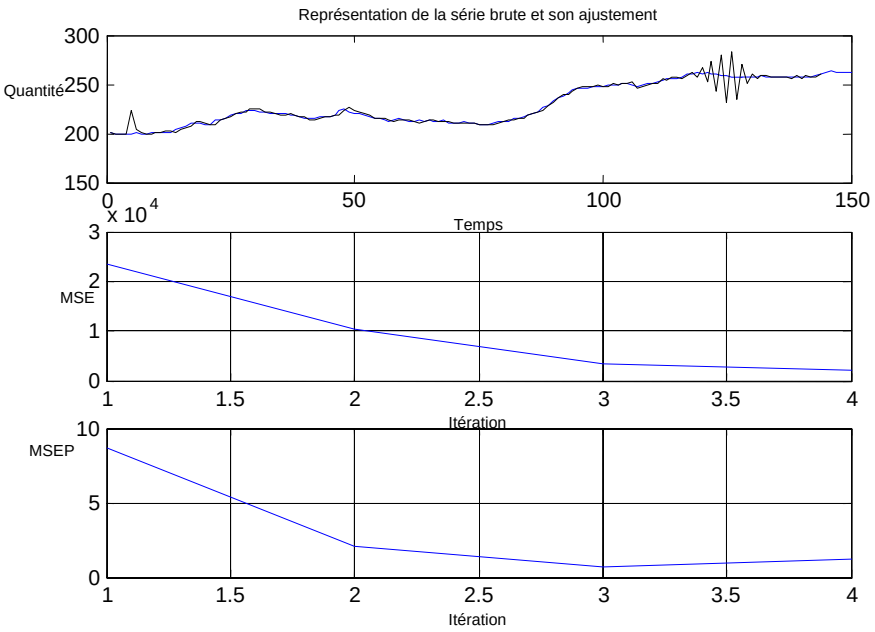


Figure 3

SCR= 3336.4

SCRp= **0.65755**

XXX.1.3 Test d'arrêt :

Théoriquement : Sur la base d'apprentissage, l'erreur diminue constamment, alors que sur la base de test elle passe par un minimum. Si l'apprentissage se prolonge au delà, les performances en test diminuent.

D'après le tableau ci-dessous on remarque que :

L'erreur quadratique de l'ajustement (SCR) diminue sans cesse au cours de l'apprentissage, par contre l'erreur quadratique de la prévision (SCR_P) passe par un minimum a **l'itération 3**, ce qui est conforme à ce qui a été écrit dans la partie théorique. Chose qu'on a pas pu voir graphiquement, ce qui peut être expliqué par :

- taille de l'échantillon (faible).
- la série chronologique est régulière.

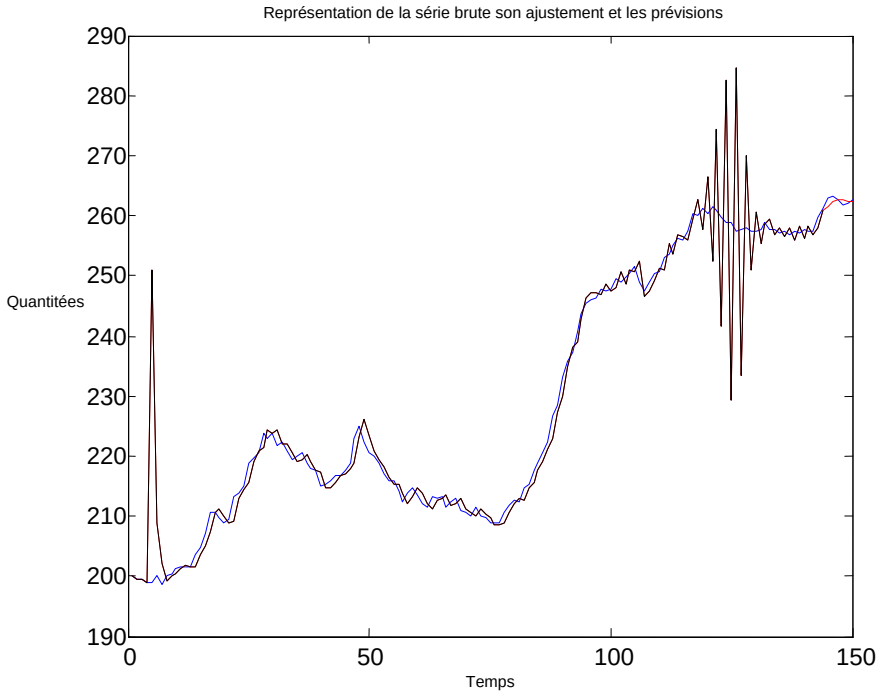
SCRP	SCR	ITERATION
8.6713	23378	1
2.0832	10468	2
0.65755	**3336.4**	**3**
1.1865	2049.6	4
1.632	1746.5	5
1.9006	1638.1	6
2.0667	1586.4	7
2.1779	1557.3	8
2.2579	1539.1	9
2.3187	1526.9	10
2.3669	1518.3	11

XXX.1.4 Prévision :

BRUT	PREVI	ERRE
262.9	262,9385	-0,0385
263.3	263,6108	-0,3108
262.8	263,2905	-0,4905
261.8	262,9426	-1,1426
262.2	262,8307	-0,6307
262.7	262,8571	-0,1571

D'après le tableau ci-dessus, on remarque que les écarts entres les prévisions obtenues par le réseau de neurones et les valeurs de série Brut sont très faibles, les valeurs sont presque confondues, d'ailleurs on peut le constater graphiquement (graphe ci-dessous).

On constat aussi d'après le graphe ci-dessous que l'écart entre la série d'ajustement et la série brut est très faible et les deux graphes sont a peine perceptible sauf au début et a la fin de la série d'ajustement.



Graphe de la série avec son ajustement et la prévision.

XXX.1.5 Etude de la complexité de cet algorithme :

XXX.1.5.1 Calcul de la complexité de la fonction prévision :

Cette fonction est constituée d'une boucle « For » a son intérieur une autre boucle « For » imbriquée, et des opérations indépendantes des boucles.

XXX.1.5.2 Complexité de la boucle For :

Cette dernière contient à l'intérieur une boucle imbriquée, et d'autres opérations indépendantes de la boucle imbriquée, et la formule pour calculer sa complexité est :

$$T_{Al}(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)+84\right)$$ Ou :

T_1 : Est la complexité de la boucle « For » imbriquée.

T_c : Nombre d'affectation d'indice.

84 : Nombre d'opérations dans la boucle « For » indépendamment de la boucle imbriquée.

Donc : la complexité de la boucle imbriquée est :

$$T_1(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)\right)=\sum_{i=1}^{n}(1+32)=\sum_{i=1}^{n}33=33\text{n.}$$

La complexité de la boucle « For » est :

$$T_{Al}(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)+84\right)=T_{Al}(n)=\sum_{i=1}^{n}(1+33n+84)=\sum_{i=1}^{n}1+33n\sum_{i=1}^{n}1+84\sum_{i=1}^{n}1=$$

$$33n^2+85n$$

XXX.1.5.3 La complexité totale de la fonction prévision :

On a : 35 est le nombre d'opérations en dehors de la boucle « For » donc la complexité total de la fonction prévision est :

Op_p (n)$=35+T_{Al}(n)=35+85n+33n^2$

XXX.1.5.4 Calcul de la complexité de la fonction Treillis :

Cette fonction est constituée de plusieurs opérations indépendamment des boucles, une boucle « For » et d'une grande boucle « For » qui contiennent 4 autres boucles « For » imbriquées comme suite :

La boucle n°4 est imbriquée dans la boucle n°3 qui est imbriquée dans la boucle n°2 cette dernière est imbriquée dans la boucle n°1 et les quatre boucles sont imbriquées dans la grande boucle « For ».

Soit :

Op_1 (n) le nombre des opérations dans la grande boucle « For ».

T_1 (i, n) le nombre des opérations dans la boucle « For » n°1.

T_2 (i, n) le nombre des opérations dans la boucle « For » n°2.

T_3 (i, n) le nombre des opérations dans la boucle « For » n°3.

T_4 (i, n) le nombre des opérations dans la boucle « For » n°4.

XXX.1.5.5 Calcul du nombre d'opérations dans la grande boucle « For » :

$T_4(i,n) = \sum_{i=1}^{n} 6 = 6n$

$T_3(i,n) = \sum_{i=1}^{n}((\sum_{i=1}^{n} 6) + 1 + 1) = \sum_{i=1}^{n}(6n+2) = n(6n+2) = 2n + 6n^2$

$T_2(i,n) = \sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n} 6) + 1 + 1)) + 1) = \sum_{i=1}^{n}(2n + 6n^2 + 1) = n + 2n^2 + 6n^3$

$T_1(i,n) = \sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n} 6) + 1 + 1)) + 1)) + 1) = \sum_{i=1}^{n}(n + 2n^2 + 6n^3 + 1) = n + n^2 + 2n^3 + 6n^4$

Donc le nombre d'opérations dans la grande boucle « For » est :

$Op_1(n) = \sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n} 6) + 1 + 1)) + 1)) + 1)) + 1) = \sum_{i=1}^{n}(1 + n + n^2 + 2n^3 + 6n^4)$

$Op_1(n) = n + n^2 + n^3 + 2n^4 + 6n^5$

XXX.1.5.6 Calcul d'opérations en dehors de la grande boucle « For » :

Soit Op_2 (n) le nombre des opérations dans la fonction treillis en dehors de la grande boucle « For ».

$T(n) = \sum_{i=1}^{n}(T_C(i,n) + T(i,n))$ est le nombre d'opérations dans la boucle « For » qui est indépendante de la grande boucle « For ».

6 : est le nombre d'opérations indépendamment de toutes les boucle qui existent dans la fonction treillis.

Donc :

$Op_2(n) = 6 + T(n) = 6 + \sum_{i=1}^{n} 2 = 6 + 2n$

Donc le nombre d'opérations total dans la fonction treillis est :

$Op_t(n)$ $= Op_1(n) + Op_2(n) = n + n^2 + n^3 + 2n^4 + 6n^5 + 6 + 2n = 6 + 3n + n^2 + n^3 + 2n^4 + 6n^5$

XXX.1.5.7 Calcul de la complexité de la fonction RBF :

XXX.1.5.7.1 Calcul du nombre d'opérations dans la boucle While :

Soit T_A (n) le nombre d'opérations dans la boucle While tel que :

$$T_A(n) = \left(\sum_{i=1}^{n} \left(T_C(i,n) + T_{A1}(i,n) + T_{A2}(i,n) + T_{A3}(i,n) + 300 + op_p(n) + 1 \right) \right) + T_C(n+1,n)$$ Tel

que :

T_c : Nombre de test de la condition C.

T_{A1} : Nombre d'opérations dans la grande boucle For, cette dernière contient a l'intérieur deux boucles « For » imbriquées (T_1 et T_2).

T_{A2} : Nombre d'opérations dans la condition « if ».

T_{A3} : Nombre d'opérations dans la condition « if ».

296 : Nombre d'opérations dans la boucle While indépendamment des boucles imbriquées.

Op_p (n) : Nombre d'opérations dans la fonction prévision.

1 : représente l'appel de la fonction prévision car l'appel est compté comme une opération.

On a :

$T_C(i,n) = 1$.

XXX.1.5.7.2 Calcul du nombre d'opération dans la grande boucle « For » :

Dans cet algorithme la grande boucle « For » est constituée d'une action conditionnelle de la forme : if (condition) action1 else action2 end if.

Donc d'après les formules illustrées ci-dessus le nombre d'opérations dans la grande boucle « For » est :

$T_{A1}(n) = \sum_{i=1}^{n} (T_C(n) + Ts) = \sum_{i=1}^{n} (T_C(n) + (1 + \text{Max}(Ta_1(n), Ta_2(n))))$ tel que :

Ta_1 : Est le nombre d'opérations dans l'action1 (a1).

Ta_2 : Est le nombre d'opérations dans l'action2 (a2).

T_c : Nombre d'affectation d'indice de la grande boucle « For ».

Ts : Est le nombre d'opérations dans la structure if (condition) action1 else action2 end if.

Soit :

$Ts = 1 + \text{Max}(Ta_1(n), Ta_2(n))$

$Ta_1(n) = T_1(n) + T_2(n) + T_3(n)$ tel que :

T_1 : est le nombre d'opération dans la première boucle de l'action 1.

T_2 : est le nombre d'opération dans la deuxième boucle de l'action 1.

T_3 : est le nombre d'opération en dehors des deux boucles «For » de l'action 1.

Donc :

$Ta_1(n) = (\sum_{i=1}^{n}(T_{C1}(i,n)+T_1(i,n)))+(\sum_{i=1}^{n}(T_{C2}(i,n)+T_2(i,n)))+T_3(i,n)$ Tel que :

$Ta_1=69+\sum_{i=1}^{n}20+\sum_{i=1}^{n}85=69+20n+85n=69+105n.$

De même pour la deuxième action :

$Ta_2(n) = (\sum_{i=1}^{n}(T_{C1}(i,n)+T_1(i,n)))+(\sum_{i=1}^{n}(T_{C2}(i,n)+T_2(i,n)))+T_3(i,n)$

$Ta_2=69+\sum_{i=1}^{n}20+\sum_{i=1}^{n}85=69+20n+85n=69+105n.$

Donc Ts=1+Max ((69+105n), (69+105n))=1+69+105n =70+105n

Ts=70+105n

Le nombre d'opérations dans la grande boucle « For » est :

$T_{A1}(n)=\sum_{i=1}^{n}(T_C(n)+(1+\text{Max}(Ta_1(n), Ta_2(n))))=\sum_{i=1}^{n}(1+70+105n)$

$T_{A1}(n)=\sum_{i=1}^{n}71+\sum_{i=1}^{n}105n=71n+105n^2$

En dehors de la grande boucle « For » on trouve deux conditions « if » :

XXX.1.5.7.3 Le nombre d'opérations dans la 1ere condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

$T_{A2}(n)= T_C(n) + \text{Max}(T_{a1}(n), T_{a2}(n))$.

Dans le cas de notre algorithme l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'affectations, donc notre équation devienne :

$T_{A2}(n)= T_C(n) + T_{a1}(n)$.

$T_{A2}(n)= 1+7=8$

XXX.1.5.7.4 Le nombre d'opérations dans la 2eme condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

$T_{A3}(n)= T_C(n) + \text{Max}(T_{a1}(n), T_{a2}(n))$.

De même pour la deuxième boucle « if » l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'instructions, donc notre équation devienne :

$T_{A3}(n)= T_C(n) + T_{a1}(n)$.

T_{A3} (n)= 1 +12=13.

Donc le nombre total d'opérations dans la boucle While est :

$$T_A(n) = \left(\sum_{i=1}^{n} \left(T_C(i,n) + T_{A1}(i,n) + T_{A2}(i,n) + T_{A3}(i,n) + 296 + op_p(n) + 1 \right) \right) + T_C(n+1,n) =$$

$$\left(\sum_{i=1}^{n} (1 + 71n + 105n^2 + 8 + 13 + 296 + 35 + 85n + 33n^2 + 1)) \right) + 1 = \left(\sum_{i=1}^{n} (354 + 156n + 138n^2)) \right) + 1$$

$$T_A(n) = 1 + 354n + 156n^2 + 138n^3$$

Le nombre total d'opération **Op(n)** de la fonction principale RBF qui est aussi le nombre total d'opérations effectuées dans l'exécution de l'algorithme est:

Op(n)= (Op_t (n)+1)+T_A(n)+186 tel que :

Op_t (n) : est le nombre d'opérations dans la fonction Treillis

T_A (n) : est le nombre d'opérations dans la boucle While de la fonction RBF.

186 : est le nombre d'opérations dans la fonction RBF indépendamment des boucle et des autres fonctions.

1 : est l'opération d'appel de la fonction Treillis dans la fonction RBF car la fonction treillis est appelé qu'une seule fois dans l'exécution de l'algorithme.

Donc :

Op(n)=$1 + 6 + 3n + n^2 + n^3 + 2n^4 + 6n^5 + 1 + 354n + 156n^2 + 138n^3 + 186$

Op(n)=$194 + 357n + 157n^2 + 139n^3 + 2n^4 + 6n^5$

On a :

$$\lim_{n \to \infty} (194 + 357n + 157n^2 + 139n^3 + 2n^4 + 6n^5)/n^5 = 6 \neq 0 \Rightarrow Op(n) \quad est\ en \quad \Theta(n^5)$$

XXX.1.6 Conclusion:

D'après les résultats ci-dessus qui ont été obtenus, l'algorithme qui a été utilisé dans cette application est d'une complexité polynomiale d'ordre 5, c'est-à-dire que notre algorithme est en $O(n^5)$.

XXX.2 2eme Cas d'application :

Dans ce deuxième cas de l'application et de la même manière l'apprentissage se fera sur les trois paramètres W, C et σ comme suite :

L'ajustement des Centres et des rayons (Ci, σ_i) s'effectue par la méthode de descente de gradient qui a été illustrée ci-dessus.

L'ajustement des poids W_i se fera par **la méthode de descente de gradient** définie dans la première méthode ci-dessus.

XXX.2.1 Le choix d'architecture de réseau à fonction de base radiale (Rbf) :

Le choix de l'architecture du réseau se fait de la même façon que la première méthode de l'application ci-dessus, c'est-à-dire qu'on choisi l'architecture qui donne une erreur minimale au sens de SCR_p (la somme des carrés des résidus de la prévision) car notre but est de calculé une prévision.

Donc d'après les expériences qui ont été menées sur plusieurs types d'architectures (voir un exemple dans le tableau ci-dessous) :

	n^bre de neurones dans la couche cachée	SCR	SCR $_P$
Rbf à 2 entrées	**2**	**7744.9**	**0.18714**
	3	**408.81**	**1.0435**
	4	**177.44**	**3.6486**
Rbf à 3 entrées	**2**	**185.82**	**5.5372**
	3	**144.18**	**8.9136**
	4	**153.89**	**6.3884**
Rbf à 4 entrées	**2**	**143.33**	**4.8713**
	3	**140.77**	**8.8206**
	4	**1099.8**	**2.5749**

Le réseau Rbf à deux (02) entrées et a deux (02) neurones cachés donne le meilleur résultat (SCR_p = **0.18714**).

XXX.2.2 Réseau à fonction de base radiale à 2 entrées

Taille de la Base d'apprentissage : 144

Taille de la Base de test : 6

Nombres d'itérations : 5

Nombres de neurones dans la couche cachée : 2

Les figures (1, 2,3) illustrent l'apprentissage qui consiste à chercher la configuration de poids qui minimise l'erreur ; ces figures représentent les graphes des fonctions suivantes :

- la série et son ajustement en utilisant les poids de la dernière itération.

- la somme des carrés des résidus de l'ajustement (SCR).
- la somme des carrés des résidus de la prévision (SCRp).

Deuxième itération :

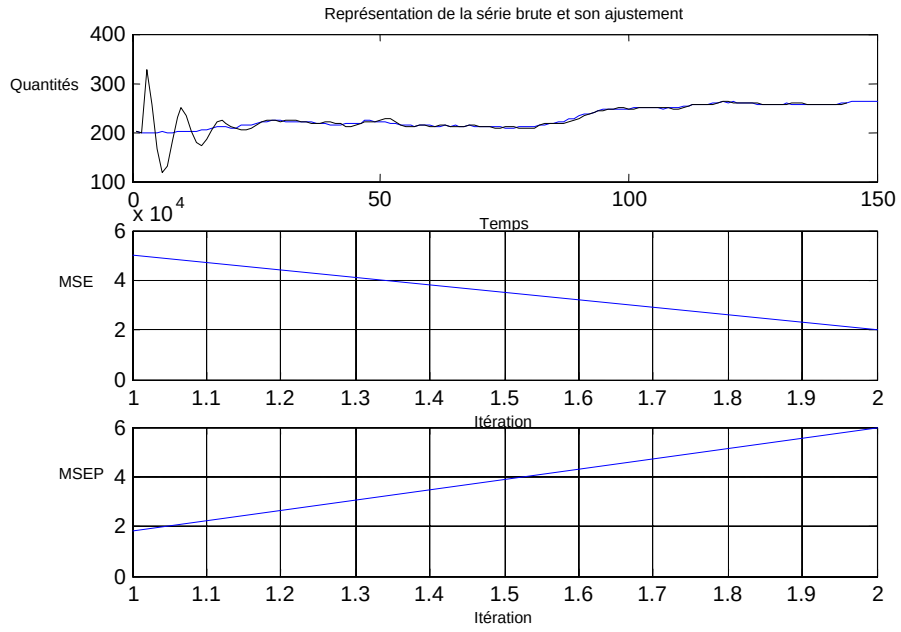


Figure 1

$$SCR = \frac{1}{2}\sum_{t=1}^{144} e_t^2 = 19939$$

$$SCR_p = \frac{1}{2}\sum_{t=145}^{150} e_t^2 = 5.9934$$

Itération intermédiaire :

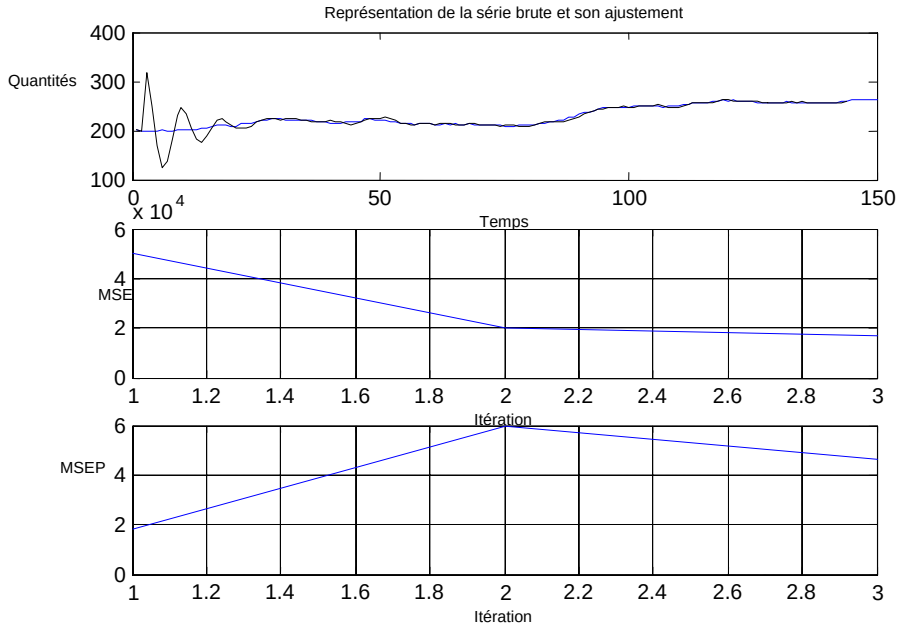


Figure 2

SCR= 16797

SCRp= 4.6569

Dernière itération :

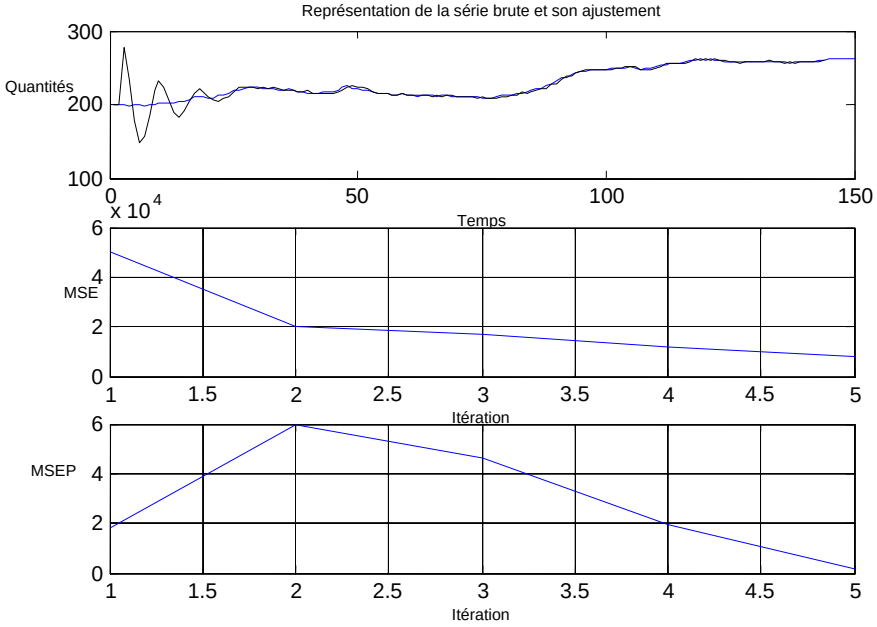


Figure 3

SCR= 7744.9

SCRp= **0.18714**

XXX.2.3 Test d'arrêt :

Théoriquement : Sur la base d'apprentissage, l'erreur diminue constamment, alors que sur la base de test elle passe par un minimum. Si l'apprentissage se prolonge au delà, les performances en test diminuent.

D'après le tableau ci-dessous on remarque que :

L'erreur quadratique de l'ajustement (SCR) diminue sans cesse au cours de l'apprentissage, par contre l'erreur quadratique de la prévision (SCR_P) passe par un minimum a **l'itération 5**, ce qui est conforme à ce qui a été écrit dans la partie théorique. Chose qu'on a pas pu voir graphiquement, ce qui peut être expliqué par :

- taille de l'échantillon (faible).
- la série chronologique est régulière.

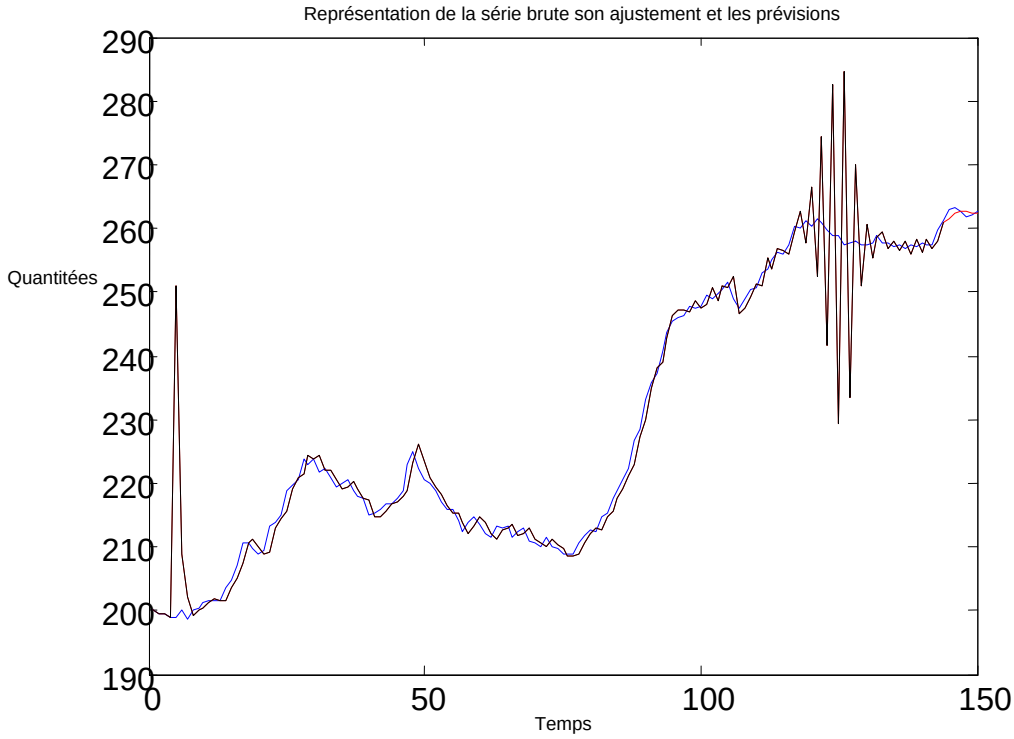
SCRP	SCR	ITERATION
1.835	49837	1
5.9934	19939	2
4.6569	16797	3
1.9786	11745	4
0.18714	**7744.9**	**5**
2.8849	4970.9	6
10.739	3255.6	7
21.491	2251.5	8
32.51	1671.2	9
42.21	1332.5	10
50.035	1131.7	11
56.026	1011.5	12
60.467	939.78	13
63.696	898.28	14

XXX.2.4 Prévision :

BRUT	PREVI	ERRE
262.9	263.1071	-0,2071
263.3	263.2538	0,0462
262.8	262.5682	0,2318
261.8	262.2077	-0,4077
262.2	262.2455	-0,0455
262.7	262.3725	0,3275

D'après le tableau ci-dessus, on remarque que les écarts entres les prévisions obtenues par le réseau de neurones et les valeurs de série Brut sont très faibles, les valeurs sont presque confondues, d'ailleurs on peut le constater graphiquement (graphe ci-dessous).

On constat aussi d'après le graphe ci-dessous que l'écart entre la série d'ajustement et la série brut est très faible et les deux graphes sont a peine perceptible sauf au début et a la fin de la série d'ajustement.



Graphe de la série avec son ajustement et la prévision.

XXX.2.5 Etude de la complexité de cet algorithme :

De la même manière que ci-dessus on détermine la complexité de cette algorithme.

XXX.2.5.1 Calcul de la complexité de la fonction prévision :

Cette fonction est constituée d'une boucle « For » à son intérieur une autre boucle « For » imbriquée, et des opérations indépendantes des boucles.

XXX.2.5.2 Complexité de la boucle For :

Cette dernière contient à l'intérieur une boucle imbriquée, et d'autres opérations indépendantes de la boucle imbriquée, et la formule pour calculer sa complexité est :

$$T_{Al}(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)+21\right)$$ Ou :

T_1 : Est la complexité de la boucle « For » imbriquée.

T_c : Nombre d'affectation d'indice.

21 : Nombre d'opérations dans la boucle « For » indépendamment de la boucle imbriquée.

Donc : la complexité de la boucle imbriquée est :

$$T_1(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)\right)=\sum_{i=1}^{n}(1+15)=\sum_{i=1}^{n}16=16\text{n.}$$

La complexité de la boucle « For » est :

$$T_{Al}(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)+21\right)=T_{Al}(n)=\sum_{i=1}^{n}(1+16n+21)=\sum_{i=1}^{n}1+16n\sum_{i=1}^{n}1+21\sum_{i=1}^{n}1=$$

$16n^2+22n$

XXX.2.5.3 La complexité totale de la fonction prévision :

On a : 30 est le nombre d'opérations en dehors de la boucle « For » donc la complexité total de la fonction prévision est :

Op_p (n)$=30+T_{Al}(n)=30+22n+12n^2$

XXX.2.5.4 Calcul de la complexité de la fonction Treillis :

Cette fonction est constituée de plusieurs opérations indépendamment des boucles, une boucle « For » et d'une grande boucle « For » qui contiennent 2 autres boucles « For » imbriquées comme suite :

La boucle n°2 est imbriquée dans la boucle n°1, et les deux boucles sont imbriquées dans la grande boucle « For ».

Soit :

Op_1 (n) le nombre des opérations dans la grande boucle « For ».

T_1 (i, n) le nombre des opérations dans la boucle « For » n°1.

T_2 (i, n) le nombre des opérations dans la boucle « For » n°2.

XXX.2.5.4.1 Calcul du nombre d'opérations dans la grande boucle « For » :

T_2 (i, n)= $\sum_{i=1}^{n} 4 = 4n$

T_1 (i, n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n} 4)+1+1) = \sum_{i=1}^{n}(4n+2) = n(4n+2) = 2n+4n^2$

Donc le nombre d'opérations dans la grande boucle « For » est :

Op_1 (n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n} 4)+1+1))+1) = \sum_{i=1}^{n}(2n+4n^2+1) = n+2n^2+4n^3$

Op_1 (n)= $n+2n^2+4n^3$

XXX.2.5.4.2 Calcul d'opérations en dehors de la grande boucle « For » :

Soit Op_2 (n) le nombre des opérations dans la fonction treillis en dehors de la grande boucle « For ».

T (n)= $\sum_{i=1}^{n}(T_C(i,n)+T(i,n))$ est le nombre d'opérations dans la boucle « For » qui est indépendante de la grande boucle « For ».

6 : est le nombre d'opérations indépendamment de toutes les boucle qui existent dans la fonction Treillis.

Op_2 (n)= 6+T (n)=6+ $\sum_{i=1}^{n} 3 = 6+3n$

Donc le nombre d'opérations total dans la fonction Treillis est :

Op_t (n)= Op_1 (n) + Op_2 (n)= $n+2n^2+4n^3+6+3n = 6+4n+2n^2+4n^3$

XXX.2.5.5 Calcul de la complexité de la fonction RBF :

XXX.2.5.5.1 Calcul du nombre d'opérations dans la boucle While :

Soit T_A (n) le nombre d'opérations dans la boucle While tel que :

$T_A(n) = (\sum_{i=1}^{n}(T_C(i,n)+T_{A1}(i,n)+T_{A2}(i,n)+T_{A3}(i,n)+300+op_p(n)+1)) + T_C(n+1,n)$ Tel que :

T_c : Nombre de test de la condition C.

T_{A1} : Nombre d'opérations dans la grande boucle For, cette dernière contient a l'intérieur deux boucles « For » imbriquées (T_1 et T_2).

T_{A2} : Nombre d'opérations dans la condition « if ».

T_{A3} : Nombre d'opérations dans la condition « if ».

300 : Nombre d'opérations dans la boucle While indépendamment des boucles imbriquées.

Op_p (n) : Nombre d'opérations dans la fonction prévision.

1 : représente l'appel de la fonction prévision car l'appel est compté comme une opération.

On a :

$T_C(i,n) = 1.$

XXX.2.5.5.2 Calcul du nombre d'opération dans la grande boucle « For » :

Dans cet algorithme la grande boucle « For » est constituée d'une action conditionnelle de la forme : if (condition) action1 else action2 end if.

Donc d'après les formules illustrées ci-dessus le nombre d'opérations dans la grande boucle « For » est :

$T_{A1}(n) = \sum_{i=1}^{n} (T_C(n) + Ts) = \sum_{i=1}^{n} (T_C(n) + (1 + \text{Max}(Ta_1(n), Ta_2(n))))$ tel que :

Ta_1 : Est le nombre d'opérations dans l'action1 (a1).

Ta_2 : Est le nombre d'opérations dans l'action2 (a2).

T_c : Nombre d'affectation d'indice de la grande boucle « For ».

Ts : Est le nombre d'opérations dans la structure if (condition) action1 else action2 end if.

Soit :

$Ts = 1 + \text{Max}(Ta_1(n), Ta_2(n))$

$Ta_1(n) = T_1(n) + T_2(n) + T_3(n)$ tel que :

T_1 : est le nombre d'opération dans la première boucle de l'action 1.

T_2 : est le nombre d'opération dans la deuxième boucle de l'action 1.

T_3 : est le nombre d'opération en dehors des deux boucles «For » de l'action 1.

Donc :

$Ta_1(n) = \left(\sum_{i=1}^{n} (T_{C1}(i,n) + T_1(i,n)) \right) + \left(\sum_{i=1}^{n} (T_{C2}(i,n) + T_2(i,n)) \right) + T_3(i,n)$ Tel que :

$Ta_1 = 21 + \sum_{i=1}^{n} 12 + \sum_{i=1}^{n} 45 = 21 + 12n + 45n = 21 + 57n.$

De même pour la deuxième action :

$$Ta_2(n) = \left(\sum_{i=1}^{n} (T_{C1}(i,n) + T_1(i,n)) \right) + \left(\sum_{i=1}^{n} (T_{C2}(i,n) + T_2(i,n)) \right) + T_3(i,n)$$

Ta_2=21+ $\sum_{i=1}^{n} 12 + \sum_{i=1}^{n} 48$ =21+12n+48n=21+60n.

Donc Ts=1+Max ((21+57n), (21+60n))=1+21+60n =22+60n

Ts=22+60n

Le nombre d'opérations dans la grande boucle « For » est :

T_{A1} (n)= $\sum_{i=1}^{n}$ (T_C (n) + (1+Max (Ta_1 (n), Ta_2 (n))))= $\sum_{i=1}^{n}$ (1+ 22+60n)

T_{A1} (n)= $\sum_{i=1}^{n} 23 + \sum_{i=1}^{n} 60n = 23n + 60n^2$

En dehors de la grande boucle « For » on trouve deux conditions « if » :

XXX.2.5.5.3 Le nombre d'opérations dans la 1ere condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

T_{A2} (n)= T_C (n) + Max (T_{a1} (n), T_{a2} (n)).

Dans le cas de notre algorithme l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'affectations, donc notre équation devienne :

T_{A2} (n)= T_C (n) + T_{a1} (n).

T_{A2} (n)= 1+7=8

XXX.2.5.5.4 Le nombre d'opérations dans la 2eme condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

T_{A3} (n)= T_C (n) + Max (T_{a1} (n), T_{a2} (n)).

De même pour la deuxième boucle « if » l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'instructions, donc notre équation devienne :

T_{A3} (n)= T_C (n) + T_{a1} (n).

T_{A3} (n)= 1 +12=13.

Donc le nombre total d'opérations dans la boucle While est :

$$T_A(n) = \left(\sum_{i=1}^{n} (T_C(i,n) + T_{A1}(i,n) + T_{A2}(i,n) + T_{A3}(i,n) + 300 + op_p(n) + 1) \right) + T_C(n+1,n) =$$

$$\left(\sum_{i=1}^{n} (1 + 23n + 60n^2 + 8 + 13 + 300 + 30 + 22n + 12n^2 + 1)) \right) + 1 = \left(\sum_{i=1}^{n} (353 + 45n + 72n^2)) \right) + 1$$

$$T_A(n) = 1 + 353n + 45n^2 + 72n^3$$

Le nombre total d'opération **Op(n)** de la fonction principale RBF qui est aussi le nombre total d'opérations effectuées dans l'exécution de l'algorithme est:

$Op(n)= (Op_t (n)+1)+T_A(n)+174$ tel que :

$Op_t (n)$: est le nombre d'opérations dans la fonction Treillis

$T_A (n)$: est le nombre d'opérations dans la boucle While de la fonction RBF.

174 : est le nombre d'opérations dans la fonction RBF indépendamment des boucle et des autres fonctions.

1 : est l'opération d'appel de la fonction Treillis dans la fonction RBF car la fonction treillis est appelé qu'une seule fois dans l'exécution de l'algorithme.

Donc :

Op(n)=$1+ 6+2n+2n^2+4n^3+1+353n+45n^2+72n^3+174$

Op(n)=$182+355n+47n^2+76n^3$

On a :

$\lim_{n\to\infty}(182+355n+47n^2+76n^3)/n^3=76\neq 0\Rightarrow Op(n) \quad est\ en \quad \Theta(n^3)$

XXX.2.5.6 Conclusion:

D'après les résultats ci-dessus qui ont été obtenus, l'algorithme qui a été utilisé dans cette application est d'une complexité polynomiale d'ordre 3, c'est-à-dire que notre algorithme est en $O(n^3)$.

XXXI Comparaison des résultats :

La comparaison des 4 techniques utilisées ci-dessus revient a comparé la somme des carrés des résidus de la prévision (SCRp) car ces techniques ont été employées pour calculer la prévision, et Le tableau ci-dessous résume les résultats trouvés :

	L'ETAT DES PARAMETRES DE LA FONCTION NOYAU	LES REGLES UTILISEES POUR LA MODIFICATION DES POIDS	complexité	SCR_P
1 ère méthode	Les paramètres (Ci, σ_i) sont fixe une fois pour toute durant tout le processus d'apprentissage par la méthode des treillis.	La règle de Widdrow-Hoff	$O(n^5)$	**0.74106**
		La règle de rétro propagation du gradient	$O(n^3)$	**0.19667**
2 eme méthode	Les paramètres (Ci, σ_i) sont variable durant tout le processus d'apprentissage par la méthode descente de gradient.	La règle de Widdrow-Hoff	$O(n^5)$	**0.65755**
		La règle de rétro propagation du gradient	$O(n^3)$	**0.18714**

D'après les résultats affichés dans le tableau ci-dessus on constat que les résultats qui ont étés obtenus avec la 2eme méthode c'est à dire : les paramètres (Ci, σ_i) qui caractérises la fonction noyau sont variables durant tout le processus d'apprentissage par la méthode descente de gradient sont meilleur que les résultats qui ont été obtenus avec la 1ere méthode c'est à dire : les paramètres (Ci, σ_i) sont fixe une fois pour toute durant tout le processus d'apprentissage par la méthode des treillis. .

On remarque aussi que les résultats qui ont étés obtenus la ou la règle de modification des poids est la règle de rétro propagation du gradient sont meilleur que les résultats obtenus la ou la règle de modification des poids est la règle de Widdrow-Hoff.

De même pour la complexité des algorithmes, on remarque que l'algorithme qui contient la règle de rétro propagation du gradient pour la modification des poids est meilleurs, c'est-à-dire polynomiale de degré inférieur que l'algorithme qui contient la règle de Widdrow-Hoff pour la modification des poids.

Deuxième partie de l'application selon La nouvelle approche sur principe du mode d'apprentissage supervisé

XXXII Nouvelle approche :

Dans cette 2eme partie de l'application une modification a été apportée sur le principe du mode d'apprentissage supervisé.

Dans un premier lieu, le principe de l'apprentissage supervisé est basé sur l'erreur déterminer en comparant la sortie calculé par le réseau a la sortie désiré, et l'erreur due a cette comparaison sera utilisé pour calculé la modification des poids de l'itération suivante (exemple : par la méthode rétro propagation du gradient ou la règle de Widdrow-Hoff) qui vise a réduire cette erreur au file d'itérations, c'est à dire utilisé l'erreur issue de l'itération **(n-1)** pour pouvoir calculé la modification des poids qui sera utiliser dans l'itération **(n)**.

La nouvelle approche est comme suite :

C'est au lieu d'utiliser l'erreur commise à l'itération **(n-1)** pour calculer la modification des poids qui sera utilisé dans l'itération **(n),** on utilisera l'erreur issue de l'itération **(n-2)** pour calculer la mise à jour des poids pour l'itération **(n)**.

Premiere Méthode

Dans cette première méthode le centre et le rayon de la fonction noyau sont fixés une fois pour toute dans le processus d'apprentissage de la même façon que la première partie de l'application.

XXXIII L'apprentissage des poids de connexions :

XXXIII.1 1ere Cas : La règle de Widrow-Hoff

Les poids sont mis à jour par la règle de Widrow-Hoff selon la nouvelle idée :

Donc la règle de Widrow-Hoff est équivalente à :

$$w_{ji}(t+1) = w_{ji}(t) - \alpha(S_{pi}(t-1) - y_{pi}(t-1))\ \phi_{pj}(t)$$

Où : $w_{ji}(t+1)$: Le poids reliant le neurone j à i à l'instant t+1.

$w_{ji}(t)$: Le poids reliant le neurone j à i à l'instant t.

α : Un paramètre d'adaptation de l'apprentissage (α >0).

$y_{pi}(t-1)$: La sortie désirée à l'instant t-1.

$S_{pi}(t-1)$: La sortie du réseau à l'instant t-1.

p : représente l'exemple courant.

XXXIII.1.1 Le choix d'architecture de réseau à fonction de base radiale (Rbf) :

De la même manière que la première partie de l'application on a choisi l'architecture qui donne une erreur minimale au sens de SCR_p (la somme des carrés des résidus de la prévision).

Donc d'après les expériences qui ont été menées sur plusieurs types d'architectures, le réseau Rbf à trois entrées et a deux neurones cachés donne le meilleur résultat (SCR_p = **0.70635**).

XXXIII.1.2 Réseau à fonction de base radiale à 3 entrées

Taille de la Base d'apprentissage : 144

Taille de la Base de test : 6

Nombres d'itérations : 1426

Nombres de neurones dans la couche cachée : 2

Les figures (1, 2,3) illustrent l'apprentissage qui consiste à chercher la configuration de poids qui minimise l'erreur ; ces figures représentent les graphes des fonctions suivantes :

- la série et son ajustement en utilisant les poids de la dernière itération.
- la somme des carrés des résidus de l'ajustement (SCR).

- la somme des carrés des résidus de la prévision (SCRp).

Deuxième itération :

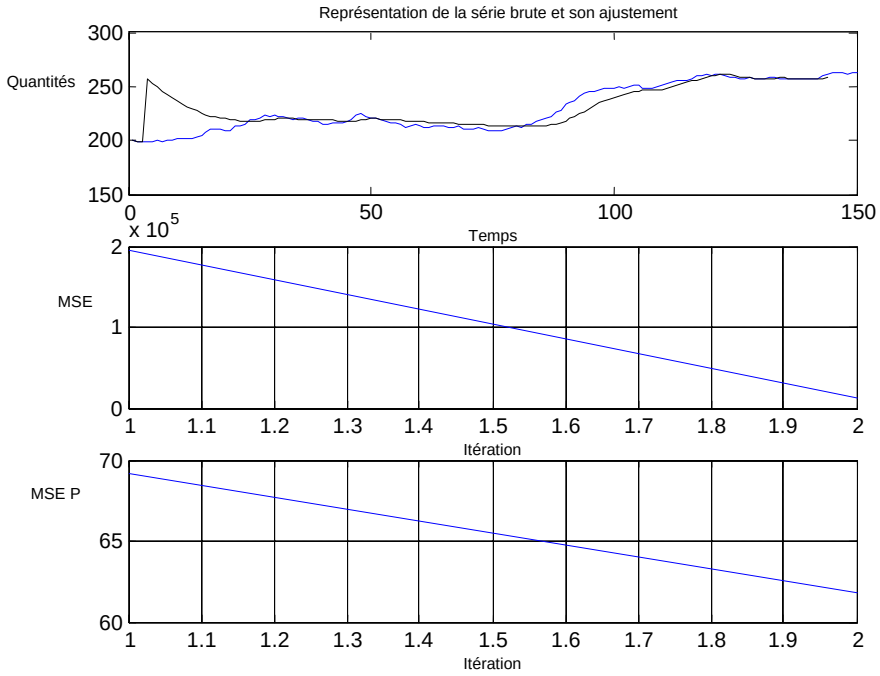


Figure 1

$$SCR = \frac{1}{2}\sum_{t=1}^{144} e_t^2 = 11866$$

$$SCR_p = \frac{1}{2}\sum_{t=145}^{150} e_t^2 = 61.819$$

Itération intermédiaire :

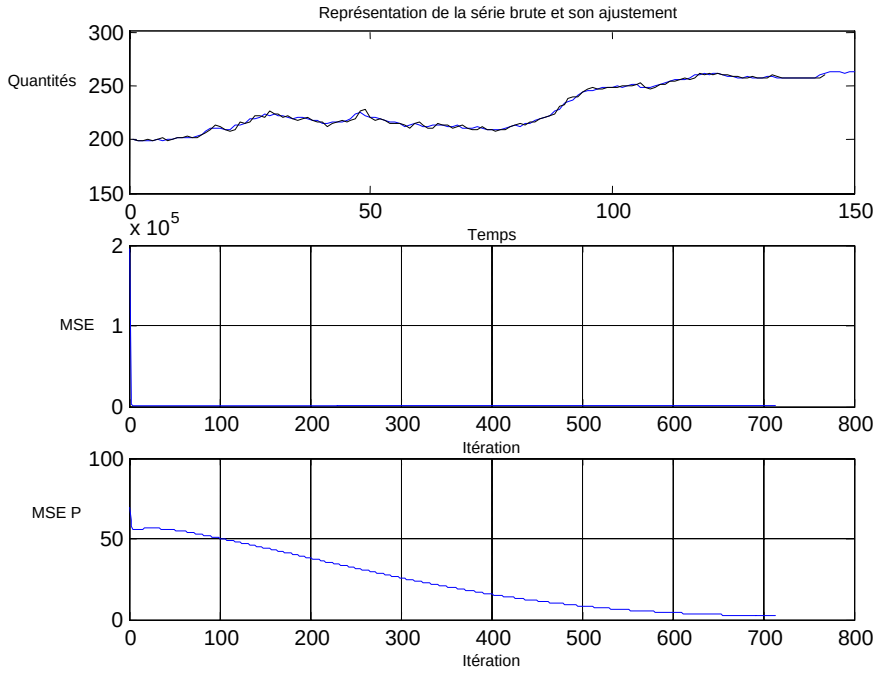


Figure 2

SCR= 176.65

SCRp= 2.1564

Dernière itération :

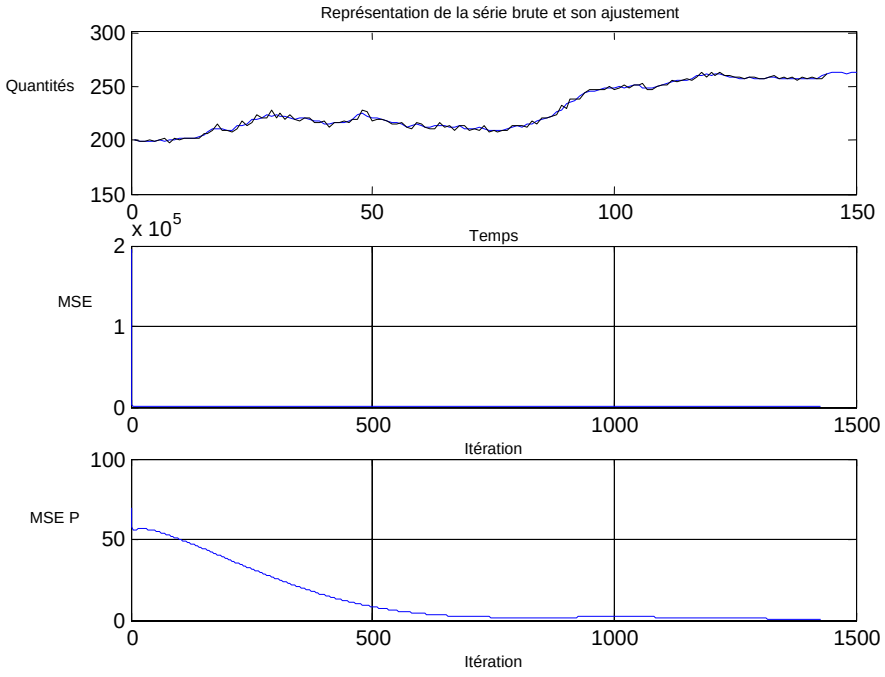


Figure 3

SCR= 254.32

SCRp= **0.70635**

XXXIII.1.3 Test d'arrêt :

Théoriquement : Sur la base d'apprentissage, l'erreur diminue constamment, alors que sur la base de test elle passe par un minimum. Si l'apprentissage se prolonge au delà, les performances en test diminuent.

Donc, d'après les résultats issues de l'application de la nouvelle technique du principe du mode d'apprentissage supervisé, qui sont affichés dans le tableau ci-dessous on remarque que :

L'erreur quadratique de l'ajustement (SCR) augmente sans cesse au cours de l'apprentissage contrairement a ce qui a été dit dans la partie théorique concernant le mode d'apprentissage supervisé, par contre l'erreur quadratique de la prévision (SCR_P) passe par un minimum a **l'itération 1426**, ce qui est conforme à ce qui a été écrit dans la partie théorique, Chose qu'on a pas pu le voir graphiquement, ce qui peut être expliqué par :

- taille de l'échantillon (faible).
- la série chronologique est régulière.

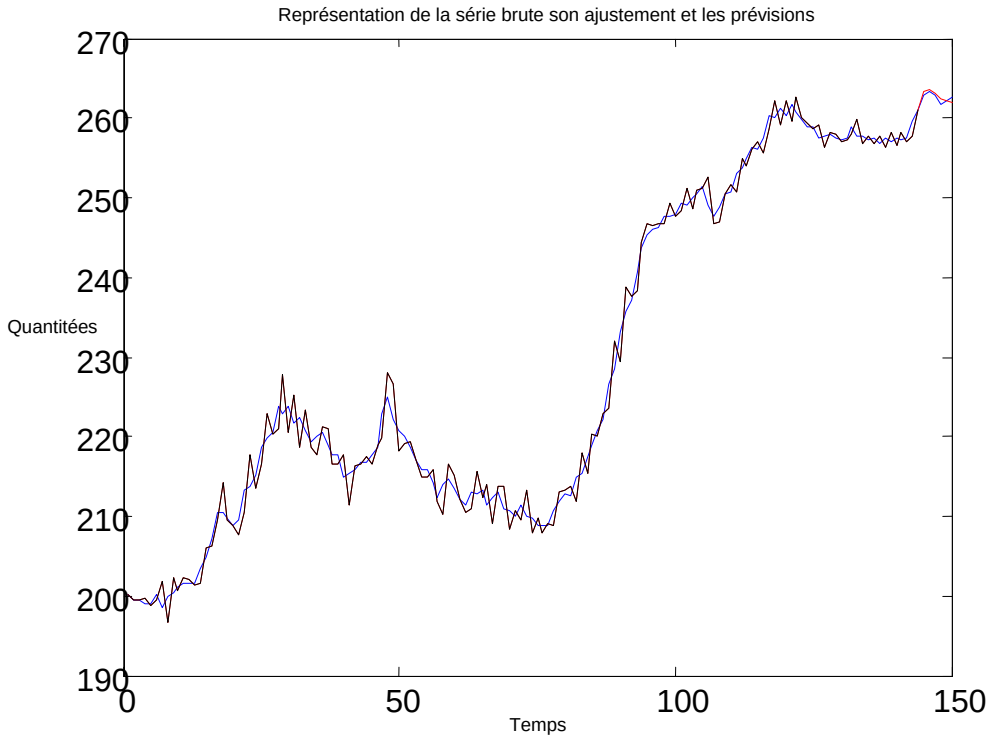
SCRP	SCR	ITERATION
0.70701	253.37	1421
0.70677	253.56	1422
0.70658	253.75	1423
0.70644	253.94	1424
0.70637	254.13	1425
0.70635	**254.32**	**1426**
0.70639	254.52	1427
0.70649	254.71	1428
0.70665	254.9	1429
0.70688	255.1	1430
0.70716	255.29	1431

XXXIII.1.4 Prévision :

BRUT	PREVI	ERRE
262.9	263,2362	-0,3362
263.3	263,6551	-0,3551
262.8	263,1061	-0,3061
261.8	262,4661	-0,6661
262.2	262,0753	0,1247
262.7	261,9122	0,7878

D'après le tableau ci-dessus, on remarque que les écarts entres les prévisions obtenues par le réseau de neurones et les valeurs de série Brut sont faibles, d'ailleurs on peut le constater graphiquement (graphe ci-dessous).

On constat aussi d'après le graphe ci-dessous que l'écart entre la série d'ajustement et la série brut est très faible et les deux graphes sont quasi confondue.



Graphe de la série avec son ajustement et la prévision.

XXXIII.1.5 Etude de la complexité de cet algorithme :

XXXIII.1.5.1 Calcul de la complexité de la fonction prévision :

Cette fonction est constituée d'une boucle « For » à son intérieur une autre boucle « For » imbriquée, et des opérations indépendantes des boucles.

XXXIII.1.5.1.1 Complexité de la boucle For :

Cette dernière contient à l'intérieur une boucle imbriquée, et d'autres opérations indépendantes de la boucle imbriquée, et la formule pour calculer sa complexité est :

$$T_{Al}(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)+37\right)$$ Ou :

T_1 : Est la complexité de la boucle « For » imbriquée.

T_c : Nombre d'affectation d'indice.

37 : Nombre d'opérations dans la boucle « For » indépendamment de la boucle imbriquée.

Donc : la complexité de la boucle imbriquée est :

$$T_1(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)\right)=\sum_{i=1}^{n}(1+15)=\sum_{i=1}^{n}16=16\text{n.}$$

La complexité de la boucle « For » est :

$$T_{Al}(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)+37\right)=T_{Al}(n)=\sum_{i=1}^{n}(1+16n+37)=\sum_{i=1}^{n}1+16n\sum_{i=1}^{n}1+37\sum_{i=1}^{n}1$$

$$T_{Al}(n)=16n^2+38n$$

XXXIII.1.5.1.2 La complexité totale de la fonction prévision :

On a : 33 est le nombre d'opérations en dehors de la boucle « For » donc la complexité total de la fonction prévision est :

Op_p (n)$=33+T_{Al}(n)=33+38n+16n^2$

XXXIII.1.5.2 Calcul de la complexité de la fonction Treillis :

Cette fonction est constituée de plusieurs opérations indépendamment des boucles, une boucle « For » et d'une grande boucle « For » qui contiennent 3 autres boucles « For » imbriquées comme suite :

La boucle n°3 est imbriquée dans la boucle n°2 qui est imbriquée dans la boucle n°1 et les trois boucles sont imbriquées dans la grande boucle « For ».

Soit :

Op_1 (n) le nombre des opérations dans la grande boucle « For ».

T_1 (i, n) le nombre des opérations dans la boucle « For » n°1.

T_2 (i, n) le nombre des opérations dans la boucle « For » n°2.

T_3 (i, n) le nombre des opérations dans la boucle « For » n°3.

XXXIII.1.5.2.1 Calcul du nombre d'opérations dans la grande boucle « For » :

T_3 (i, n)= $\sum_{i=1}^{n} 6 = 6n$

T_2 (i, n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n} 6)+1+1) = \sum_{i=1}^{n}(6n+2) = n(6n+2) = 2n+6n^2$

T_1 (i, n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n} 6)+1+1))+1) = \sum_{i=1}^{n}(2n+6n^2+1) = n+2n^2+6n^3$

Donc le nombre d'opérations dans la grande boucle « For » est :

Op_1 (n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n} 6)+1+1))+1))+1) = \sum_{i=1}^{n}(n+2n^2+6n^3+1) = n+n^2+2n^3+6n^4$

Op_1 (n)= $n+n^2+2n^3+6n^4$

XXXIII.1.5.2.2 Calcul d'opérations en dehors de la grande boucle « For » :

Soit Op_2 (n) le nombre des opérations dans la fonction Treillis en dehors de la grande boucle « For ».

T (n)= $\sum_{i=1}^{n}(T_C(i,n)+T(i,n))$ est le nombre d'opérations dans la boucle « For » qui est indépendante de la grande boucle « For ».

6 : est le nombre d'opérations indépendamment de toutes les boucle qui existent dans la fonction treillis.

Donc :

Op_2 (n)= 6+T (n)=6+ $\sum_{i=1}^{n} 2 = 6+2n$

Donc le nombre d'opérations total dans la fonction treillis est :

Op_t (n)= Op_1 (n) + Op_2 (n)= $n+n^2+2n^3+6n^4+6+2n = 6+3n+n^2+2n^3+6n^4$

XXXIII.1.5.3 Calcul de la complexité de la fonction RBF :

XXXIII.1.5.3.1 Calcul du nombre d'opérations dans la boucle While :

Soit T_A (n) le nombre d'opérations dans la boucle While tel que :

$T_A(n) = \left(\sum_{i=1}^{n} \left(T_C(i,n) + T_{A1}(i,n) + T_{A2}(i,n) + T_{A3}(i,n) + 308 + op_p(n) + 1 \right) \right) + T_C(n+1,n)$ Tel

que :

T_c : Nombre de test de la condition C.

T_{A1} : Nombre d'opérations dans la grande boucle For, cette dernière contient a l'intérieur deux boucles « For » imbriquées (T_1 et T_2).

T_{A2} : Nombre d'opérations dans la condition « if ».

T_{A3} : Nombre d'opérations dans la condition « if ».

298 : Nombre d'opérations dans la boucle While indépendamment des boucles imbriquées.

Op_p (n) : Nombre d'opérations dans la fonction prévision.

1 : représente l'appel de la fonction prévision car l'appel est compté comme une opération.

On a :

$T_C(i,n) = 1$.

XXXIII.1.5.3.2 Calcul du nombre d'opération dans la grande boucle « For » :

Dans cet algorithme la grande boucle « For » est constituée d'une action conditionnelle de la forme : if (condition) action1 else action2 end if.

Donc d'après les formules illustrées ci-dessus le nombre d'opérations dans la grande boucle « For » est :

$T_{A1}(n) = \sum_{i=1}^{n} (T_C(n) + Ts) = \sum_{i=1}^{n} (T_C(n) + (1 + \text{Max}(Ta_1(n), Ta_2(n))))$ tel que :

Ta_1 : Est le nombre d'opérations dans l'action1 (a1).

Ta_2 : Est le nombre d'opérations dans l'action2 (a2).

T_c : Nombre d'affectation d'indice de la grande boucle « For ».

Ts : Est le nombre d'opérations dans la structure if (condition) action1 else action2 end if.

Soit :

$Ts = 1 + \text{Max}(Ta_1(n), Ta_2(n))$

$Ta_1(n) = T_1(n) + T_2(n) + T_3(n)$ tel que :

T_1 : est le nombre d'opération dans la première boucle de l'action 1.

T_2 : est le nombre d'opération dans la deuxième boucle de l'action 1.

T_3 : est le nombre d'opération en dehors des deux boucles «For » de l'action 1.

Donc :

$Ta_1(n) = \left(\sum_{i=1}^{n} (T_{C1}(i,n) + T_1(i,n)) \right) + \left(\sum_{i=1}^{n} (T_{C2}(i,n) + T_2(i,n)) \right) + T_3(i,n)$ Tel que :

$Ta_1 = 35 + \sum_{i=1}^{n} 16 + \sum_{i=1}^{n} 5 = 35 + 16n + 5n = 35 + 21n$.

De même pour la deuxième action :

$Ta_2(n) = \left(\sum_{i=1}^{n} (T_{C1}(i,n) + T_1(i,n)) \right) + \left(\sum_{i=1}^{n} (T_{C2}(i,n) + T_2(i,n)) \right) + T_3(i,n)$

$Ta_2 = 35 + \sum_{i=1}^{n} 16 + \sum_{i=1}^{n} 5 = 35 + 16n + 5n = 35 + 21n$.

Donc $Ts = 1 + \text{Max}((35 + 21n), (35 + 21n)) = 1 + 35 + 21n = 36 + 21n$

$Ts = 36 + 21n$

Le nombre d'opérations dans la grande boucle « For » est :

$T_{A1}(n) = \sum_{i=1}^{n} (T_C(n) + (1 + \text{Max}(Ta_1(n), Ta_2(n)))) = \sum_{i=1}^{n} (1 + 36 + 21n)$

$T_{A1}(n) = \sum_{i=1}^{n} 37 + \sum_{i=1}^{n} 21n = 37n + 21n^2$

En dehors de la grande boucle « For » on trouve deux conditions « if » :

XXXIII.1.5.3.3 Le nombre d'opérations dans la 1ere condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

$T_{A2}(n) = T_C(n) + \text{Max}(T_{a1}(n), T_{a2}(n))$.

Dans le cas de notre algorithme l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'affectations, donc notre équation devienne :

$T_{A2}(n) = T_C(n) + T_{a1}(n)$.

$T_{A2}(n) = 1 + 7 = 8$

XXXIII.1.5.3.4 Le nombre d'opérations dans la 2eme condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

$T_{A3}(n) = T_C(n) + \text{Max}(T_{a1}(n), T_{a2}(n))$.

De même pour la deuxième boucle « if » l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'instructions, donc notre équation devienne :

$T_{A3}(n) = T_C(n) + T_{a1}(n)$.

$T_{A3}(n) = 1 + 12 = 13$.

Donc le nombre total d'opérations dans la boucle While est :

$$T_A(n) = \left(\sum_{i=1}^{n} \left(T_C(i,n) + T_{A1}(i,n) + T_{A2}(i,n) + T_{A3}(i,n) + 298 + op_p(n) + 1 \right) \right) + T_C(n+1,n) =$$

$$\left(\sum_{i=1}^{n} (1 + 37n + 21n^2 + 8 + 13 + 298 + 33 + 38n + 16n^2 + 1) \right) + 1 = \left(\sum_{i=1}^{n} (354 + 75n + 37n^2) \right) + 1$$

$$T_A(n) = 1 + 354n + 75n^2 + 37n^3$$

Le nombre total d'opération **Op(n)** de la fonction principale RBF qui est aussi le nombre total d'opérations effectuées dans l'exécution de l'algorithme est:

$Op(n) = (Op_t(n)+1) + T_A(n) + 179$ tel que :

Op_t (n) : est le nombre d'opérations dans la fonction Treillis

T_A (n) : est le nombre d'opérations dans la boucle While de la fonction RBF.

179 : est le nombre d'opérations dans la fonction RBF indépendamment des boucle et des autres fonctions.

1 : est l'opération d'appel de la fonction Treillis dans la fonction RBF car la fonction treillis est appelé qu'une seule fois dans l'exécution de l'algorithme.

Donc :

Op(n)$= 1 + 6 + 3n + n^2 + 2n^3 + 6n^4 + 1 + 354n + 75n^2 + 37n^3 + 179$

Op(n)$= 187 + 357n + 76n^2 + 39n^3 + 6n^4$

On a :

$$\lim_{n \to \infty} (187 + 357n + 76n^2 + 39n^3 + 6n^4)/n^4 = 6 \neq 0 \Rightarrow Op(n) \quad est\ en \quad \Theta(n^4)$$

XXXIII.1.5.4 Conclusion :

D'après les résultats ci-dessus qui ont été obtenus, l'algorithme qui a été utilisé dans cette application est d'une complexité polynomiale d'ordre 4, c'est-à-dire que notre algorithme est en $O(n^4)$.

XXXIII.2 2eme Cas : Algorithme de rétro propagation du Gradient

Dans cette section on utilisera l'algorithme de rétro propagation du Gradient pour la mise à jour des poids de connexions entre les neurones et cela selon la nouvelle idée comme suite :

$$w_{ji}(t+1) = w_{ji}(t) - \alpha(S_{pi}(t-1) - y_{pi}(t-1))\ \phi_{pj}(t) + \gamma \left[w_{ji}(t-1) - w_{ji}(t) \right]$$ Tel que:

$w_{ji}(t+1)$: Le poids reliant le neurone j à i à l'instant t+1.

$w_{ji}(t)$: Le poids reliant le neurone j à i à l'instant t.

$w_{ji}(t-1)$: Le poids reliant le neurone j à i à l'instant t-1.

α : Un paramètre d'adaptation de l'apprentissage (α >0).

$S_{pi}(t-1)$: La sortie du réseau à l'instant t-1.

$y_{pi}(t-1)$: La sortie désirée à l'instant t-1.

p : représente l'exemple courant.

γ **:** est un facteur d'oubli inférieur à 1.

XXXIII.2.1 Le choix d'architecture de réseau à fonction de base radiale (Rbf) :

De la même manière que la première partie de l'application on a choisi l'architecture qui donne une erreur minimale au sens de SCR_p (la somme des carrés des résidus de la prévision).

Donc d'après les expériences qui ont été menées sur plusieurs types d'architectures, le réseau Rbf à trois entrées et a trois neurones cachés donne le meilleur résultat (SCR_p = **0.11697**).

XXXIII.2.2 Réseau à fonction de base radiale à 3 entrées

Taille de la Base d'apprentissage : 144

Taille de la Base de test : 6

Nombres d'itérations : 172

Nombres de neurones dans la couche cachée : 3

Les figures (1, 2,3) illustrent l'apprentissage qui consiste à chercher la configuration de poids qui minimise l'erreur ; ces figures représentent les graphes des fonctions suivantes :

- la série et son ajustement en utilisant les poids de la dernière itération.
- la somme des carrés des résidus de l'ajustement (SCR).
- la somme des carrés des résidus de la prévision (SCRp).

Deuxième itération :

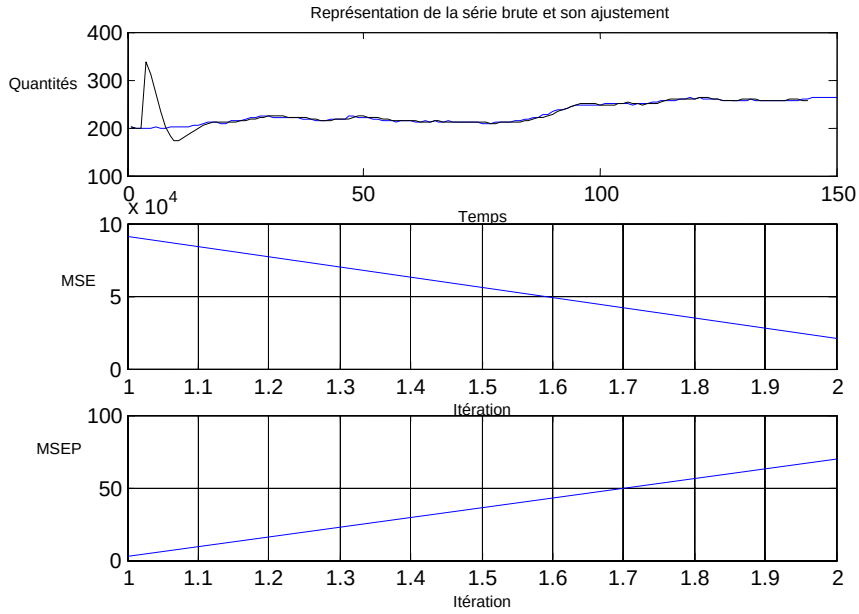


Figure 1

$$SCR = \frac{1}{2}\sum_{t=1}^{144} e_t^2 = 20731$$

$$SCR_p = \frac{1}{2}\sum_{t=145}^{150} e_t^2 = 69.211$$

Itération intermédiaire :

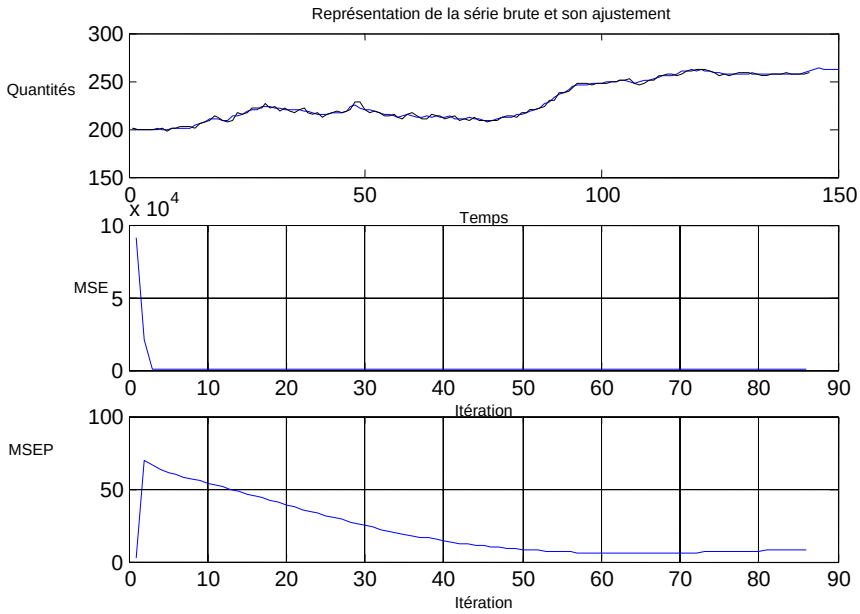


Figure 2

SCR= 225.15

SCRp= 8.1688

Dernière itération :

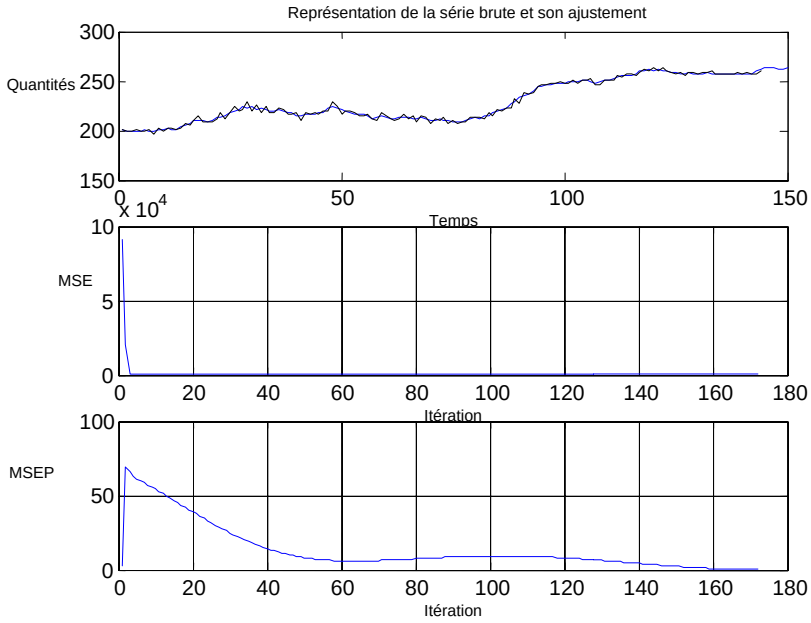


Figure 3

SCR= 333.67

SCRp= 0.11697

XXXIII.2.3 Test d'arrêt :

Théoriquement : Sur la base d'apprentissage, l'erreur diminue constamment, alors que sur la base de test elle passe par un minimum. Si l'apprentissage se prolonge au delà, les performances en test diminuent.

Donc, d'après les résultats issues de l'application de la nouvelle technique du principe du mode d'apprentissage supervisé, qui sont affichés dans le tableau ci-dessous on remarque que :

L'erreur quadratique de l'ajustement (SCR) augmente sans cesse au cours de l'apprentissage contrairement a ce qui a été dit dans la partie théorique concernant le mode d'apprentissage supervisé, par contre l'erreur quadratique de la prévision (SCR_P) passe par un minimum a **l'itération 172**, ce qui est conforme à ce qui a été écrit dans la partie théorique, Chose qu'on a pas pu le voir graphiquement, ce qui peut être expliqué par :

- taille de l'échantillon (faible).
- la série chronologique est régulière.

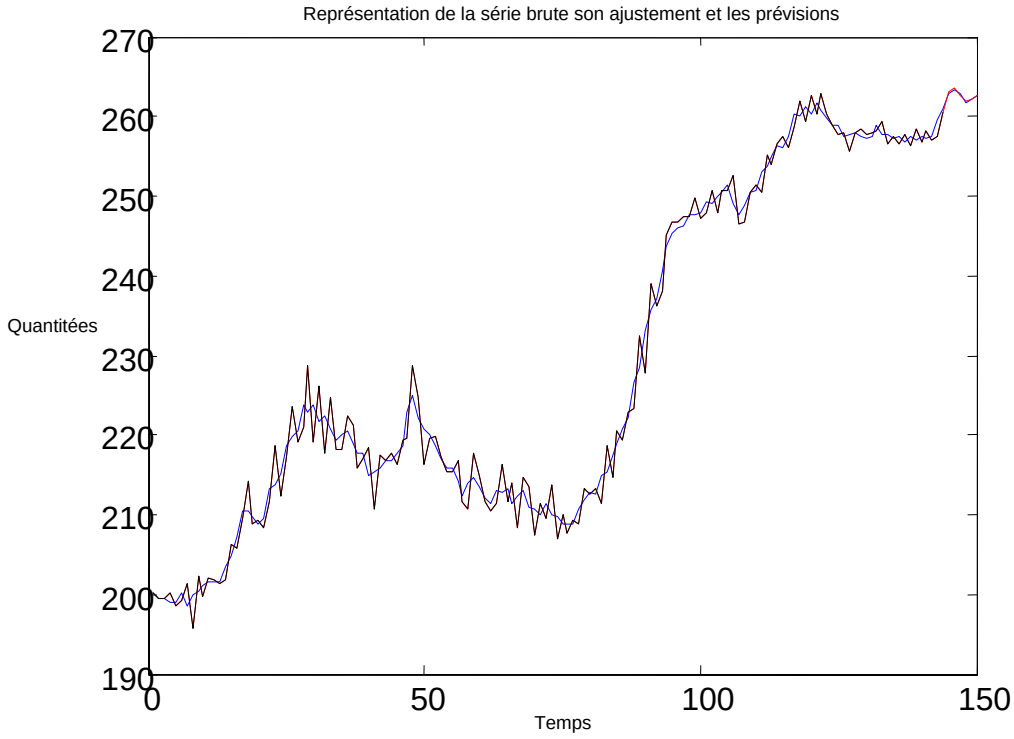
SCRP	SCR	ITERATION
0.24469	319.45	167
0.19724	322.17	168
0.16045	324.96	169
0.13461	327.8	170
0.12002	330.71	171
0.11697	**333.67**	**172**
0.12574	336.69	173
0.14662	339.78	174
0.17986	342.93	175
0.22573	346.14	176
0.2845	349.42	178

XXXIII.2.4 Prévision :

BRUT	PREVI	ERRE
262.9	263.1752	-0,2752
263.3	263.5191	-0,2191
262.8	262.6598	0,1402
261.8	262.0317	-0,2317
262.2	262.0923	0,1077
262.7	262.5410	0,159

D'après le tableau ci-dessus, on remarque que les écarts entres les prévisions obtenues par le réseau de neurones et les valeurs de série Brut sont très faibles, les valeurs sont presque confondues, d'ailleurs on peut le constater graphiquement (graphe ci-dessous).

On constat aussi d'après le graphe ci-dessous que l'écart entre la série d'ajustement et la série brut est très faible et les deux graphes sont quasi confondue.



Graphe de la série avec son ajustement et la prévision.

XXXIII.2.5 Etude de la complexité de cet algorithme :

XXXIII.2.5.1 Calcul de la complexité de la fonction prévision :

Cette fonction est constituée d'une boucle « For » à son intérieur une autre boucle « For » imbriquée, et des opérations indépendantes des boucles.

XXXIII.2.5.1.1 Complexité de la boucle For :

Cette dernière contient à l'intérieur une boucle imbriquée, et d'autres opérations indépendantes de la boucle imbriquée, et la formule pour calculer sa complexité est :

$$T_{Al}(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)+113\right)$$ Ou :

T_1 : Est la complexité de la boucle « For » imbriquée.

T_c : Nombre d'affectation d'indice.

113 : Nombre d'opérations dans la boucle « For » indépendamment de la boucle imbriquée.

Donc : la complexité de la boucle imbriquée est :

$$T_1(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)\right)=\sum_{i=1}^{n}(1+15)=\sum_{i=1}^{n}16=16\text{n}.$$

La complexité de la boucle « For » est :

$$T_{A1}(n)=\sum_{i=1}^{n}(T_C(i,n)+T_1(i,n)+113)=T_{A1}(n)=\sum_{i=1}^{n}(1+16n+113)=\sum_{i=1}^{n}1+16n\sum_{i=1}^{n}1+113\sum_{i=1}^{n}1$$

$$T_{A1}(n)=16n^2+114n$$

XXXIII.2.5.1.2 La complexité totale de la fonction prévision :

On a : 33 est le nombre d'opérations en dehors de la boucle « For » donc la complexité total de la fonction prévision est :

Op_p (n)$=33+T_{A1}(n)=33+114n+16n^2$

XXXIII.2.5.2 Calcul de la complexité de la fonction Treillis :

Cette fonction est constituée de plusieurs opérations indépendamment des boucles, une boucle « For » et d'une grande boucle « For » qui contiennent 3 autres boucles « For » imbriquées comme suite :

La boucle n°3 est imbriquée dans la boucle n°2 qui est imbriquée dans la boucle n°1 et les trois boucles sont imbriquées dans la grande boucle « For ».

Soit :

Op_1 (n) le nombre des opérations dans la grande boucle « For ».

T_1 (i, n) le nombre des opérations dans la boucle « For » n°1.

T_2 (i, n) le nombre des opérations dans la boucle « For » n°2.

T_3 (i, n) le nombre des opérations dans la boucle « For » n°3.

XXXIII.2.5.2.1 Calcul du nombre d'opérations dans la grande boucle « For » :

$$T_3(i,n)=\sum_{i=1}^{n}6=6n$$

$$T_2(i,n)=\sum_{i=1}^{n}((\sum_{i=1}^{n}6)+1+1)=\sum_{i=1}^{n}(6n+2)=n(6n+2)=2n+6n^2$$

$$T_1(i,n)=\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}6)+1+1))+1)=\sum_{i=1}^{n}(2n+6n^2+1)=n+2n^2+6n^3$$

Donc le nombre d'opérations dans la grande boucle « For » est :

$$Op_1(n)=\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}6)+1+1))+1))+1)=\sum_{i=1}^{n}(n+2n^2+6n^3+1)=n+n^2+2n^3+6n^4$$

$$Op_1(n)=n+n^2+2n^3+6n^4$$

XXXIII.2.5.2.2 Calcul d'opérations en dehors de la grande boucle « For » :

Soit Op_2 (n) le nombre des opérations dans la fonction Treillis en dehors de la grande boucle « For ».

T (n)= $\sum_{i=1}^{n}(T_C(i,n)+T(i,n))$ est le nombre d'opérations dans la boucle « For » qui est indépendante de la grande boucle « For ».

6 : est le nombre d'opérations indépendamment de toutes les boucle qui existent dans la fonction treillis.

Donc : Op_2 (n)= 6+T (n)=6+ $\sum_{i=1}^{n}2$ =6+ $2n$

Donc le nombre d'opérations total dans la fonction treillis est :

Op_t (n)= Op_1 (n) + Op_2 (n)= $n+n^2+2n^3+6n^4+6+2n=6+3n+n^2+2n^3+6n^4$

XXXIII.2.5.3 Calcul de la complexité de la fonction RBF :

XXXIII.2.5.3.1 Calcul du nombre d'opérations dans la boucle While :

Soit T_A (n) le nombre d'opérations dans la boucle While tel que :

$$T_A(n)=\left(\sum_{i=1}^{n}\left(T_C(i,n)+T_{A1}(i,n)+T_{A2}(i,n)+T_{A3}(i,n)+308+op_p(n)+1\right)\right)+T_C(n+1,n)$$ Tq :

T_c : Nombre de test de la condition C.

T_{A1} : Nombre d'opérations dans la grande boucle For, cette dernière contient a l'intérieur deux boucles « For » imbriquées (T_1 et T_2).

T_{A2} : Nombre d'opérations dans la condition « if ».

T_{A3} : Nombre d'opérations dans la condition « if ».

298 : Nombre d'opérations dans la boucle While indépendamment des boucles imbriquées.

Op_p (n) : Nombre d'opérations dans la fonction prévision.

1 : représente l'appel de la fonction prévision car l'appel est compté comme une opération.

On a : $T_C(i,n)=1$.

XXXIII.2.5.3.2 Calcul du nombre d'opération dans la grande boucle « For » :

Dans cet algorithme la grande boucle « For » est constituée d'une action conditionnelle de la forme : if (condition) action1 else action2 end if.

Donc d'après les formules illustrées ci-dessus le nombre d'opérations dans la grande boucle « For » est :

T_{A1} (n)= $\sum_{i=1}^{n}$(T_C (n) + Ts)= $\sum_{i=1}^{n}$(T_C (n) + (1+ Max (Ta_1 (n), Ta_2 (n)))) tel que :

Ta_1 : Est le nombre d'opérations dans l'action1 (a1).

Ta_2 : Est le nombre d'opérations dans l'action2 (a2).

T_c : Nombre d'affectation d'indice de la grande boucle « For ».

Ts : Est le nombre d'opérations dans la structure if (condition) action1 else action2 end if.

Soit :

Ts=1+ Max (Ta_1 (n), Ta_2 (n))

$Ta_1(n)$= T_1 (n) +T_2 (n) +T_3 (n) tel que :

T_1 : est le nombre d'opération dans la première boucle de l'action 1.

T_2 : est le nombre d'opération dans la deuxième boucle de l'action 1.

T_3 : est le nombre d'opération en dehors des deux boucles «For » de l'action 1.

Donc :

$$Ta_1(n) = \left(\sum_{i=1}^{n} (T_{C1}(i,n) + T_1(i,n)) \right) + \left(\sum_{i=1}^{n} (T_{C2}(i,n) + T_2(i,n)) \right) + T_3(i,n)$$ Tel que :

$$Ta_1 = 111 + \sum_{i=1}^{n} 16 + \sum_{i=1}^{n} 5 = 111 + 16n + 5n = 111 + 21n.$$

De même pour la deuxième action :

$$Ta_2(n) = \left(\sum_{i=1}^{n} (T_{C1}(i,n) + T_1(i,n)) \right) + \left(\sum_{i=1}^{n} (T_{C2}(i,n) + T_2(i,n)) \right) + T_3(i,n)$$

$$Ta_2 = 111 + \sum_{i=1}^{n} 16 + \sum_{i=1}^{n} 8 = 111 + 16n + 8n = 111 + 24n.$$

Donc $Ts = 1 + \max((111 + 21n), (111 + 24n)) = 1 + 111 + 24n = 112 + 24n$

$Ts = 112 + 24n$

Le nombre d'opérations dans la grande boucle « For » est :

$$T_{A1}(n) = \sum_{i=1}^{n} (T_C(n) + (1 + \max(Ta_1(n), Ta_2(n)))) = \sum_{i=1}^{n} (1 + 112 + 24n)$$

$$T_{A1}(n) = \sum_{i=1}^{n} 113 + \sum_{i=1}^{n} 24n = 113n + 24n^2$$

En dehors de la grande boucle « For » on trouve deux conditions « if » :

XXXIII.2.5.3.3 Le nombre d'opérations dans la 1ere condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

$T_{A2}(n) = T_C(n) + \max(T_{a1}(n), T_{a2}(n))$.

Dans le cas de notre algorithme l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'affectations, donc notre équation devienne :

$T_{A2}(n) = T_C(n) + T_{a1}(n)$.

$T_{A2}(n) = 1+7=8$

XXXIII.2.5.3.4 Le nombre d'opérations dans la 2eme condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

$T_{A3}(n) = T_C(n) + \text{Max}(T_{a1}(n), T_{a2}(n))$.

De même pour la deuxième boucle « if » l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'instructions, donc notre équation devienne :

$T_{A3}(n) = T_C(n) + T_{a1}(n)$.

$T_{A3}(n) = 1 + 12 = 13$.

Donc le nombre total d'opérations dans la boucle While est :

$$T_A(n) = \left(\sum_{i=1}^{n} \left(T_C(i,n) + T_{A1}(i,n) + T_{A2}(i,n) + T_{A3}(i,n) + 297 + op_p(n) + 1 \right) \right) + T_C(n+1,n) =$$

$$\left(\sum_{i=1}^{n} (1 + 113n + 24n^2 + 8 + 13 + 298 + 33 + 114n + 16n^2 + 1)) + 1 = \left(\sum_{i=1}^{n} (354 + 127n + 40n^2 \right) \right) + 1$$

$$T_A(n) = 1 + 354n + 127n^2 + 40n^3$$

Le nombre total d'opération **Op(n)** de la fonction principale RBF qui est aussi le nombre total d'opérations effectuées dans l'exécution de l'algorithme est:

$Op(n) = (Op_t(n)+1) + T_A(n) + 189$ tel que :

$Op_t(n)$: est le nombre d'opérations dans la fonction Treillis

$T_A(n)$: est le nombre d'opérations dans la boucle While de la fonction RBF.

198 : est le nombre d'opérations dans la fonction RBF indépendamment des boucle et des autres fonctions.

1 : est l'opération d'appel de la fonction Treillis dans la fonction RBF car la fonction treillis est appelé qu'une seule fois dans l'exécution de l'algorithme.

Donc :

Op(n)$= 1 + 6 + 3n + n^2 + 2n^3 + 6n^4 + 1 + 354n + 127n^2 + 40n^3 + 198$

Op(n)$= 206 + 357n + 128n^2 + 42n^3 + 6n^4$

On a:

$$\lim_{n \to \infty} (206 + 357n + 128n^2 + 42n^3 + 6n^4)/n^4 = 6 \neq 0 \Rightarrow Op(n) \quad est\ en \quad \Theta(n^4)$$

XXXIII.2.5.4 Conclusion:

D'après les résultats ci-dessus qui ont été obtenus, l'algorithme qui a été utilisé dans cette application est d'une complexité polynomiale d'ordre 4, c'est-à-dire que notre algorithme est en $O(n^4)$.

Deuxieme Méthode

Dans cette deuxième méthode de la 2eme partie de l'application les paramètres (centres et rayons) seront variables durant le processus d'apprentissage de la même façon définie dans la première partie de l'application (méthode descente de gradient), c'est-à-dire que l'apprentissage ne se fera pas seulement par la modification des poids de connexions entre les neurones mais il se fera aussi par la modification de la position des centres et la modification de la largeur des noyaux.

Donc, dans cette méthode l'apprentissage se fera par l'ajustement de trois paramètres (W, C et σ) au lieu d'un seul paramètre (w) dans la première méthode.

XXXIV 1^{ere} Cas d'application :

Dans ce premier cas d'application l'apprentissage se fera sur les trois paramètres W, C et σ comme suite :

L'ajustement des Centres et des rayons (Ci, σ_i) s'effectue par la méthode de descente de gradient qui a été illustrée ci-dessus.

L'ajustement des poids W_i se fera par la règle de Widdrow-Hoff définie dans la première méthode ci-dessus selon la nouvelle idée.

XXXIV.1 Le choix d'architecture de réseau à fonction de base radiale (Rbf) :

Le choix de l'architecture du réseau se fait de la même façon que la première méthode de 2eme partie de l'application ci-dessus, c'est-à-dire qu'on choisi l'architecture qui donne une erreur minimale au sens de SCR_p (la somme des carrés des résidus de la prévision) car notre but est de calculé une prévision. Donc d'après les expériences qui ont été menées sur plusieurs types d'architectures : Le réseau Rbf à quatre entrées et a deux neurones cachés donne le meilleur résultat (SCR_p = **0.55942**).

XXXIV.1.1 Réseau à fonction de base radiale à 4 entrées

Taille de la Base d'apprentissage : 144

Taille de la Base de test : 6

Nombres d'itérations : 35

Nombres de neurones dans la couche cachée : 2

Les figures (1, 2,3) illustrent l'apprentissage qui consiste à chercher la configuration de poids qui minimise l'erreur ; ces figures représentent les graphes des fonctions suivantes :

- la série et son ajustement en utilisant les poids de la dernière itération.
- la somme des carrés des résidus de l'ajustement (SCR).
- la somme des carrés des résidus de la prévision (SCRp).

Deuxième itération :

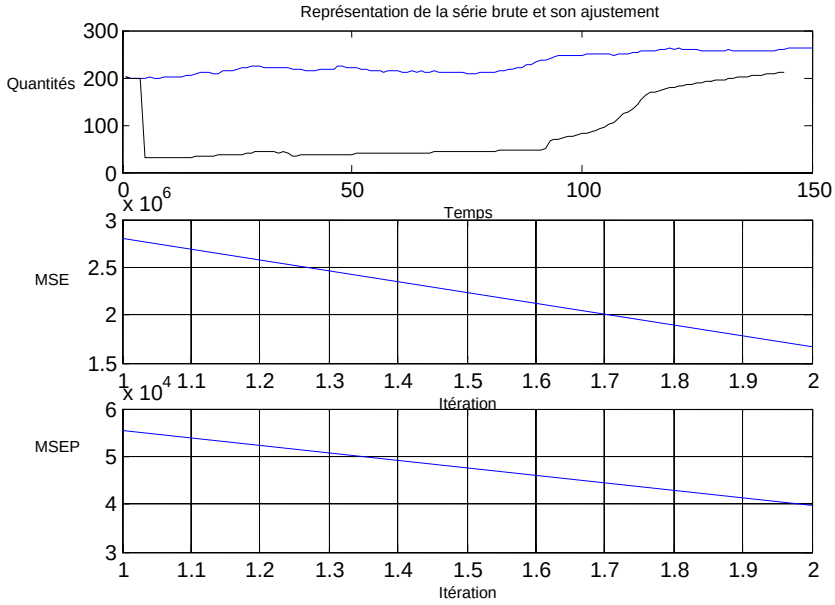


Figure 1

$$SCR = \frac{1}{2}\sum_{t=1}^{144} e_t^2 = 1.6718e{+}006$$

$$SCR_p = \frac{1}{2}\sum_{t=145}^{150} e_t^2 = 39749$$

Itération intermédiaire :

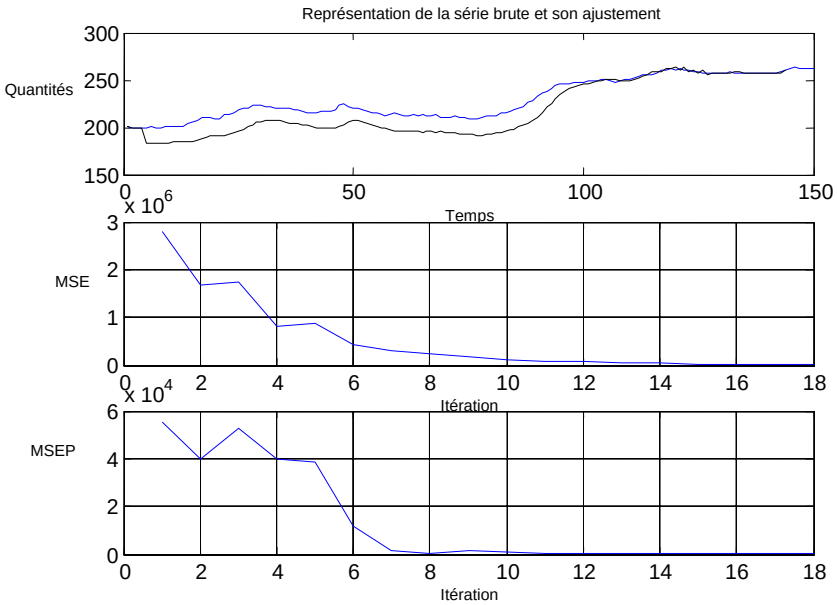


Figure 2

SCR= 14289

SCRp= 49.958

Dernière itération :

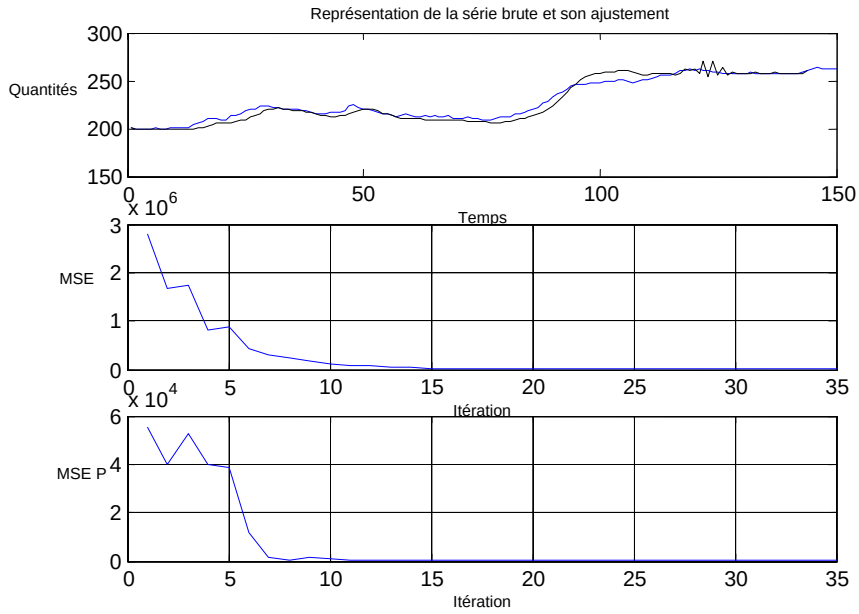


Figure 3

SCR= 1672.4

SCRp= **0.55942**

XXXIV.1.2 Test d'arrêt :

Théoriquement : Sur la base d'apprentissage, l'erreur diminue constamment, alors que sur la base de test elle passe par un minimum. Si l'apprentissage se prolonge au delà, les performances en test diminuent.

Donc, d'après les résultats issues de l'application de la nouvelle technique du principe du mode d'apprentissage supervisé, qui sont affichés dans le tableau ci-dessous on remarque que :

L'erreur quadratique de l'ajustement (SCR) augmente sans cesse au cours de l'apprentissage contrairement a ce qui a été dit dans la partie théorique concernant le mode d'apprentissage supervisé, par contre l'erreur quadratique de la prévision (SCR_P) passe par un minimum a **l'itération 35**, ce qui est conforme à ce qui a été écrit dans la partie théorique, Chose qu'on a pas pu le voir graphiquement, ce qui peut être expliqué par :

- taille de l'échantillon (faible).
- la série chronologique est régulière.

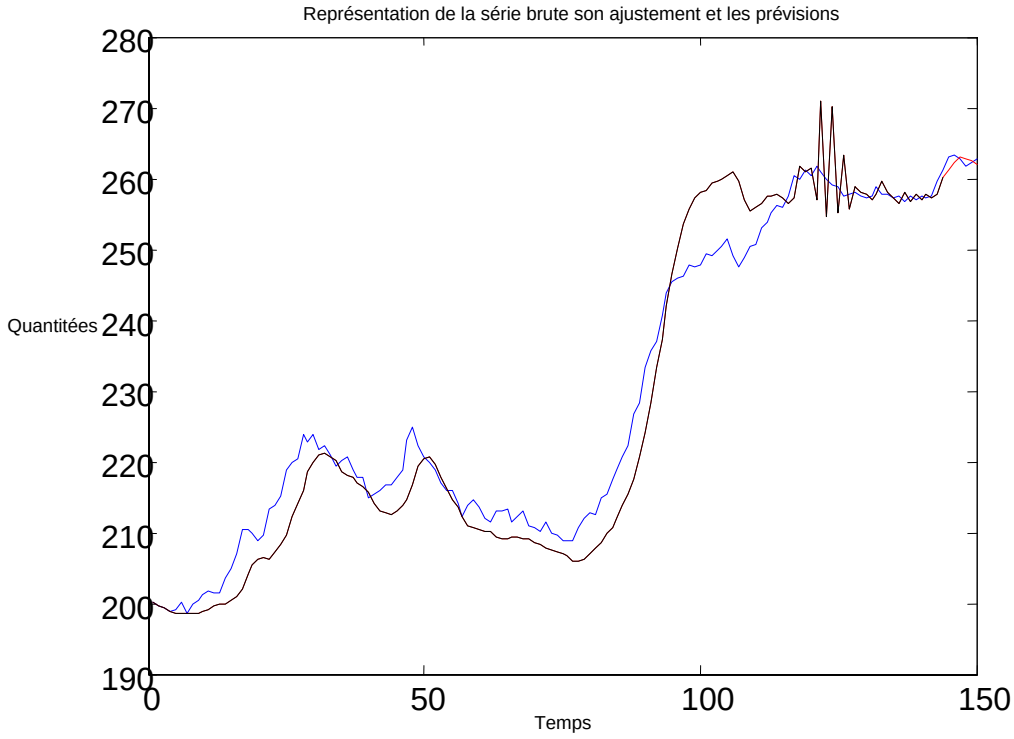
SCRP	SCR	ITERATION
0.72	2188.8	30
0.6462	2043.8	31
0.6027	1925.1	32
0.57773	1826.5	33
0.56448	1743.4	34
0.55942	**1672.4**	**35**
0.56108	1610.9	36
0.56909	1556.8	37
0.58366	1508.8	38
0.60523	1465.5	39
0.63417	1426.2	40

XXXIV.1.3 Prévision :

BRUT	PREVI	ERRE
262.9	261.1219	1,7781
263.3	262.3380	0,962
262.8	262.9341	-0,1341
261.8	262.8456	-1,0456
262.2	262.6093	-0,4093
262.7	262.0917	0,6083

D'après le tableau ci-dessus, on remarque que les écarts entres les prévisions obtenues par le réseau de neurones et les valeurs de série Brut sont faibles, d'ailleurs on peut le constater graphiquement (graphe ci-dessous).

On constat aussi d'après le graphe ci-dessous que l'écart entre la série d'ajustement et la série brut est très perceptible et les deux graphes sont presque parallèle sauf dans quelques régions ou ils se croisent et à la fin de la série d'ajustement.



Graphe de la série avec son ajustement et la prévision.

XXXIV.1.4 Etude de la complexité de cet algorithme :

XXXIV.1.4.1 Calcul de la complexité de la fonction prévision :

Cette fonction est constituée d'une boucle « For » a son intérieur une autre boucle « For » imbriquée, et des opérations indépendantes des boucles.

XXXIV.1.4.1.1 Complexité de la boucle For :

Cette dernière contient à l'intérieur une boucle imbriquée, et d'autres opérations indépendantes de la boucle imbriquée, et la formule pour calculer sa complexité est :

$$T_{Al}(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)+84\right)$$ Ou :

T_1 : Est la complexité de la boucle « For » imbriquée.

T_c : Nombre d'affectation d'indice.

84 : Nombre d'opérations dans la boucle « For » indépendamment de la boucle imbriquée.

Donc : la complexité de la boucle imbriquée est :

$$T_1(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)\right)=\sum_{i=1}^{n}(1+32)=\sum_{i=1}^{n}33=33n.$$

La complexité de la boucle « For » est :

$$T_{A1}(n)=\sum_{i=1}^{n}(T_C(i,n)+T_1(i,n)+84)=T_{A1}(n)=\sum_{i=1}^{n}(1+33n+84)=\sum_{i=1}^{n}1+33n\sum_{i=1}^{n}1+84\sum_{i=1}^{n}1=$$

$$33n^2+85n$$

XXXIV.1.4.1.2 La complexité totale de la fonction prévision :

On a : 35 est le nombre d'opérations en dehors de la boucle « For » donc la complexité total de la fonction prévision est :

Op_p (n)$=35+T_{A1}(n)=35+85n+33n^2$

XXXIV.1.4.2 Calcul de la complexité de la fonction Treillis :

Cette fonction est constituée de plusieurs opérations indépendamment des boucles, une boucle « For » et d'une grande boucle « For » qui contiennent 4 autres boucles « For » imbriquées comme suite :

La boucle n°4 est imbriquée dans la boucle n°3 qui est imbriquée dans la boucle n°2 cette dernière est imbriquée dans la boucle n°1 et les quatre boucles sont imbriquées dans la grande boucle « For ».

Soit :

Op_1 (n) le nombre des opérations dans la grande boucle « For ».

T_1 (i, n) le nombre des opérations dans la boucle « For » n°1.

T_2 (i, n) le nombre des opérations dans la boucle « For » n°2.

T_3 (i, n) le nombre des opérations dans la boucle « For » n°3.

T_4 (i, n) le nombre des opérations dans la boucle « For » n°4.

XXXIV.1.4.2.1 Calcul du nombre d'opérations dans la grande boucle « For » :

T_4 (i, n)= $\sum_{i=1}^{n}6=6n$

T_3 (i, n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n}6)+1+1)=\sum_{i=1}^{n}(6n+2)=n(6n+2)\ =2n+6n^2$

T_2 (i, n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}6)+1+1))+1)=\sum_{i=1}^{n}(2n+6n^2+1)=n+2n^2+6n^3$

T_1 (i, n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}6)+1+1))+1))+1)=\sum_{i=1}^{n}(n+2n^2+6n^3+1)=n+n^2+2n^3+6n^4$

Donc le nombre d'opérations dans la grande boucle « For » est :

Op_1 (n)= $\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}6)+1+1))+1))+1))+1)=\sum_{i=1}^{n}(1+n+n^2+2n^3+6n^4)$

$Op_1(n) = n + n^2 + n^3 + 2n^4 + 6n^5$

XXXIV.1.4.2.2 Calcul d'opérations en dehors de la grande boucle « For » :

Soit $Op_2(n)$ le nombre des opérations dans la fonction treillis en dehors de la grande boucle « For ».

$T(n) = \sum_{i=1}^{n} (T_C(i,n) + T(i,n))$ est le nombre d'opérations dans la boucle « For » qui est indépendante de la grande boucle « For ».

6 : est le nombre d'opérations indépendamment de toutes les boucle qui existent dans la fonction treillis.

Donc : $Op_2(n) = 6 + T(n) = 6 + \sum_{i=1}^{n} 2 = 6 + 2n$

Donc le nombre d'opérations total dans la fonction treillis est :

$Op_t(n)$ $= Op_1(n) + Op_2(n) = n + n^2 + n^3 + 2n^4 + 6n^5 + 6 + 2n = 6 + 3n + n^2 + n^3 + 2n^4 + 6n^5$

XXXIV.1.4.3 Calcul de la complexité de la fonction RBF :

XXXIV.1.4.3.1 Calcul du nombre d'opérations dans la boucle While :

Soit $T_A(n)$ le nombre d'opérations dans la boucle While tel que :

$T_A(n) = \left(\sum_{i=1}^{n} \left(T_C(i,n) + T_{A1}(i,n) + T_{A2}(i,n) + T_{A3}(i,n) + 300 + op_p(n) + 1 \right) \right) + T_C(n+1,n)$ Tel que :

T_c : Nombre de test de la condition C.

T_{A1} : Nombre d'opérations dans la grande boucle For, cette dernière contient a l'intérieur deux boucles « For » imbriquées (T_1 et T_2).

T_{A2} : Nombre d'opérations dans la condition « if ».

T_{A3} : Nombre d'opérations dans la condition « if ».

296 : Nombre d'opérations dans la boucle While indépendamment des boucles imbriquées.

$Op_p(n)$: Nombre d'opérations dans la fonction prévision.

1 : représente l'appel de la fonction prévision car l'appel est compté comme une opération.

On a : $T_C(i,n) = 1$.

XXXIV.1.4.3.2 Calcul du nombre d'opération dans la grande boucle « For » :

Dans cet algorithme la grande boucle « For » est constituée d'une action conditionnelle de la forme : if (condition) action1 else action2 end if.

Donc d'après les formules illustrées ci-dessus le nombre d'opérations dans la grande boucle « For » est :

$T_{A1}(n)= \sum_{i=1}^{n}(T_C (n) + Ts)= \sum_{i=1}^{n}(T_C (n) + (1+ Max (Ta_1 (n), Ta_2 (n))))$ tel que :

Ta_1 : Est le nombre d'opérations dans l'action1 (a1).

Ta_2 : Est le nombre d'opérations dans l'action2 (a2).

T_c : Nombre d'affectation d'indice de la grande boucle « For ».

Ts : Est le nombre d'opérations dans la structure if (condition) action1 else action2 end if.

Soit : $Ts=1+ Max (Ta_1 (n), Ta_2 (n))$

$Ta_1(n)= T_1 (n) +T_2 (n) +T_3 (n)$ tel que :

T_1 : est le nombre d'opération dans la première boucle de l'action 1.

T_2 : est le nombre d'opération dans la deuxième boucle de l'action 1.

T_3 : est le nombre d'opération en dehors des deux boucles «For » de l'action 1.

Donc :

$$Ta_1(n)=\left(\sum_{i=1}^{n}(T_{C1}(i,n)+T_1(i,n)) \right)+\left(\sum_{i=1}^{n}(T_{C2}(i,n)+T_2(i,n)) \right)+T_3(i,n)$$ Tel que :

$$Ta_1=69+ \sum_{i=1}^{n}20 + \sum_{i=1}^{n}85=69+20n+85n=69+105n.$$

De même pour la deuxième action :

$$Ta_2(n)=\left(\sum_{i=1}^{n}(T_{C1}(i,n)+T_1(i,n)) \right)+\left(\sum_{i=1}^{n}(T_{C2}(i,n)+T_2(i,n)) \right)+T_3(i,n)$$

$$Ta_2=69+ \sum_{i=1}^{n}20 + \sum_{i=1}^{n}85=69+20n+85n=69+105n.$$

Donc $Ts=1+Max ((69+105n), (69+105n))=1+69+105n =70+105n$

$Ts=70+105n$

Le nombre d'opérations dans la grande boucle « For » est :

$$T_{A1}(n)= \sum_{i=1}^{n}(T_C (n) + (1+Max (Ta_1 (n), Ta_2 (n))))= \sum_{i=1}^{n}(1+ 70+105n)$$

$$T_{A1}(n)= \sum_{i=1}^{n}71+\sum_{i=1}^{n}105n=71n+105n^2$$

En dehors de la grande boucle « For » on trouve deux conditions « if » :

XXXIV.1.4.3.3 Le nombre d'opérations dans la 1ere condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

$T_{A2}(n) = T_C(n) + \text{Max}(T_{a1}(n), T_{a2}(n))$.

Dans le cas de notre algorithme l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'affectations, donc notre équation devienne :

$T_{A2}(n) = T_C(n) + T_{a1}(n)$.

$T_{A2}(n) = 1+7=8$

XXXIV.1.4.3.4 Le nombre d'opérations dans la 2eme condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

$T_{A3}(n) = T_C(n) + \text{Max}(T_{a1}(n), T_{a2}(n))$.

De même pour la deuxième boucle « if » l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'instructions, donc notre équation devienne :

$T_{A3}(n) = T_C(n) + T_{a1}(n)$.

$T_{A3}(n) = 1 +12=13$.

Donc le nombre total d'opérations dans la boucle While est :

$$T_A(n) = \left(\sum_{i=1}^{n} \left(T_C(i,n) + T_{A1}(i,n) + T_{A2}(i,n) + T_{A3}(i,n) + 296 + op_p(n) + 1 \right) \right) + T_C(n+1,n) =$$

$$\left(\sum_{i=1}^{n} (1+71n+105n^2 +8+13+296+35+85n+33n^2 +1)) \right) +1 = \left(\sum_{i=1}^{n} (354+156n+138n^2)) \right) +1$$

$$T_A(n) = 1 + 354n + 156n^2 + 138n^3$$

Le nombre total d'opération **Op(n)** de la fonction principale RBF qui est aussi le nombre total d'opérations effectuées dans l'exécution de l'algorithme est:

$Op(n) = (Op_t(n)+1)+T_A(n)+186$ tel que :

$Op_t(n)$: est le nombre d'opérations dans la fonction Treillis

$T_A(n)$: est le nombre d'opérations dans la boucle While de la fonction RBF.

186 : est le nombre d'opérations dans la fonction RBF indépendamment des boucle et des autres fonctions.

1 : est l'opération d'appel de la fonction Treillis dans la fonction RBF car la fonction treillis est appelé qu'une seule fois dans l'exécution de l'algorithme.

Donc :

Op(n)$= 1+ 6+3n + n^2 + n^3 + 2n^4 + 6n^5 +1 + 354n + 156n^2 + 138n^3 +186$

Op(n)$= 194 + 357n + 157n^2 + 139n^3 + 2n^4 + 6n^5$

On a :

$$\lim_{n\to\infty}(194 + 357n + 157n^2 + 139n^3 + 2n^4 + 6n^5)/n^5 = 6 \neq 0 \Rightarrow Op(n) \quad est\,en \quad \Theta(n^5)$$

XXXIV.1.5 Conclusion:

D'après les résultats ci-dessus qui ont été obtenus, l'algorithme qui a été utilisé dans cette application est d'une complexité polynomiale d'ordre 5, c'est-à-dire que notre algorithme est en $O(n^5)$.

XXXV 2eme Cas d'application :

Dans ce deuxième cas d'application et de la même manière l'apprentissage se fera sur les trois paramètres W, C et σ comme suite :

L'ajustement des Centres et des rayons (Ci, σ_i) s'effectue par la méthode de descente de gradient qui a été illustrée ci-dessus.

L'ajustement des poids W_i se fera par **la méthode de descente de gradient** définie dans la première méthode ci-dessus mais selon la nouvelle idée.

XXXV.1 Le choix d'architecture de réseau à fonction de base radiale (Rbf) :

Le choix de l'architecture du réseau se fait de la même façon que la première méthode de la 1eme première partie de l'application ci-dessus, c'est-à-dire qu'on choisi l'architecture qui donne une erreur minimale au sens de SCR_p (la somme des carrés des résidus de la prévision) car notre but est de calculé une prévision.

Donc d'après les expériences qui ont été menées sur plusieurs types d'architectures : Le réseau Rbf à trois (03) entrées et a trois (03) neurones cachés donne le meilleur résultat (SCR_p = **0.17067**).

XXXV.1.1 Réseau à fonction de base radiale à 3 entrées

Taille de la Base d'apprentissage : 144

Taille de la Base de test : 6

Nombres d'itérations : 154

Nombres de neurones dans la couche cachée : 3

Les figures (1, 2,3) illustrent l'apprentissage qui consiste à chercher la configuration de poids qui minimise l'erreur ; ces figures représentent les graphes des fonctions suivantes :

- la série et son ajustement en utilisant les poids de la dernière itération.
- la somme des carrés des résidus de l'ajustement (SCR).
- la somme des carrés des résidus de la prévision (SCRp).

Deuxième itération :

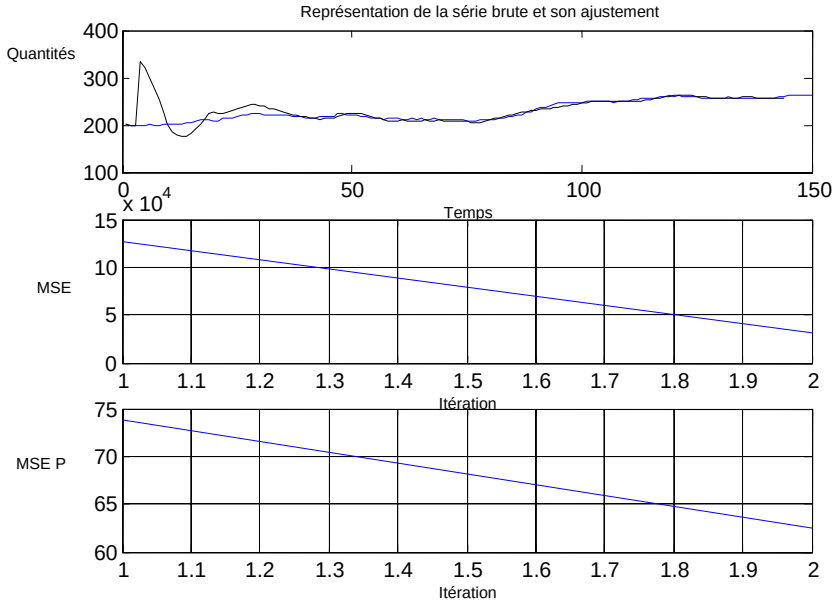


Figure 1

$$SCR = \frac{1}{2}\sum_{t=1}^{144} e_t^2 = 30398$$

$$SCR_p = \frac{1}{2}\sum_{t=145}^{150} e_t^2 = 62.4$$

Itération intermédiaire :

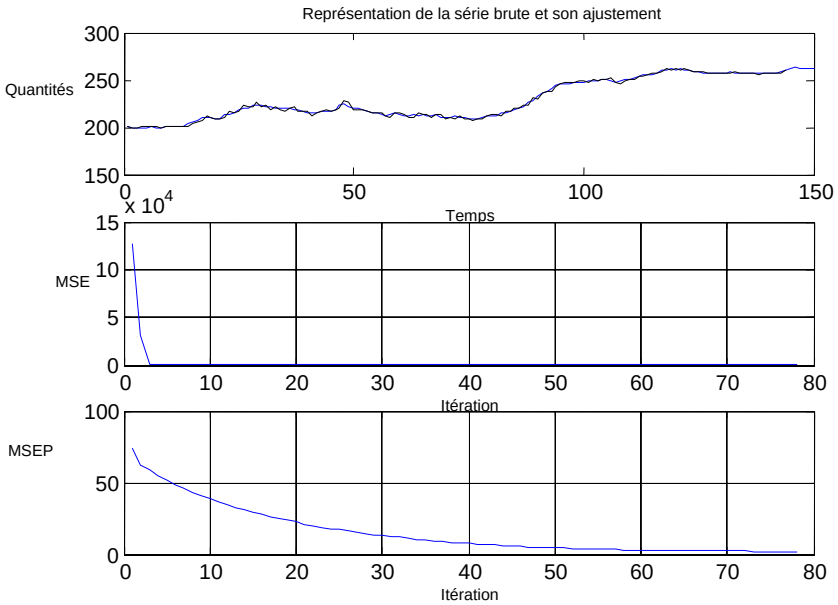


Figure 2

SCR= 203.31

SCRp= 1.993

Dernière itération :

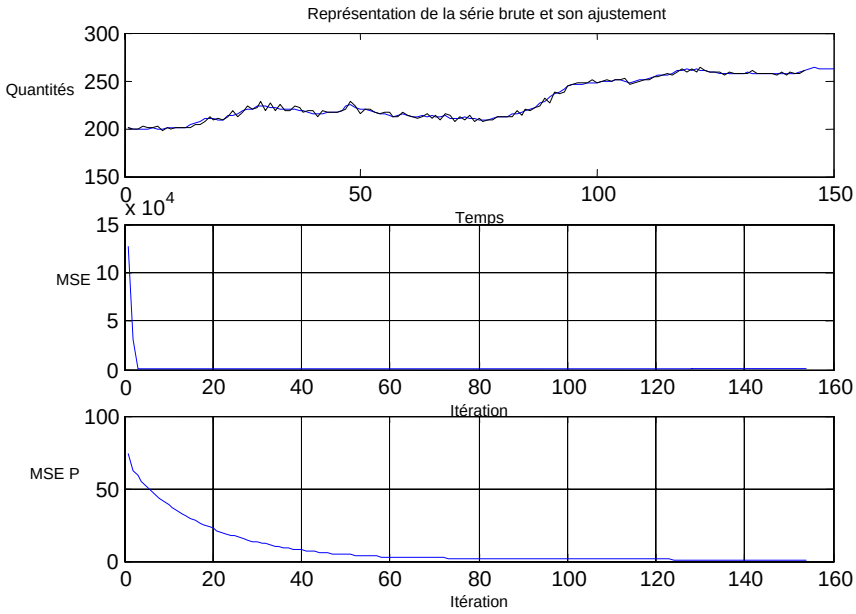


Figure 3

SCR= 325.03

SCRp= **0.17067**

XXXV.1.2 Test d'arrêt :

Théoriquement : Sur la base d'apprentissage, l'erreur diminue constamment, alors que sur la base de test elle passe par un minimum. Si l'apprentissage se prolonge au delà, les performances en test diminuent.

Donc, d'après les résultats issues de l'application de la nouvelle technique du principe du mode d'apprentissage supervisé, qui sont affichés dans le tableau ci-dessous on remarque que :

L'erreur quadratique de l'ajustement (SCR) augmente sans cesse au cours de l'apprentissage contrairement a ce qui a été dit dans la partie théorique concernant le mode d'apprentissage supervisé, par contre l'erreur quadratique de la prévision (SCR_P) passe par un minimum a **l'itération 154**, ce qui est conforme à ce qui a été écrit dans la partie théorique, Chose qu'on a pas pu le voir graphiquement, ce qui peut être expliqué par :

- taille de l'échantillon (faible).
- la série chronologique est régulière.

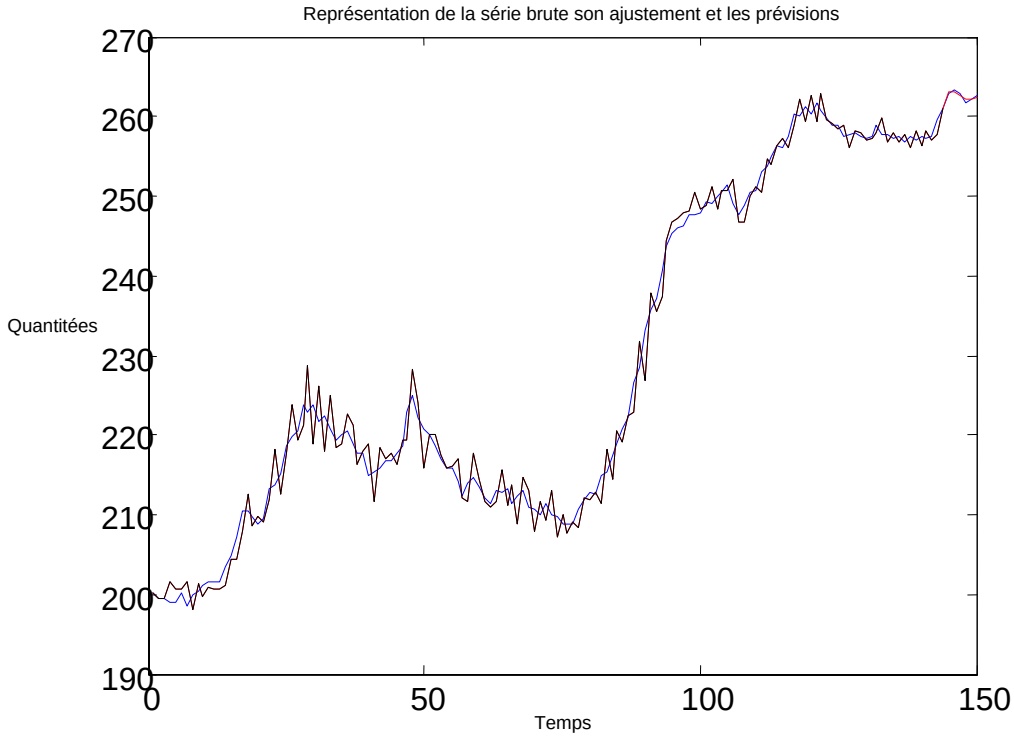
SCRP	SCR	ITERATION
0.1993	316.44	149
0.18863	318.18	150
0.18044	319.91	151
0.17473	321.63	152
0.17149	323.34	153
0.17067	**325.03**	**154**
0.17223	326.72	155
0.17612	328.39	156
0.18224	330.05	157
0.19051	331.7	158
0.20082	333.33	159

XXXV.1.3 Prévision :

BRUT	PREVI	ERRE
262.9	263.1166	-0,2166
263.3	263.1778	0,1222
262.8	262.5232	0,2768
261.8	262.1807	-0,3807
262.2	262.2607	-0,0607
262.7	262.4671	0,2329

D'après le tableau ci-dessus, on remarque que les écarts entres les prévisions obtenues par le réseau de neurones et les valeurs de série Brut sont très faibles, les valeurs sont presque confondues, d'ailleurs on peut le constater graphiquement (graphe ci-dessous).

On constat aussi d'après le graphe ci-dessous que l'écart entre la série d'ajustement et la série brut est très faible et les deux graphes sont quasi confondues.



Graphe de la série avec son ajustement et la prévision.

XXXV.1.4 Etude de la complexité de cet algorithme :

XXXV.1.4.1 Calcul de la complexité de la fonction prévision :

Cette fonction est constituée d'une boucle « For » à son intérieur une autre boucle « For » imbriquée, et des opérations indépendantes des boucles.

XXXV.1.4.2 Complexité de la boucle For :

Cette dernière contient à l'intérieur une boucle imbriquée, et d'autres opérations indépendantes de la boucle imbriquée, et la formule pour calculer sa complexité est :

$$T_{Al}(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)+113\right)$$ Ou :

T_1 : Est la complexité de la boucle « For » imbriquée.

T_c : Nombre d'affectation d'indice.

113 : Nombre d'opérations dans la boucle « For » indépendamment de la boucle imbriquée.

Donc : la complexité de la boucle imbriquée est :

$$T_1(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)\right)=\sum_{i=1}^{n}(1+23)=\sum_{i=1}^{n}24=24\text{n}.$$

La complexité de la boucle « For » est :

$$T_{A1}(n)=\sum_{i=1}^{n}\left(T_C(i,n)+T_1(i,n)+113\right)=T_{A1}(n)=\sum_{i=1}^{n}\left(1+24n+113\right)=\sum_{i=1}^{n}1+24n\sum_{i=1}^{n}1+113\sum_{i=1}^{n}1$$

$$T_{A1}(n)=24n^2+114n$$

XXXV.1.4.3 La complexité totale de la fonction prévision :

On a : 33 est le nombre d'opérations en dehors de la boucle « For » donc la complexité total de la fonction prévision est :

Op_p (n)$=33+T_{A1}(n)=33+114n+24n^2$

XXXV.1.4.4 Calcul de la complexité de la fonction Treillis :

Cette fonction est constituée de plusieurs opérations indépendamment des boucles, une boucle « For » et d'une grande boucle « For » qui contiennent 3 autres boucles « For » imbriquées comme suite :

La boucle n°3 est imbriquée dans la boucle n°2 qui est imbriquée dans la boucle n°1 et les trois boucles sont imbriquées dans la grande boucle « For ».

Soit :

Op_1 (n) le nombre des opérations dans la grande boucle « For ».

T_1 (i, n) le nombre des opérations dans la boucle « For » n°1.

T_2 (i, n) le nombre des opérations dans la boucle « For » n°2.

T_3 (i, n) le nombre des opérations dans la boucle « For » n°3.

XXXV.1.4.4.1Calcul du nombre d'opérations dans la grande boucle « For » :

$$T_3(i,n)=\sum_{i=1}^{n}6=6n$$

$$T_2(i,n)=\sum_{i=1}^{n}((\sum_{i=1}^{n}6)+1+1)=\sum_{i=1}^{n}(6n+2)=n(6n+2)=2n+6n^2$$

$$T_1(i,n)=\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}6)+1+1))+1)=\sum_{i=1}^{n}(2n+6n^2+1)=n+2n^2+6n^3$$

Donc le nombre d'opérations dans la grande boucle « For » est :

$$Op_1(n)=\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}((\sum_{i=1}^{n}6)+1+1))+1))+1)=\sum_{i=1}^{n}(n+2n^2+6n^3+1)=n+n^2+2n^3+6n^4$$

$$Op_1(n)=n+n^2+2n^3+6n^4$$

XXXV.1.4.4.2Calcul d'opérations en dehors de la grande boucle « For » :

Soit Op_2 (n) le nombre des opérations dans la fonction Treillis en dehors de la grande boucle « For ».

T (n)= $\sum_{i=1}^{n}(T_C(i,n)+T(i,n))$ est le nombre d'opérations dans la boucle « For » qui est indépendante de la grande boucle « For ».

6 : est le nombre d'opérations indépendamment de toutes les boucle qui existent dans la fonction treillis.

Donc :

Op_2 (n)= 6+T (n)=6+ $\sum_{i=1}^{n}2$ =6+ $2n$

Donc le nombre d'opérations total dans la fonction treillis est :

Op_t (n)= Op_1 (n) + Op_2 (n)= $n+n^2+2n^3+6n^4+6+2n=6+3n+n^2+2n^3+6n^4$

XXXV.1.4.5 Calcul de la complexité de la fonction RBF :

XXXV.1.4.5.1Calcul du nombre d'opérations dans la boucle While :

Soit T_A (n) le nombre d'opérations dans la boucle While tel que :

$T_A(n)=(\sum_{i=1}^{n}(T_C(i,n)+T_{A1}(i,n)+T_{A2}(i,n)+T_{A3}(i,n)+308+op_p(n)+1))+T_C(n+1,n)$ Tel que :

T_c : Nombre de test de la condition C.

T_{A1} : Nombre d'opérations dans la grande boucle For, cette dernière contient a l'intérieur deux boucles « For » imbriquées (T_1 et T_2).

T_{A2} : Nombre d'opérations dans la condition « if ».

T_{A3} : Nombre d'opérations dans la condition « if ».

298 : Nombre d'opérations dans la boucle While indépendamment des boucles imbriquées.

Op_p (n) : Nombre d'opérations dans la fonction prévision.

1 : représente l'appel de la fonction prévision car l'appel est compté comme une opération.

On a :

$T_C(i,n)=1$.

XXXV.1.4.5.2Calcul du nombre d'opération dans la grande boucle « For » :

Dans cet algorithme la grande boucle « For » est constituée d'une action conditionnelle de la forme : if (condition) action1 else action2 end if.

Donc d'après les formules illustrées ci-dessus le nombre d'opérations dans la grande boucle « For » est :

$T_{A1}(n)= \sum_{i=1}^{n}(T_C (n) + Ts)= \sum_{i=1}^{n}(T_C (n) + (1+ Max (Ta_1 (n), Ta_2 (n))))$ tel que :

Ta_1 : Est le nombre d'opérations dans l'action1 (a1).

Ta_2 : Est le nombre d'opérations dans l'action2 (a2).

T_c : Nombre d'affectation d'indice de la grande boucle « For ».

Ts : Est le nombre d'opérations dans la structure if (condition) action1 else action2 end if.

Soit :

$Ts=1+ Max (Ta_1 (n), Ta_2 (n))$

$Ta_1(n)= T_1 (n) +T_2 (n) +T_3 (n)$ tel que :

T_1 : est le nombre d'opération dans la première boucle de l'action 1.

T_2 : est le nombre d'opération dans la deuxième boucle de l'action 1.

T_3 : est le nombre d'opération en dehors des deux boucles «For » de l'action 1.

Donc :

$Ta_1(n) = (\sum_{i=1}^{n}(T_{C1}(i,n)+T_1(i,n)))+(\sum_{i=1}^{n}(T_{C2}(i,n)+T_2(i,n)))+T_3(i,n)$ Tel que :

$Ta_1=113+ \sum_{i=1}^{n}24 + \sum_{i=1}^{n}65 =113+24n+65n=113+89n$.

De même pour la deuxième action :

$Ta_2(n) = (\sum_{i=1}^{n}(T_{C1}(i,n)+T_1(i,n)))+(\sum_{i=1}^{n}(T_{C2}(i,n)+T_2(i,n)))+T_3(i,n)$

$Ta_2=113+ \sum_{i=1}^{n}24 + \sum_{i=1}^{n}68 =113+24n+68n=113+92n$.

Donc $Ts=1+Max ((113+89n), (113+92n))=1+113+92n=114+92n$

$Ts=114+92n$

Le nombre d'opérations dans la grande boucle « For » est :

$T_{A1}(n)= \sum_{i=1}^{n}(T_C (n) + (1 Max (Ta_1 (n), Ta_2 (n))))= \sum_{i=1}^{n}(1+114+92n)$

$T_{A1}(n)= \sum_{i=1}^{n}115 + \sum_{i=1}^{n}92n =115n+92n^2$

En dehors de la grande boucle « For » on trouve deux conditions « if » :

XXXV.1.4.5.3Le nombre d'opérations dans la 1ere condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

$T_{A2}(n) = T_C(n) + \text{Max}(T_{a1}(n), T_{a2}(n))$.

Dans le cas de notre algorithme l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'affectations, donc notre équation devienne :

$T_{A2}(n) = T_C(n) + T_{a1}(n)$.

$T_{A2}(n) = 1+7=8$

XXXV.1.4.5.4Le nombre d'opérations dans la 2eme condition « if »:

D'après la règle du calcul de la complexité du conditionnelle on a :

$T_{A3}(n) = T_C(n) + \text{Max}(T_{a1}(n), T_{a2}(n))$.

De même pour la deuxième boucle « if » l'action a_2 n'existe pas et l'action a_1 est constituée uniquement d'instructions, donc notre équation devienne :

$T_{A3}(n) = T_C(n) + T_{a1}(n)$.

$T_{A3}(n) = 1 +12=13$.

Donc le nombre total d'opérations dans la boucle While est :

$$T_A(n) = \left(\sum_{i=1}^{n} \left(T_C(i,n) + T_{A1}(i,n) + T_{A2}(i,n) + T_{A3}(i,n) + 298 + op_p(n) + 1 \right) \right) + T_C(n+1,n) =$$

$$\left(\sum_{i=1}^{n} (1+115n + 92n^2 +8+13+298+33 + 114n + 24n^2 +1)) +1 = \left(\sum_{i=1}^{n} (354+229n+116n^2 \right)) +1$$

$$T_A(n) = 1 + 354n + 229n^2 + 116n^3$$

Le nombre total d'opération **Op(n)** de la fonction principale RBF qui est aussi le nombre total d'opérations effectuées dans l'exécution de l'algorithme est:

$Op(n) = (Op_t(n)+1)+T_A(n)+189$ tel que :

$Op_t(n)$: est le nombre d'opérations dans la fonction Treillis

$T_A(n)$: est le nombre d'opérations dans la boucle While de la fonction RBF.

198 : est le nombre d'opérations dans la fonction RBF indépendamment des boucle et des autres fonctions.

1 : est l'opération d'appel de la fonction Treillis dans la fonction RBF car la fonction treillis est appelé qu'une seule fois dans l'exécution de l'algorithme.

Donc :

Op(n)=$1+ 6 + 3n + n^2 + 2n^3 + 6n^4 + 1 + 354n + 229n^2 + 116n^3 + 198$

Op(n)=$206 + 357n + 230n^2 + 118n^3 + 6n^4$

On a :

$$\lim_{n\to\infty}(206+357n+230n^2+118n^3+6n^4)/n^4=6\neq 0\Rightarrow Op(n) \quad est\,en \quad \Theta(n^4)$$

XXXV.1.5 Conclusion :

D'après les résultats ci-dessus qui ont été obtenus, l'algorithme qui a été utilisé dans cette application est d'une complexité polynomiale d'ordre 4, c'est-à-dire que notre algorithme est en $O(n^4)$.

XXXVI Comparaison des résultats :

La comparaison des 4 techniques utilisées ci-dessus selon la nouvelle idée du mode d'apprentissage supervisé revient aussi a comparé la somme des carrés des résidus de la prévision (SCRp) car ces techniques ont été employées pour calculer la prévision, et Le tableau ci-dessous résume les résultats trouvés :

	L'APPLICATION DE LA NOUVELLE APPROCHE DU MODE D'APPRENTISSAGE SUPERVISE			
	l'état des paramètres de la fonction noyau	**les règles utilisées pour la modification des poids**	**La complexité**	**SCR_p**
1 ère méthode	Les paramètres (Ci, σ_i) sont fixe une fois pour toute durant tout le processus d'apprentissage par la méthode des treillis.	La règle de Widdrow-Hoff	$O(n^4)$	**0.70635**
		La règle de rétro propagation du gradient	$O(n^4)$	**0.11697**
2 eme méthode	Les paramètres (Ci, σ_i) sont variable durant tout le processus d'apprentissage par la méthode descente de gradient.	La règle de Widdrow-Hoff	$O(n^5)$	**0.55942**
		La règle de rétro propagation du gradient	$O(n^4)$	**0.17067**

D'après les résultats affichés dans le tableau ci-dessus on constat que les résultats qui ont étés obtenus avec la 2eme méthode c'est à dire : les paramètres (Ci, σ_i) de la fonction sont variable durant tout le processus d'apprentissage par la méthode descente de gradient sont meilleur que les résultats qui ont été obtenus avec la 1ere méthode c'est à dire : les paramètres (Ci, σ_i) sont fixe une fois pour toute durant tout le processus d'apprentissage par la méthode des treillis. .

On remarque aussi que les résultats qui ont étés obtenus la ou la règle de modification des poids est la règle de rétro propagation du gradient sont meilleur que les résultats obtenus la ou la règle de modification des poids est la règle de Widdrow-Hoff.

De même pour la complexité des algorithmes, on remarque que l'algorithme qui contient la règle de rétro propagation du gradient et la règle de Widdrow-Hoff pour la modification des poids dans la première méthode ont la même complexité polynomiale.

Par contre dans la deuxième méthode on constat que la complexité de l'algorithme qui contient la règle de rétro propagation du gradient pour la modification des poids est meilleurs, c'est-à-dire polynomiale de degré inférieur a l'algorithme qui contient la règle de Widdrow-Hoff pour la modification des poids.

XXXVII Conclusion générale:

D'apes les deux tableaux ci-dessus, et qui illustrent les résultats issus du principe du mode d'apprentissage existant d'une part, et d'autre part les résultats issus de l'application de la nouvelle approche sur le mode d'apprentissage supervisé on remarque que :

Les résultats obtenus on appliquant la nouvelle approche d'apprentissage supervisé sont meilleurs que les résultats obtenus par le principe du mode d'apprentissage existant.

Concernant la complexité on remarque que l'application de la nouvelle approche sur le mode d'apprentissage supervisé a fait augmenté d'un degré la complexité des algorithmes la ou la règle de rétro propagation du gradient est utilisé pour la modification des poids par rapport au principe du mode d'apprentissage supervisé existant.

Pour les algorithmes qui utilisent la règle de Widdrow-Hoff pour la modification des poids on constat que la complexité a resté la même pour les algorithmes qui ont les centres et les rayons de la fonction noyau variables, par contre dans les algorithmes ou les centres et les rayons de la fonction noyau sont fixes, l'application de la nouvelle approche sur l'apprentissage supervisé a fait diminué la complexité des algorithmes d'un degré.

Bibliographie

Bibliographie :

[1] Statistique et méthodes neuronales. S-THIRIA, Y-LECHEVALLIER, O-GUSCUEL, S-CANU : DUNOD 1997.

[2] Les réseaux de neurones : principes et définitions. Jean-François JODOUIN : Hermes 1994.

[3] Techniques avancées pour le traitement de l'information : réseaux de neurones, logique floue, algorithmes génétiques. Jean-Louis AMAT, Gérard YAHIAOUI : Cépadués- éditions 1996.

[4] Réseaux neuronaux et traitement du signal. J-HERAUlT, C-JUTTEN: Hermes 1994.

[5] Algorithmes génétiques et réseaux de neurones. Jean-Michel RENDERS, Hermes 1995.

[6] Matlab 6.1

[7] Algorithmique et langage C : Chapitre 1 : Complexité des algorithmes « Tarik FDIL, membre fondateur d'OSIM (IT Maroc) »

[8] Algorithmique et Complexité : UE de S5-Licence d'Informatique « Christine FROIDEVAUX ; Université de Paris Sud».

[9] Cours d'algorithmique « http://web.ifrance.com/ »

[10] Cours VARI : « **V**aleur d'**A**ccueil et de **R**econversion à l'**I**nformatique » du professeur :

LOENENGUTH Pascal à CAMOS-CNAM qui est un organisme d'enseignement supérieur et de promotion supérieure du travail associé au CNAM.
Il a été créé en 1959. Il est géré par l'Association Mosellane d'Enseignement Scientifique, Technique et Economique (AMESTE) dont le siège est à la préfecture de la Moselle. Le ministère de tutelle est le Ministère de l'Education Nationale, de le Recherche et de la Technologie.
Le CAMOS-CNAM dispense des enseignements scientifiques et techniques, d'informatique, d'économie et gestion et de management des entreprises.

[11] Cours 'Deug-Mias-2' : Complexité des algorithmes. « Institut de Galilée de l'université de Paris 13, LIPN : **L**aboratoire d'**I**nformatique de **P**aris **N**ord ».

[12] Cours UE : Algorithmes et programmation L1/S2 « Département d'informatique, Université de Bretagne Occidentale ».

[13] Magasine 'the Remover's Magazine' : Article : Programmation « la complexité des algorithmes ».

[14] Cours introduction à la programmation : Chapitre 7 : Introduction à la complexité des algorithmes « François Morain : Professeur Informatique a l'école polytechnique France ».

[15] Cours Master Informatique –Année 1- Semestre 1 et 2 « Chapitre 8 : Temps de calcul, classes de complexité P et NP » du prof. Jean Bétérma 'Labri Université Bordeaux 1'.

[16] Théorie de la complexité « un article de Wikipédia » Références :

- Patrick Blackburn, Maarten de Rijke et Yde Venema, Modal Logic, 2001 [détail des éditions], annexe C « A Computational Toolkit », pp. 504-515.
- Michael R. Garey, David S. Johnson, Computers and Intractability: A guide to the theory of NP-completeness, W.H. Freeman & Company, 1979. ISBN 0716710455.
- Pierre Wolper, Introduction à la calculabilité, Dunod, 2001. ISBN 2100048538.
- Richard Lassaigne, Michel de Rougemont, Logique et Complexité, Hermes, 1996. ISBN 2866014960.
- Christos Papadimitriou, *Computational Complexity*, Addison-Wesley, 1993. ISBN 0201530821.

[17] Algorithmique Avancée et Complexité 'Master 1 d'Informatique' S.Tison. Université des Sciences et Technologies de Lille. À la complexité des algorithmes – Francois Morain – Professeur en informatique.

[18] www.aiaccess.net « Glossaire de Statistique et Modélisation de données ».

[19] Introduction au connexionnisme « Cours, exercices et travaux Pratiques » 'Claude TOUZET Juillet 1992'.

[20] Principes et applications des Réseaux de neurones « Denis THUILLER » Professeur à L'école de Sciences de la gestion de l'université du Québec à Montréal.

[21] Les réseaux de neurones «http://psylon.free.fr/formatio/doc/rn.doc »

[22] Cours : les réseaux de neurones « Rafic YOUNES, enseignant chercheur a la Faculté de Génie Electrique/ l'Université Libanaise ».

[23] Cours : Chapitre 4 : Processus d'apprentissage « Département génie électrique et génie Informatique »' Université LAVAL Canada'.

[24] Rapport de stage (première année Master Physique) : Le développement d'un réseau de neurones pour l'analyse en forme de signal dans l'expérience DVCS par : Azza CHETTAOUI, Responsable de stage : Carlos MUNOZ CAMACHO.

[25] Publication : Modélisation, classification et commande par réseaux de neurones : Principes fondamentaux, méthodologie de conception et illustrations industrielles. « I. RIVALS, L.

PERSONNAZ, G. DREYFUS » Laboratoire d'Electronique. 'Ecole Supérieure de physique et de chimie Industrielles de la ville de Paris'

[26] Réseaux de neurones « GIF-21140 et GIF-64326 » par MARC Parizeau 'Automne 2004' université de LAVAL.