UNIVERSITE D'ANTANANARIVO

FACULTE DES SCIENCES

Domaine : Sciences et Technologies

Mention : Physique et Applications

Parcours : Physique des Hautes Énergies

MEMOIRE

Pour l'obtention du diplôme de

MASTER EN PHYSIQUE ET APPLICATIONS

Intitulé :

Création d'un perceptron multicouche et application à la prédiction des séries temporelles générées par l'application logistique

présenté par :

ANDRY MANANTENA Kiady Mahefa

devant la commission d'examen composée par :

Président : M. RANAIVO-NOMENJANAHARY Flavien	*Professeur Titulaire*
Rapporteur : M. RABOANARY Roland	*Professeur Titulaire*
Examinateurs : M. HANITRIARIVO Rakotoson	*Maître de Conférences*

le 26 juillet 2016

UNIVERSITE D'ANTANANARIVO

FACULTE DES SCIENCES

Domaine : Sciences et Technologies

Mention : Physique et Applications

Parcours : Physique des Hautes Énergies

MEMOIRE

Pour l'obtention du diplôme de

MASTER EN PHYSIQUE ET APPLICATIONS

Intitulé :

Création d'un perceptron multicouche et application à la prédiction des séries temporelles générées par l'application logistique

présenté par :

ANDRY MANANTENA Kiady Mahefa

devant la commission d'examen composée par :

Président : M. RANAIVO-NOMENJANAHARY Flavien	*Professeur Titulaire*
Rapporteur : M. RABOANARY Roland	*Professeur Titulaire*
Examinateurs : M. HANITRIARIVO Rakotoson	*Maître de Conférences*

Remerciements

Je tiens à exprimer mes chaleureux remerciements à Monsieur RABOANARY Roland, *Professeur Titulaire* à l'Université d'Antananarivo et Responsable de l'Équipe d'Accueil Doctoral Physique Nucléaire et Physique des Hautes Énergies, de m'avoir proposé ce présent thème de recherche et par la suite de bien vouloir m'encadrer du début jusqu' à la rédaction et à cette soutenance de ce mémoire. Je lui dois toute ma gratitude pour sa patience, sa disponibilité et surtout ses judicieux conseils.

Je tiens aussi à remercier Monsieur RANAIVO-NOMENJANAHARY Flavien, *Professeur Titulaire* à l'Université d'Antananarivo et Responsable de l'Équipe d'Accueil Doctoral Physique du Globe, de l'Energie et de l'Environnement à la Faculté des Sciences d'accepter et de bien vouloir présider la commission du Jury de cette soutenance.

Un grand remerciement est aussi adressé à Monsieur HANITRIARIVO Rakotoson, *Maître de Conférences* à la Facultés des Sciences et Responsable du parcours Physiques des Hautes Énergies pour ses enseignements durant les deux derniers années et d'avoir accepté d'examiner ce travail.

Sans oublier ma famille, mes ami(e)s qui m'ont soutenu tout le long de mes études et m'ont aidé à la réalisation du présent travail et à toutes personnes qui ont contribué de près ou de loin à l'élaboration de ce travail. Un grand MERCI à vous !

Misaotra indrindra tompoko !

Table des matières

Liste des tableaux

Table des figures

Liste des Abréviations

API Application Programming Interface. 12

IA Intelligence Artificielle. 1

JVM Java Virtual Machine ou Machine Virtuelle Java. 12

MSE Mean Squared Error ou Erreur quadratique. 16, 31

NMSE Normalised Mean Square Error ou Erreur quadratique normalisée. 16, 31

PMC Perceptron Multicouche. iii, 1, 12, 29

RNA Réseau de Neurones Artificiels. 1, 28, 29, 31

UML Unified Modeling Language. v, 13–15, 18

Introduction

Depuis quelques années, l'apparition de l'intelligence artificielle (IA) dans la branche de l'informatique, a changé la vie de l'humanité. Elle a pour but de développer des machines capables d'avoir un comportement intelligent. Plusieurs sont les modèles de systèmes intelligents, mais celui qui nous intéresse est le modèle connexionniste : les réseaux de neurones artificiels (RNA). Imités des fonctionnements des neurones biologiques, les RNA ont la capacité d'apprendre, de généraliser ou d'organiser des données. Déjà, les RNA sont largement exploités dans les domaines de classification et de reconnaissance des formes ou des paroles mais dans ce travail nous les appliquons à la prédiction des séries temporelles chaotiques.

Nous savons que l'histoire des progrès scientifiques est souvent assimilée comme l'histoire des prédictions réussies : Thalès a prédit une éclipse, Newton le retour de la comète de Halley et Einstein la courbure des rayons lumineux par le Soleil. D'une façon générale en physique expérimentale, on essaie d'établir des lois mathématiques qui permettent de prédire le comportement d'un objet : les lois de Newton permettent de prédire la trajectoire d'un objet en connaissant les forces qui s'y appliquent, sa position et sa vitesse initiale. Non seulement dans ce domaine, une prédiction réussie nous avantage aussi dans de nombreux domaines comme la finance ou bien la météorologie. Mais l'apparition du chaos a bouleversé ces réussites. Le chaos est un comportement de certains systèmes naturels ou artificiels qui est classé comme imprévisible et complexe. Mais existe-il un modèle de prédiction capable de résoudre ce problème ? C'est pour répondre à cette question qu'est le but de ce présent travail.

Les séries temporelles sont des mesures d'une variable que l'on prend à un intervalle de temps régulier comme l'indice boursier ou la température journalière ou bien l'évolution de la population au cours du temps ou tout simplement l'évolution d'une variable d'un système dynamique. Dans ce travail, nous prendrons le cas d'un système dynamique tel que l'application logistique. La prédiction des séries temporelles consiste à estimer les valeurs futures de la série en se basant sur les valeurs passées et présents.

Notre travail se divise en deux grandes parties. La première partie est consacrée à l'étude des réseaux de neurones artificiels, ou plus précisément des réseaux à propagation de l'information vers l'avant et à la création de ces derniers. Dans le chapitre 1, nous voyons les théories de bases des réseaux de neurones tels que les perceptrons multicouches (PMC) et l'algorithme de rétro-propagation. Le chapitre 2 est destiné au développement de ce dernier en utilisant le langage de programmation java. Ensuite dans la seconde partie de notre travail, nous voyons dans le chapitre 3 les caractéristiques et les particularités de notre application logistique. Nous parlons ensuite des séries temporelles et des méthodes de prédictions par les réseaux de neurones dans le chapitre 4. Et pour terminer, un dernier chapitre a été créé pour représenter les résultats ainsi trouvés.

Première partie

Réseaux de neurones Théorie et création

Chapitre 1

Les réseaux de neurones

1.1 Historique

Depuis 1943, Warren McCulloch et Walter Pitts ont introduit les modèles de réseaux de neurone [1]. Ils créèrent un modèle simplifié de neurones biologiques communément appelé neurone formel. Ils montrèrent que des réseaux de neurones formels simples peuvent théoriquement réaliser des fonctions logiques, arithmétiques et symboliques complexes. Ensuite, en 1947, ils ont indiqué un champ d'application pratique nommé la reconnaissance des formes spatiales par les réseaux de neurones[2]. Comme les travaux de McCulloch et Pitts n'ont pas donné d'indication sur une méthode pour adapter les coefficients synaptiques, cette question sur l'apprentissage a connu un début de réponse grâce aux travaux de Donald O. HEBB, un physiologiste canadien. Il a proposé une règle classique appelé règle de Hebb [3] qui permet de modifier la valeur des coefficients synaptiques. Hebb pouvait postuler cette règle mais à cause de l'absence de recherche neurologique, il n'était pas capable de la vérifier.

En 1951, dans sa dissertation Marvin Minsky a développé le *Snark*, qui avait déjà été capable d'adjuster automatiquement ses poids. Mais ceci n'a jamais été pratiquement implanté. Ensuite, en 1956, des scientifiques bien connus et des étudiants ambitieux se rencontrent dans le Dart mouth Summer Research Project et discutent de la façon de simuler le comportement du cerveau.

De 1957 à 1958, Frank Rosenblatt, Charles Wightman, et leurs collègues ont développé le premier succès du neurocomputer baptisé *Mark I perceptron*[4], capable de reconnaitre de simples chiffres . Ensuite, Rosenblatt a décrit en 1959, différentes versions de perceptron,à formulé et a vérifié le théorème de convergence du perceptron. En 1960, Bernard Widrow et Marcian E. Hoff ont introduit l'ADALINE *ADAptive LInear NEuron* [5], un rapide et précis système d'apprentissage qui utilise les réseaux de neurones. L'ADALINE a été ensuite renommé en *adaptive linear element*. Mais un long silence est apparu et a duré environ 20 ans.

Malgré ce silence, les recherches continuèrent lentement. En 1969 Marvin Minsky et S Papert ont publié un précis mathématique du perceptron en démontrant qu'il ne peut pas résoudre tout problème tel que le problème de l'XOR[6]. En 1972, Teuvo Kohonen a introduit le modèle *linear associator*[8], un modèle d'une mémoire associative. Dans la même année, un même modèle a été indépendamment présenté mais d'après le point de vue d'un neurophysiologiste James A. Anderson. En 1974, dans sa dissertation au Harvard,

Paul Wervbos a développé une procédure d'apprentissage appelé *rétro-propagation des erreurs*[7]. Ensuite entre 1976 et 1980 Stephen Grossberg a développé plusieurs articles dans lesquels de nombreux modèles de neurones ont été analysés mathématiquement[9]. Sous la coopération de Gail Carpenter ceci tend vers les modèles appelés *adaptive resonance theory* (ART).

En 1982, un nouveau souffle est apparu avec les travaux de T.Kohonen qui a décrit ce qu'on appelle *self-organizing feature maps* (SOM)[10, 11], encore connu sous le nom de *Carte de Kohonen* et de Joseph Hopfield, un physicien reconnu, en publiant un nouveau modèle de réseau de neurones ou modèle d'Hopfield[12].

Jusqu'à maintenant, le développement de ce champ de recherche a été toujours explosif, les réseaux de neurones prennent puissance avec l'apparition des nouvelles générations de processeurs.

1.2 Les neurones biologiques

Les réseaux de neurones artificiels ont fortement imité les fonctionnements des neurones biologiques[13, 14]. Dans un cerveau humain on compte environ mille milliards de neurones, avec 10^3 à 10^4 connexions. Un neurone est une cellule excitable capable de transmettre des informations à d'autres neurones. La transmission des informations se fait au niveau de la synapse en échangeant une substance chimique appelée *neurotransmetteur*.

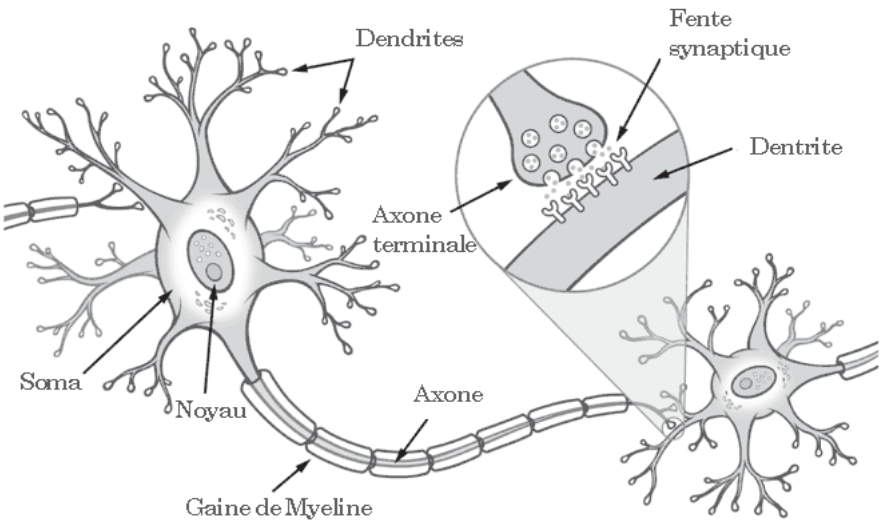


FIGURE 1.1 – neurone biologique

Et l'information traitée chemine le long de l'axone pour être transmise aux autres neurones.

1.3 Le neurone formel

Le neurone formel est la composante principale d'un réseau de neurones artificiels[13, 14]. Imité du neurone biologique, un neurone formel reçoit des variables d'entrées en provenance des autres neurones. A chacune des entrée est associé un poids w_{ij} représentatif de la force de connexion. Les poids peuvent être positifs ou négatifs.

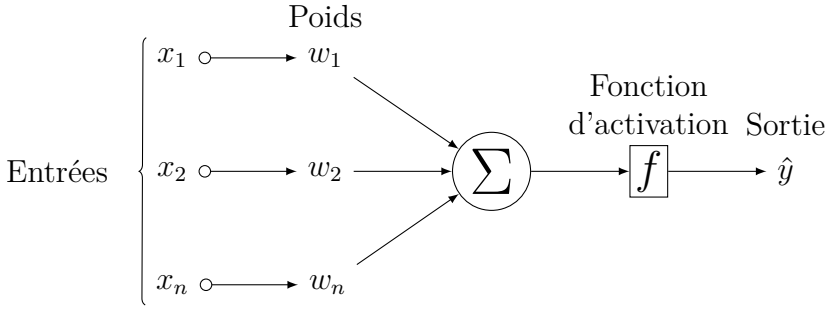


FIGURE 1.2 – Neurone formel

Les informations ainsi recueillies sont traitées par une fonction dite d'activation ou de transfert. Cette fonction joue le rôle de corps cellulaire.

Neurone biologique	Neurone formel
Dendrite	Signal d'entrée
Synapses	Poids
Soma	Fonction d'activation
Axones	Sortie

TABLE 1.1 – Analogie neurone biologique et neurone formel

La valeur sortie $\hat{y}$ est donnée par

$$\hat{y} = f(\sum_{i=1}^{n} x_i w_i) \tag{1.1}$$

Un autre modèle consiste à ajouter une autre entrée $x_0 = +1$[13]. Le poids synaptique associé est noté $b = w_0$ et joue le rôle de seuil pour le neurone. b est aussi appelé biais.

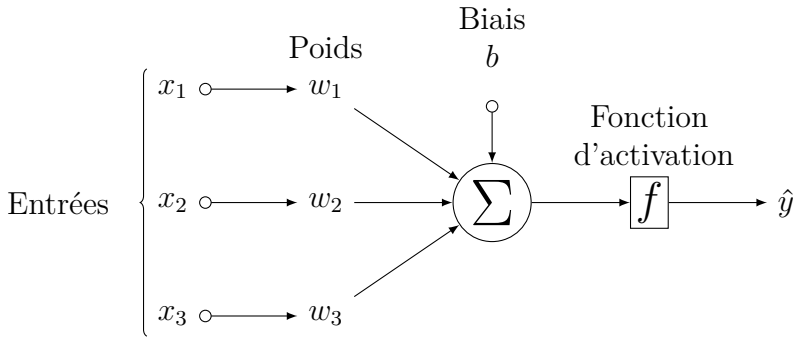


FIGURE 1.3 – Neurone formel avec biais(Perceptron)

$$\hat{y} = f(\sum_{i=1}^{n} x_i w_i + b) = f(\sum_{i=0}^{n} x_i w_i) \tag{1.2}$$

1.3.1 Les fonctions d'activation

Une fonction d'activation sert à convertir le résultat de la somme pondérée des entrées d'un neurone en une valeur de sortie. Cette conversion s'effectue par un calcul de l'état du neurone en introduisant une non-linéarité dans le fonctionnement du neurone [16].

Voyons quelques exemples

a. La fonction sigmoïde

La fonction sigmoïde est définie par

$$f(x) = \frac{1}{1 + \mathrm{e}^{-\alpha x}} \tag{1.3}$$

$\alpha > 0$ et $x \in \mathbb{R}$.

b. La fonction linéaire

$$f(x) = \alpha x \tag{1.4}$$

α un paramètre réel

c. La fonction tangente hyperbolique[15]

$$f(x) = \tanh(x) \tag{1.5}$$

d. La fonction escalier

$$f(x) = \begin{cases} 1 \ si \ x \geq 0 \\ 0 \ sinon \end{cases} \tag{1.6}$$

1.4 Les réseaux de neurones artificiels

Un réseau de neurone artificiel est une interconnexion de plusieurs neurones formels. L'interconnexion entre les neurones suit une architecture bien définie. Cette architecture décrit la topologie d'un réseau de neurones.

1.4.1 Topologie

Les réseaux de neurones se divisent en deux classes selon leur topologie :

— Les réseaux à propagation de l'information vers l'avant ou *feedforward network*.

— Les réseaux récurrents ou *feedback network*

a. Feedforward network

Dans un feedforward network les informations se propagent dans une seule direction de l'entrée vers la sortie. Le sens de propagation est indiqué par les flèches dans l'exemple suivant :

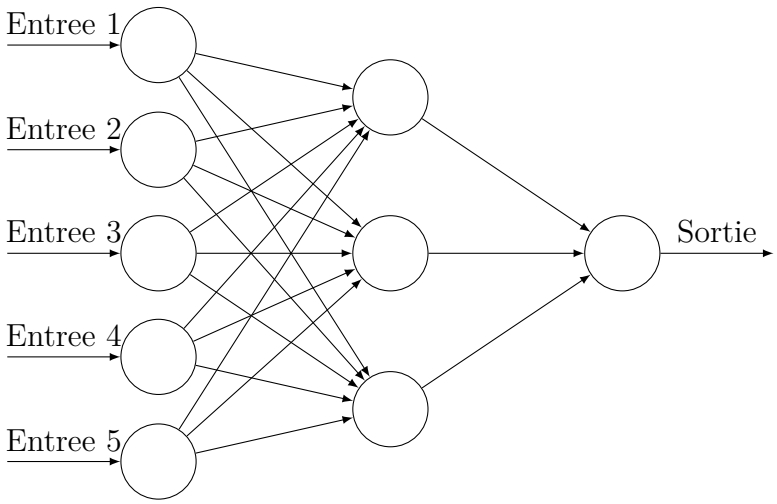


FIGURE 1.4 – Exemple de réseaux à propagation de l'information vers l'avant

b. Réseaux récurrents

Dans un réseau récurrent, la notion de sens de propagation n'existe plus, les informations se propagent dans les deux sens. On peut citer comme exemples les réseaux d'Elman [17], de Jordan [18] ou bien le réseau d' Hopfield [12] que montre la figure suivante

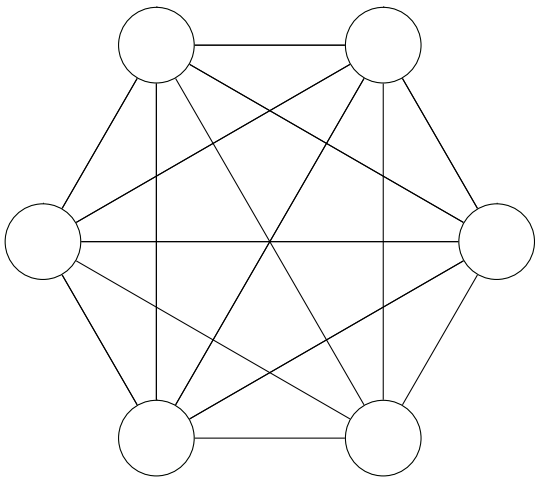


FIGURE 1.5 – Exemple d'architecture de réseaux récurrents(Réseau d'Hopfield)

1.5 Apprentissage des réseaux

La question la plus intéressante dans les réseaux de neurones artificiels est l'apprentissage. Il consiste à modifier les poids synaptiques du réseau jusqu'à l'obtention des résultats désirés. On distingue deux grandes classes d'algorithme d'apprentissage des réseaux de neurones : L'apprentissage supervisé et l'apprentissage non supervisée.

1.5.1 L'apprentissage supervisé

Dans un apprentissage supervisé, on connait en avance les résultats désirés. Si on pose $y_i^{(p)}$ les sorties désirées et $\hat{y}_i^{(p)}$ les sorties du réseau où p indique l'indice du prototype ou exemple. La phase d'apprentissage supervisée consiste à comparer la sortie estimée par le réseau avec la sortie désirée et mettre à jour les poids synaptiques jusqu'à ce que le réseau donne un résultat acceptable. La façon dont ces poids sont mis à jour définit la règle d'apprentissage du réseau.

1.5.1.1 Règle delta

La règle delta[20],aussi appelé règle du perceptron [19], est appliquée dans le cas où la fonction d'activation est linéaire. Elle utilise la méthode de descente de gradient. D'après l'équation (1.2), on a

$$\hat{y}_i^{(p)} = \sum_{i=1}^{n} x_i w_i + b \tag{1.7}$$

L'erreur entre les valeurs désirées et les valeurs estimées par le réseau est $\delta^{(p)} = \left(y_i^{(p)} - \hat{y}_i^{(p)}\right)$
On définit la fonction de coût ou fonction d'erreur par E^p

$$E^p = \frac{1}{2} \sum_i \left(y_i^{(p)} - \hat{y}_i^{(p)}\right)^2 \tag{1.8}$$

Cette fonction doit être continue et dérivable pour qu'on puisse appliquer la méthode de descente de gradient. L'apprentissage consiste alors à trouver toutes les valeurs des poids pour minimiser la fonction d'erreur. L'idée est d'ajuster les poids avec une quantité proportionnelle à $\Delta_p w_i = -\eta \frac{\partial E^p}{\partial w_i}$.
On a

$$\frac{\partial E^p}{\partial w_i} = \frac{\partial E^p}{\partial \hat{y}_i^p} \frac{\partial \hat{y}_i^p}{\partial w_i} \tag{1.9}$$

Or d'après (1.7)

$$\frac{\partial \hat{y}_i^p}{\partial w_i} = x_i \tag{1.10}$$

et (1.8) nous donne

$$\frac{\partial E^p}{\partial \hat{y}_i^p} = -\left(y_i^{(p)} - \hat{y}_i^{(p)}\right) \tag{1.11}$$

D'où

$$\Delta_p w_i = \eta \delta^p x_i \tag{1.12}$$

$\eta < 1$ appelé pas d'apprentissage.

1.5.2 L'apprentissage non supervisé

Contrairement à l'apprentissage supervisé seules les valeurs d'entrée sont disponibles. Dans ce cas , les prototypes présentés à l'entrée provoquent une auto-adaptation du réseau afin de produire des valeurs de sortie qui soient proches en réponse à des valeurs d'entrée similaires. Les apprentissages non-supervisés sont souvent utilisés dans les réseaux récurrentes. On peut citer quelques exemples d'apprentissage comme la règle du Hebb [3] ou le règle de Kohonen[10]. Dans ce travail, nous n'allons pas détailler ce type d'apprentissage mais nous pouvons citer quelques références pour plus de détails.[13]

1.6 Les réseaux multicouches à propagation de l'information vers l'avant

Les réseaux multicouches à propagation de l'information vers l'avant ou perceptron multicouche, comme leur nom l'indique sont organisés en couches. Chaque couche est composée de neurones et chaque neurone reçoit ses entrées des neurones de la couche amont et renvoie ses sorties aux neurones de la couche suivante. La première couche est appelée couche d'entrée et la dernière couche de sortie. Les couches entre ces deux couches sont appelées couches cachées. Souvent on utilise la fonction sigmoïde (1.3) ou la fonction hyper-tangente (1.5) comme fonction d'activation.

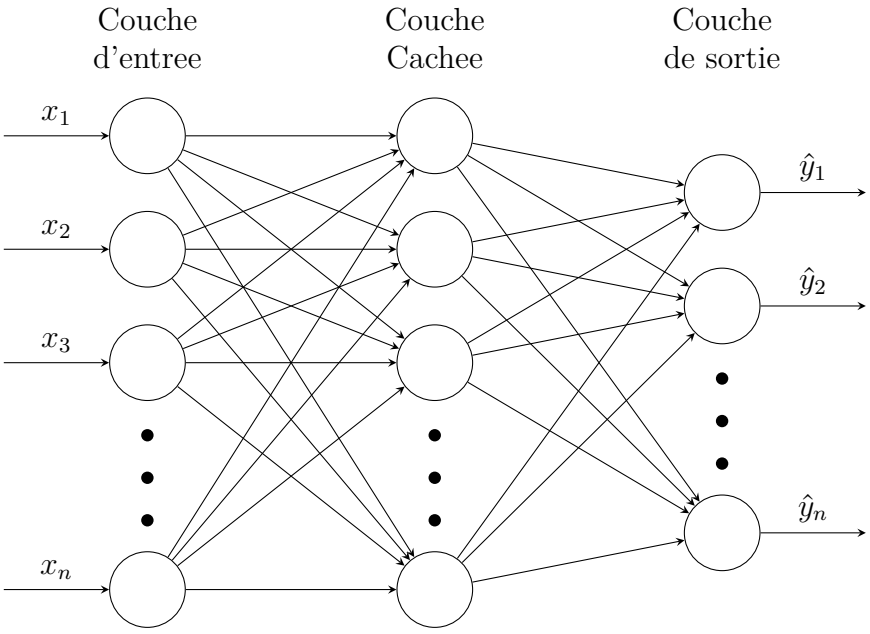


FIGURE 1.6 – Réseau de neurones multicouche à propagation de l'information vers l'avant avec une couche cachée

1.6.1 L'algorithme de rétro-propagation

En anglais *backpropagation*, la rétro-propagation [7] est une généralisation de la règle delta. Prenons une fonction d'activation f quelconque.

Connexion couche de sortie couche cachée

Posons j l'indice des neurones de la couche cachée, i et k, pour les neurones de la couche d'entrée et de sortie respectivement. On pose aussi $\hat{y}_j^p$, la sortie de la $j^{ème}$ neurone de la couche cachée et $\hat{y}_k^p$, la sortie du neurone du $k^{ème}$ neurone de la couche de sortie. p étant l'indice du prototype.
Nous avons

$$\hat{y}_k^{(p)} = f\left(\sum_{k=1} \hat{y}_j^p w_{kj} + b_k\right) = f(v_k^p) \tag{1.13}$$

avec

$$v_k^p = \sum_{k=1} \hat{y}_j^p w_{kj} + b_k \tag{1.14}$$

b_k biais pour la couche de sortie.
Nous définissons l'erreur entre la sortie du réseau et la sortie désirée par

$$e_k^p = (y_k^p - \hat{y}_k^p) \tag{1.15}$$

avec y_k^p la $k^{ème}$ sortie désirée et $\hat{y}_k^p$ la sortie du réseau du neurone k.
La fonction de coût définie en (1.8) donne

$$E^p = \frac{1}{2}\sum_k (y_k^p - \hat{y}_k^p)^2 = \frac{1}{2}\sum_k (e_k^p)^2 \tag{1.16}$$

L'idée est donc de trouver la quantité

$$\Delta_p w_{kj} = -\eta \frac{\partial E^p}{\partial w_{kj}} \tag{1.17}$$

Avec w_{kj} le poids synaptique entre les neurones k et j.
Utilisant la règle de chaîne, nous avons

$$\frac{\partial E^p}{\partial w_{kj}} = \frac{\partial E^p}{\partial e_k^p}\ \frac{\partial e_k^p}{\partial \hat{y}_k^p}\ \frac{\partial \hat{y}_k^p}{\partial v_k^p}\ \frac{\partial v_k^p}{\partial w_{kj}} \tag{1.18}$$

D'après (1.16),(1.15),(1.13) et (1.14), nous avons

$$\frac{\partial E^p}{\partial e_k^p} = e_k^p \ \ , \ \ \frac{\partial e_k^p}{\partial \hat{y}_k^p} = -1 \ , \ \ \frac{\partial \hat{y}_k^p}{\partial v_k^p} = f'(v_k^p) \ et \ \frac{\partial v_k^p}{\partial w_{kj}} = \hat{y}_j^p$$

En combinant ces résultats nous trouvons la valeur de l'équation (1.18) et en portant dans (1.17), nous avons finalement

$$\Delta_p w_{kj} = \eta \delta_k^p \hat{y}_j^p \tag{1.19}$$

avec

$$\delta_k^p = \frac{\partial E^p}{\partial v_k^p} = -e_k^p f'(v_k^p) \tag{1.20}$$

Connexion couche d'entrée couche cachée

Notons $\hat{y}_i^p$, la sortie du $i^{ème}$ neurone de la couche d'entrée. Nous avons

$$\hat{y}_j^p = f(v_j^p) \tag{1.21}$$

où $v_j^p = \sum_i \hat{y}_i^p w_{ji} + b_j$, où j indique l'indice du neurone sur la couche cachée et y_j^p la sortie du $j^{ème}$ neurone. Calculons maintenant δ_j^p.
En utilisant la règle de chaîne nous avons

$$\delta_j^p = \frac{\partial E^p}{\partial \hat{y}_j^p}\ \frac{\partial \hat{y}_j^p}{\partial v_j^p} \tag{1.22}$$

Or d'après (3.21)

$$\frac{\partial \hat{y}_j^p}{\partial v_j^p} = f'(v_j^p) \tag{1.23}$$

Or nous savons que $E^p = \frac{1}{2}\sum_k (e_k^p)^2$
Alors

$$\begin{aligned} \frac{\partial E^p}{\partial \hat{y}_j^p} &= \sum_k e_k^p\ \frac{\partial e_k^p}{\partial \hat{y}_j^p} \\ &= \sum_k e_k^p\ \frac{\partial e_k^p}{\partial v_k^p}\ \frac{\partial v_k^p}{\partial \hat{y}_j^p} \end{aligned} \tag{1.24}$$

Or $e_k^p = y_k^p - f(v_k^p)$ d'après (1.13) et (1.15), donc

$$\frac{\partial e_k^p}{\partial v_k^p} = -f'(v_k^p) \tag{1.25}$$

Et d'après (1.14)

$$\frac{\partial v_k^p}{\partial \hat{y}_j^p} = w_{kj} \tag{1.26}$$

En portant ces deux dernières équations dans (1.24), on a

$$\frac{\partial E^p}{\partial \hat{y}_j^p} = -\sum_k e_k^p \; f'(v_k^p) \; w_{kj} = -\sum_k \delta_k^p w_{kj} \tag{1.27}$$

Finalement nous trouvons

$$\delta_j^p = -f'(v_j^p) \; \sum_k \; \delta_k^p \; w_{kj} \tag{1.28}$$

donc

$$\Delta_p w_{ji} = \eta \; \delta_j^p \; \hat{y}_i^p \tag{1.29}$$

Les équations (1.20) et (1.28) donnent les récursivités des deltas dans la programmation du backpropagation (cf. 1.3.2). La mise à jour des poids entre la couche de sortie et la couche cachée est donnée par (1.19). Entre la couche cachée et la couche d'entrée, on utilise (1.29) pour mettre à jour chaque poids.

1.6.2 Amélioration

Pour optimiser la vitesse de convergence de notre algorithme et pour résoudre une partie du problème des minimums locaux, introduisons un paramètre supplémentaire α nommé "momentum"[21]. Avec

$$\Delta_p w_{ij}(t+1) = \Delta_p w_{ij}(t) + \alpha \Delta w_{ij}(t-1) \tag{1.30}$$

Où $0 < \alpha < 1$ et t itération des époques. $\Delta_p w_{ij}(t)$ est donné par l' équation (1.29) ou (1.19) et $\Delta w_{ij}(t-1)$ désigne la même quantité mais à un instant juste avant. Cette équation (1.30) nous donne la quantité nécessaire pour la mise à jour des poids synaptiques du réseau.

1.6.3 Overfitting

Le "Overfitting" est un phénomène qui arrive au réseau de neurones quand il apprend les détails spécifiques des entrées et non pas leurs caractéristiques générales trouvées dans les données. Ce phénomène est aussi appelé sur-apprentissage ou apprentissage par-cœur.

Pour qu'un réseau soit atteint par ce phénomène il faut que, soit l'entraînement du réseau est trop long, soit le nombre de nœuds cachés est trop grand.

Pour résoudre ce problème on peut citer quelques solutions : l'arrêt prématuré de l'apprentissage du réseau lorsque l'erreur augmente [14], l'apprentissage avec bruit[14] et le *weight decay* [14] qui consiste à pondérer la fonction coût avec la somme des carrés des poids synaptiques. Mais pour ne pas compliquer notre algorithme, on adoptera la pondération des poids comme test de sur-apprentissage.

Notre fonction de coût devient

$$E^p = \frac{1}{2}\sum_k (y_k^p - \hat{y}_k^p)^2 - \frac{\lambda}{2}\sum_i^{Np} (w_i^p)^2 \tag{1.31}$$

Où N_p désigne le nombre de poids dans le réseau et λ est un paramètre dite de régularisation.
Finalement l'équation (1.30) devient

$$\Delta_p w_{ij}(t+1) = \Delta_p w_{ij}(t) + \alpha \Delta w_{ij}(t-1) - \lambda\ w_{ij}(t) \tag{1.32}$$

Dans la pratique, λ doit être un nombre ≤ 0.0001

Chapitre 2

Création d'un réseau de neurones

Dans ce chapitre nous allons créer un réseau de neurones à propagation de l'information vers l'avant, plus précisément un perceptron multicouche. Nous allons utiliser le langage de programmation java [23].

Le Java est un langage de programmation orienté objet développé par Sun Microsystems, qui fut officiellement présenté en 1995. Non seulement c'est un langage de programmation, mais il est aussi un environnement d'exécution.Pourquoi avons-nous choisi ce langage ? Pour répondre à cette question, voyons quelques avantages du Java par rapport aux autres langages de programmations comme le C/C++.

Au niveau de la portabilité (compatibilité entre les systèmes d'exploitation et les architectures matérielles), le C/C++ est très faible [42]. Nous avons une obligation de recompiler le code sur chaque architecture sur laquelle le programme devra tourner. Par contre, le Java, grâce à sa JVM (Machine Virtuelle Java), permet d'avoir un très haut niveau d'abstraction par rapport à la machine. Nous n'aurons pas besoin de nous occuper de la compatibilité matérielle. De plus, avec Java nous n'aurons pas besoin de manipuler les pointeurs qui, en C/C++, peuvent être fatals s'ils sont mal utilisés [42]. Le java nous offre aussi une gestion automatique de mémoire grâce au « Garbage Collector » et une librairie de codes (API) très riche. Dans notre programme, nous utilisons par exemple le JFreeChart et le javacsv, qui sont des API tierces[23].

Bref, Java est en constante amélioration. Chaque nouvelle version apporte son lot de nouveautés au niveau de l'API et apporte de nettes améliorations au niveau des performances et de la stabilité. Il nous permet de développer plus rapidement et d'être plus reproductif.

Grâce à ses avantages cités ci-dessus, le Java nous facilite beaucoup le développement de notre PMC. Plusieurs auteurs ont aussi choisi le langage java pour développer les réseaux de neurones, on peut citer comme référence les travaux de Jeff Heaton [22].

2.1 Les neurones

Nous définissons un neurone comme un objet et chaque neurone peut avoir sa liste des poids d'entrée et sa liste des poids de sortie. Chaque poids doit être initialisé à une petite valeur aléatoire. Pour initialiser ces poids nous utiliserons une fonction qu'on appelle

initNeurone dans la classe Neurone. Nous avons aussi besoin de deux conteneurs pour stocker les anciens deltas associés aux poids synaptiques du neurone.

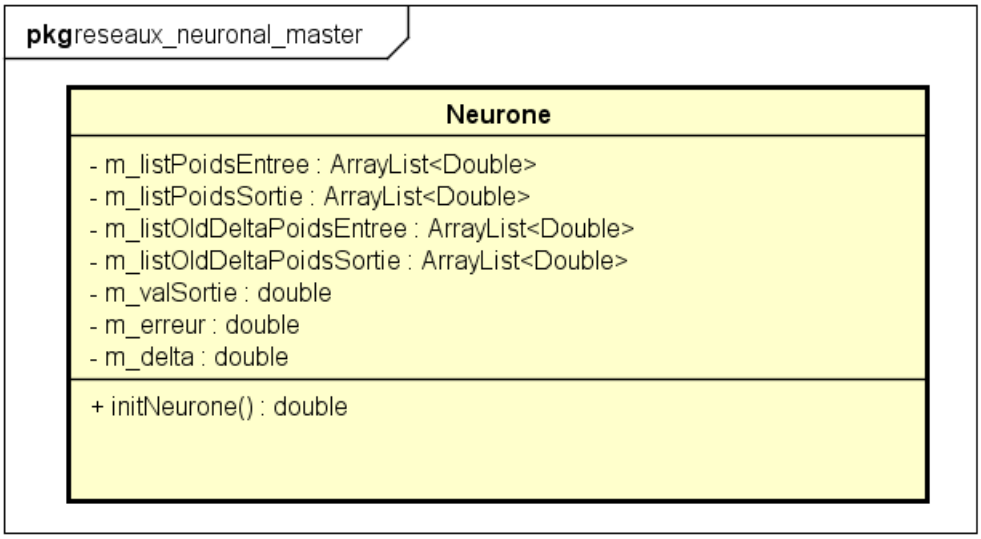


FIGURE 2.1 – Diagramme UML pour la classe Neurone

2.2 L'architecture du réseau

Nous allons considérer un réseau de neurones comme un objet. Nous savons que dans un réseau à propagation de l'information vers l' avant les neurones sont arrangés en couches : une couche d'entrée, des couches cachées et une couche de sortie. Pour ce faire nous allons créer une classe abstraite appelée *Couche* et les différentes couches des classes qui héritent de cette classe.

2.2.1 Notre classe couche

Une couche est composée de neurones, alors dans notre classe nous avons besoin d'une liste qu'on stockera les neurones sur la couche et le nombre de neurones sur la couche.

• La couche d'entrée

La couche d'entrée est une couche, donc elle hérite de la classe mère. Pour initialiser chaque poids des neurones sur la couche, nous utilisons la fonction *initCouche* en faisant appel à la fonction *initNeurone* de la classe neurone. Dans cette classe, nous créons un neurone supplémentaire pour définir le biais.

• Les couches cachées

Les couches cachées sont les couches de connexion entre les neurones de la couche d'entrée et ceux de la couche de sortie. Le nombre de couches cachées peut être supérieur à un, alors il y a aussi la connexion entre les couches cachées. Nous devons donc initialiser chaque poids suivant ces trois types de connexions. Nous devons aussi tenir compte des biais pour chaque couche cachée.

• **La couche de sortie**

La couche de sortie ne contient que l'initialisation et la liste des neurones sur la couche.

Nous avons le diagramme suivant :

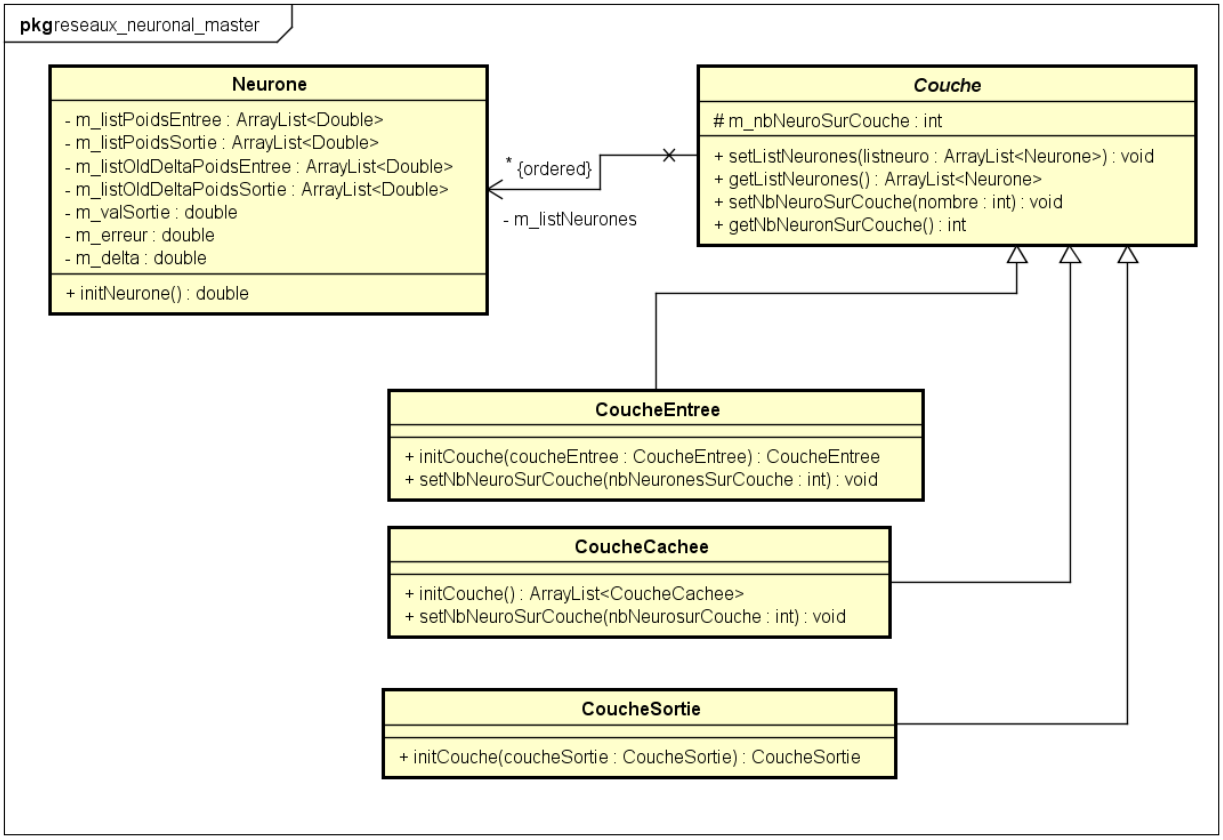


FIGURE 2.2 – Diagramme UML pour les classes Couches

2.2.2 Le réseau

Notre classe *Réseaux* est divisée en deux parties, l'initialisation et la phase d'apprentissage du réseau.

Initialisation du réseau

On initialise un réseau en définissant son nombre de neurones sur chaque couche et son nombre de couches cachées. Ainsi on définit un réseau de la façon suivante : Réseau(nombre de neurones sur la couche d'entrée,nombre de couches cachées, nombre de neurones sur la couche cachée,nombre de neurones sur la couche de sortie). L'initialisation du réseau consiste aussi à initialiser les poids entre les différentes connexions dans le réseau.

La phase d'apprentissage dans la classe réseau

La phase d'apprentissage du réseau consiste à appeler l'algorithme de backpropagation qu'on définira plus tard (cf. 2.3.2).

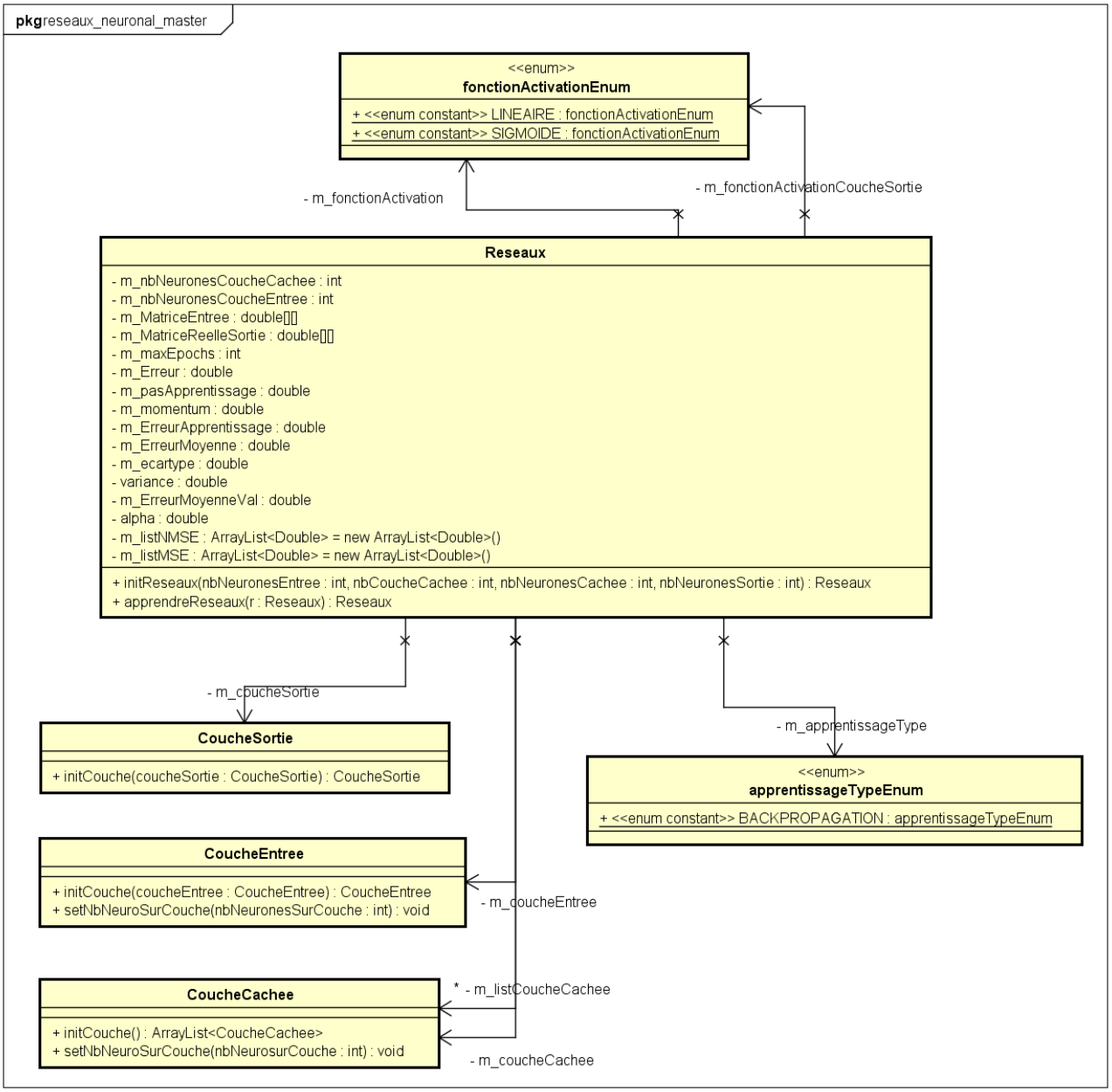


FIGURE 2.3 – Diagramme UML pour la classe Réseaux

Ces deux fonctions forment la classe réseau. Mais pour faire fonctionner le réseau avec l'algorithme de backpropagation, on a besoin de quelques variables supplémentaires comme la matrice d'entrée, la matrice de sortie, le pas d'apprentissage, le momentum et le nombre maximal d'époques. A chaque époque, on fait une mise à jour de tous les poids du réseau.

2.3 Apprentissage du réseau

2.3.1 L'apprentissage du réseau

Pour apprendre le réseau avec l'algorithme de rétro-propagation, nous allons créer une classe abstraite qu'on nommera *Apprentissage* qui contient les fonctions d'activations et leurs dérivées. Nous utilisons dans notre réseau la fonction sigmoïde comme fonction d'activation des neurones des couches cachées et la fonction linéaire pour les neurones de

la couche de sortie.

2.3.1.1 Les erreurs

Nous allons utiliser deux types d'erreurs pour vérifier la validité des résultats estimés et pour voir le comportement du réseau.

• **L'erreur quadratique MSE**

La MSE ou *Mean square error* est calculée à partir de la fonction coût E^p (1.31),

$$MSE = \frac{1}{N.P} \sum_p E^p \tag{2.1}$$

N indique le nombre de neurones sur la couche de sortie donc $N = 1$ dans notre cas. P le nombre de prototypes présentés lors de l'apprentissage du réseau.

• **L'erreur quadratique normalisé NMSE**

Il est très utile de calculer l'erreur quadratique normalisé lorsqu'on fait de la prédiction des séries temporelles. Elle est donnée par

$$NMSE = \frac{1}{P.\sigma^2} \sum_p E^p \tag{2.2}$$

P est le nombre de prototypes et σ l'écart-type de la série. E^p est toujours donné par (1.31)

Ces deux erreurs sont des attributs de notre classe *Apprentissage* mais elles sont calculées dans la classe *Backpropagation*.

2.3.1.2 Dérivation des fonctions d'activation

Prenons la fonction sigmoïde (1.3) nous avons

$$\hat{y}^p = f(v^p) = \frac{1}{1 + \mathrm{e}^{-v^p}} \tag{2.3}$$

où v^p est donné par (1.14).
Calculons maintenant $f'(v^p)$:

$$\begin{aligned} f'(v^p) &= \frac{\partial}{\partial v^p} \frac{1}{1 + \mathrm{e}^{-v^p}} \\ &= \frac{1}{(1 + \mathrm{e}^{v^p})^2} (\mathrm{e}^{-v^p}) \\ &= \frac{1}{(1 + \mathrm{e}^{-v^p})} \frac{\mathrm{e}^{-v^p}}{(1 + \mathrm{e}^{-v^p})} \\ &= \hat{y}^p (1 - \hat{y}^p) \end{aligned} \tag{2.4}$$

Nous utilisons l'équation (2.4) pour programmer la dérivée de la fonction sigmoïde [24]. Pour la fonction linéaire (1.4) on a $f'(v^p) = 1$, si on prend $\alpha = 1$.

2.3.2 Backpropagation

L'algorithme de rétro-propagation des erreurs se divise en deux étapes. La première étape consiste à la propagation de l'information vers l'avant et la deuxième étape c'est la rétro-propagation des erreurs. A chaque prototype, on fait propager les informations et rétro-propager l'erreur.

2.3.2.1 La propagation vers l'avant

Nous avons besoin de trois variables :
— *sortieEstimee* : désigne la sortie du réseau
— *sortieDesiree* : Valeur attendue
— *SomErreur* : qui définit la somme des fonctions de coût de chaque prototype

• Entre la couche d'entrée et la couche cachée

Les sorties de la couche cachée sont données par (1.21) mais on doit exclure le biais parce qu'il n'a pas d'entrée.

• Entre la couche cachée et la couche de sortie

Pour calculer la sortie du réseau, nous utilisons l'équation (1.13) mais en ne tenant pas compte du biais. Calculons tout d'abord (1.14) puis appliquons la fonction d'activation. (1.15)

2.3.2.2 La rétro-propagation des erreurs

La rétro-propagation des erreurs consiste à calculer les deltas (1.20) et (1.28) dans la section (cf. 1.6.1) et de mettre à jour les poids synaptiques utilisant les équations (1.19) et (1.29).

2.3.2.3 La régularisation

Pour pouvoir calculer le second terme de régularisation dans notre fonction de coût (1.31), nous avons besoin d'une fonction que nous appelons*Régularisation*.

Pour exploiter ces trois fonctions dans notre classe backpropagation, nous avons créé une autre fonction que nous appelons *apprendre*. La fonction sert non seulement de propager les entrées et rétro-propager les erreurs pour chaque prototype mais aussi de tester si le réseau subit un overfitting.

Notre classe *Backpropagation* est donc composée par les quatre fonctions : apprendre, propager, régularisation et backpropagation.

Nous pouvons résumer l'apprentissage de notre réseau avec le diagramme suivant :

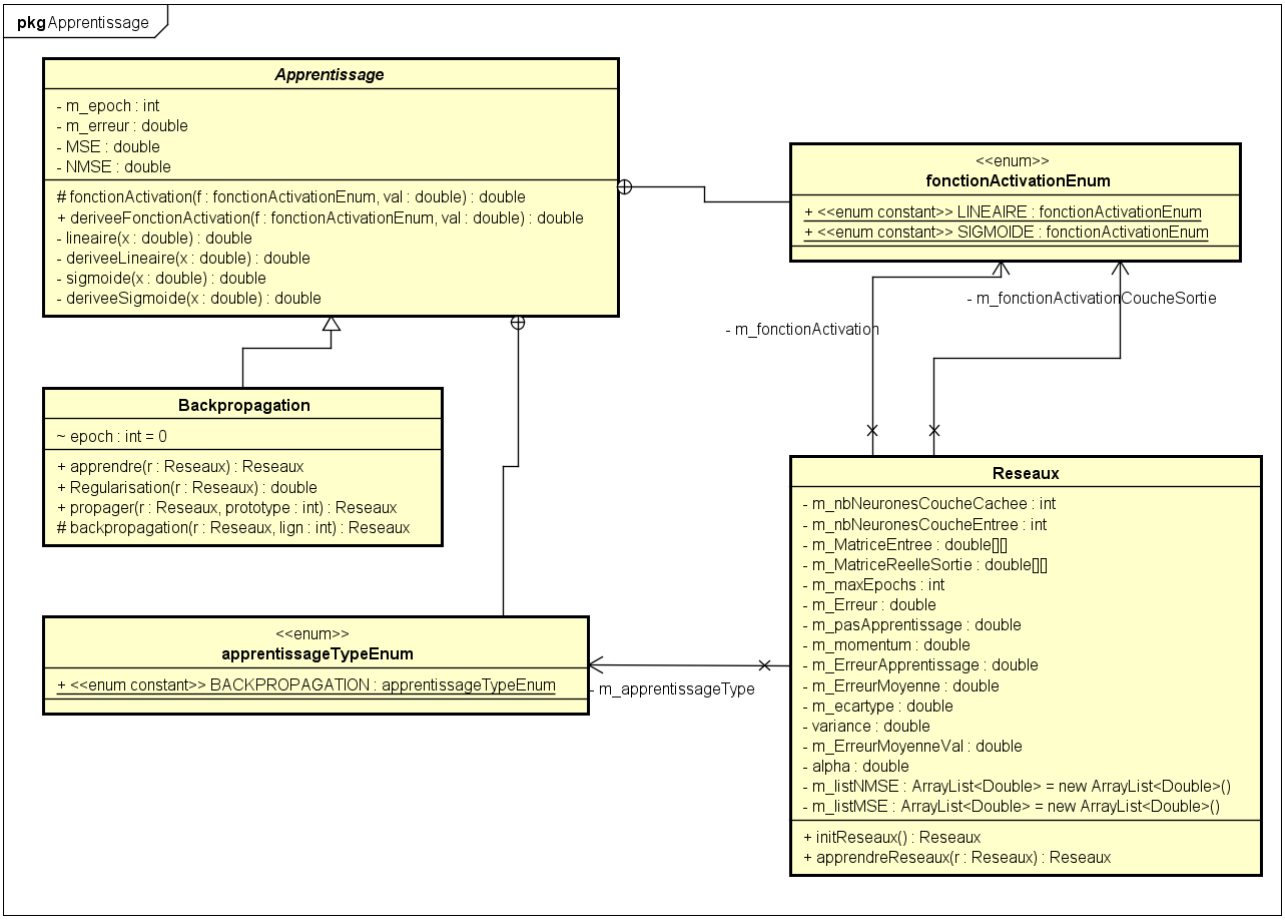


FIGURE 2.4 – Diagramme UML pour les classes Apprentissage et Backpropagation

En résumant, vus en tant qu'objet, les réseaux de neurones sont devenus simple à programmer. Notre perceptron est donc composé de deux paquets de classes. Le premier défini l'architecture du réseau. Il est composé par la classe abstraite couche et les différentes classes filles *CoucheEntree*, *CoucheSortie* et *CoucheCachee*, notre classe *Neurone* et notre classe *Réseaux* qui définit la topologie du réseau(feed-forward dans notre cas). Le deuxième paquet contient notre classe *Backpropagation* qui hérite de notre classe abstraite *Apprentissage*.

Deuxième partie

Application logistique et prédiction des séries temporelles

Chapitre 3

Théorie du chaos et l'application logistique

3.1 Introduction

L'étude du chaos nous fait remonter jusqu' aux travaux du météorologiste Edward Lorenz en 1963[26]. Il a calculé les courants de convections de l'atmosphère à l'aide d'un ordinateur et retrouva l'effet d'amplification considérable des petites différences des conditions initiales. Des petites causes comme des orages localisés peuvent avoir de grands effets sur le temps au niveau de l'hémisphère. La sensibilité de tels systèmes à de très petits changements dans les conditions initiales entrainait l'impossibilité de prédire leur comportement.

Le terme *chaos* fut proposé peu après pour décrire ce genre de situation. Dans les années 1970, Lorenz appliqua ses considérations à la biologie des populations [25] et l'on s'aperçut plus tard que les comportements chaotiques concernent de multiples phénomènes dans divers domaines, comme le fonctionnement du laser[27], l'évolution de l'écosystème [28] ou la cinétique des réactions chimiques[29].

En 1971, Floris Takens et David R montrèrent que le comportement chaotique n'est pas intrinsèquement lié à un grand nombre de paramètres et introduisirent la notion d'*attracteur étrange* pour désigner la courbe caractéristique des paramètres d'un système chaotique [30].

Depuis, le chaos est devenu incontournable dans de nombreux domaines comme : l'étude des systèmes dynamiques, l'électronique, la biologie, la finance et l'économie et dans l'analyse des séries temporelles, etc.

3.2 L'application logistique

L'application logistique est le cas le plus simple d'un système dynamique discret et non autonome [31] .c'est-à-dire dépendant du temps. Elle est apparue pour la première fois en 1838, et qui est utilisée par Pierre-François Verhulst pour modéliser la dynamique de la population en Biologie [32]. En 1985, Mandelbrot a ensuite étudié ce modèle pour ses aspects chaotiques en construisant son diagramme de phase appelé ensemble de Mandelbrot[33].

En général, un système dynamique discret est décrit par un système d'équations aux différences finies ou bien une récurrence.

3.2.1 Définition

Soit μ un paramètre réel strictement positif. L'application logistique est la fonction f définie par :

$$f : \begin{cases} \mathbb{R} \longmapsto \mathbb{R} \\ \\ x \longrightarrow \mu x(1-x) \end{cases} \tag{3.1}$$

Le système dynamique associé est :

Soit la condition initiale $x_0 \in \mathbb{R}$ et soit un nombre entier $n \geq 0$. En partant de ce point on pose :

$$x_1 = f(x_0), x_2 = f(x_1), ..., x_{n+1} = f(x_n) \tag{3.2}$$

La loi de mouvement ou d'évolution est donnée par

$$x_{n+1} = \mu x_n(1-x_n) \tag{3.3}$$

3.3 Observations

Pour étudier le comportement du système en fonction de μ, on fait recours à la théorie du chaos.

3.3.1 Allure de la fonction

Cette fonction est définie sur $\mathbb{R}$ à valeur dans $\mathbb{R}$, mais les cas les plus intéressants se situent entre l'intervalle $[0,1]$ pour $0 < \mu < 4$.
On a $f'(x) = \mu(1-2x)$, pour $x = 1/2$ la fonction prend sa valeur maximale égale à $\frac{\mu}{4}$. D'où les courbes suivantes

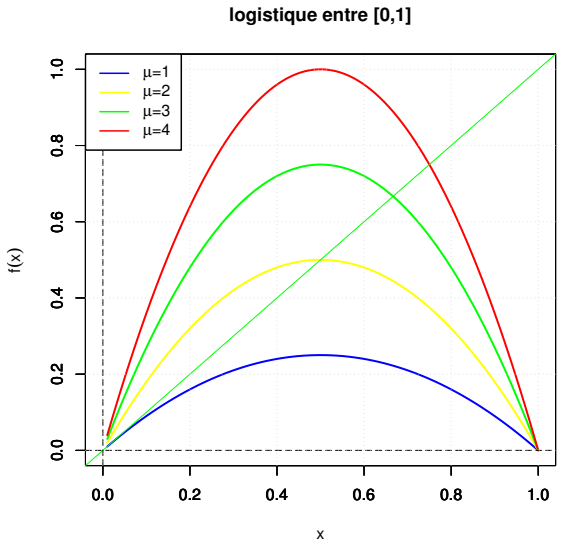


FIGURE 3.1 – Allure de la fonction définie par l'application logistique

3.3.2 Orbite ou trajectoire du système

L'orbite est l'ensemble de tous les points du système noté par $O(x_0)$ où x_0 est le point à l'instant initial.

$$O(x_0) = \{x(0) = x_0, x(1) = \mu x(0)(1 - x(0)), ..., x(n) = \mu x(n-1)(1 - x(n-1))\} \quad (3.4)$$

3.3.3 Points fixes

Un point fixe ou point stationnaire ou point d'équilibre d'un système est un point vérifiant la relation $f(x) = x$.

$$\mu x(1 - x) = x \quad (3.5)$$

En résolvant cette équation, on trouve

$$x_1 = 0 \; et \; x_2 = 1 - \frac{1}{\mu} \quad (3.6)$$

Un point fixe x est attractif si et seulement si :

$$\left| \frac{d}{dx} f(x) \right| < 1 \quad (3.7)$$

Dans le cas où

$$\left| \frac{d}{dx} f(x) \right| > 1 \quad (3.8)$$

x est répulsif. Nous avons alors :
Si x_1 et x_2 sont attractifs alors

$$\left| \frac{d}{dx} f(x_1) \right| = \mu(1 - 2x_1) = \mu < 1$$

et

$$\left| \frac{d}{dx} f(x_2) \right| = \mu(1 - 2x_2) = |2 - \mu| < 1$$

En résolvant ces deux inéquations nous avons

— Si $0 < \mu < 1$, x_1 est un point fixe attractif et x_2 un point répulsif pour le système.

— Si $1 < \mu < 3$ x_1 devient un point répulsif et x_2 devient un point attractif.

Voyons ces deux cas suivant le diagramme de cobweb :

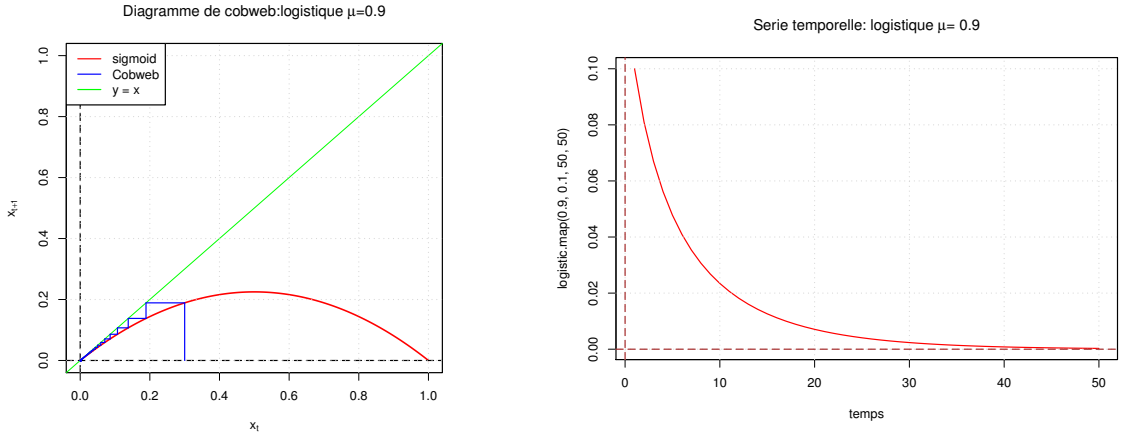


FIGURE 3.2 – Attracteur $x_1 = 0$

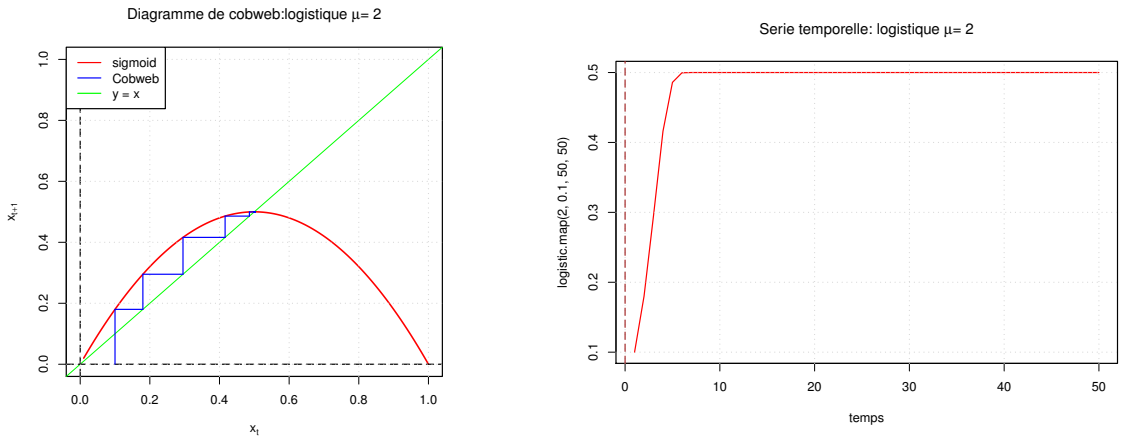


FIGURE 3.3 – Attracteur $x_2 = 1 - \frac{1}{\mu}$ et $x_1 = 0$ répulsif

Dans ces deux figures, le diagramme de gauche est appelé diagramme de cobweb, il a pour but de visualiser le comportement du système en fonction du paramètre μ en partant d'une condition initiale donnée x_0. Le diagramme de droite représente l'application en tant que série temporelle, on appelle ce diagramme un chronogramme.

3.3.4 Points périodiques et Orbites périodiques

Une orbite $O(x_0)$ est dite périodique s'il existe $p > 0$ tel que

$$x(n+p) = x(n), \forall n \in \mathbb{N} \tag{3.9}$$

Une orbite périodique est donc une suite de points périodiques, appelés points périodiques de période p du système.

Tous les points périodiques de période p sont solution de l'équation :

$$f^{(p)}(x) = x \tag{3.10}$$

avec $f^{(p)}(x) = \underbrace{f(f(f...(f(x)...)))}_{p-fois}$

Notons que les points fixes sont des points périodiques de période 1. Trouvons alors les points périodiques de période 2. Pour trouver ces points, on résoud

$$f(f(x)) = x \tag{3.11}$$

$$\mu^2 x(1-x)(1-\mu x(1-x)) = x$$

$$\mu^3 x^4 - 2\mu^3 x^3 + \mu^2(1+\mu)x^2 - (\mu^2 - 1)x = 0$$

On connait déjà deux racines de ce polynôme qui sont nos deux points fixes x_1 et x_2. Après factorisation, les points périodiques d'ordre 2 sont solution de l'équation

$$\mu^2 x^2 - (\mu^2 - \mu)x + \mu + 1 = 0 \tag{3.12}$$

Ce qui nous donne

$$x_3 = \frac{\mu + 1 - \sqrt{(\mu-3)(\mu+1)}}{2\mu} \tag{3.13}$$

$$x_4 = \frac{\mu + 1 + \sqrt{(\mu-3)(\mu+1)}}{2\mu} \tag{3.14}$$

— Si $\mu < 3$, il n'y a pas de points périodiques
— Si $\mu = 3$, il n'y a qu'un seul point périodique qui coïncide avec l'un des points fixes.
— Si $\mu > 3$, il existe deux points périodiques distincts et le système possède une orbite périodique de période 2.

3.3.5 Orbite attractive et orbite répulsive

Une orbite est attractive ou respectivement répulsive si chacun de ses points est un point fixe attractif ou respectivement répulsif.
L'orbite $O(x_0)$ est attractive si et seulement si

$$\left|\frac{d}{dx}f^{(p)}(x_0)\right| = \left|\prod_{j=0}^{p-1} f'(x(j))\right| < 1 \tag{3.15}$$

L'orbite $O(x_0)$ est répulsive si et seulement si

$$\left|\frac{d}{dx}f^{(p)}(x_0)\right| = \left|\prod_{j=0}^{p-1} f'(x(j))\right| > 1 \tag{3.16}$$

Pour connaitre si l'orbite est attractive ou répulsive, prenons deux cas particuliers :

Prenons $\mu = 3,2$

$$\left|\frac{d}{dx}f^{(2)}(x_3)\right| = |f'(x_3)f'(x_4)| \approx 0.16 < 1$$

Dons l'orbite est attractive.
Prenons $\mu = 3,5$

$$\left|\frac{d}{dx}f^{(2)}(x_3)\right| = |f'(x_3)f'(x_4)| = \frac{5}{4} > 1$$

Dans ce cas l'orbite est répulsive.

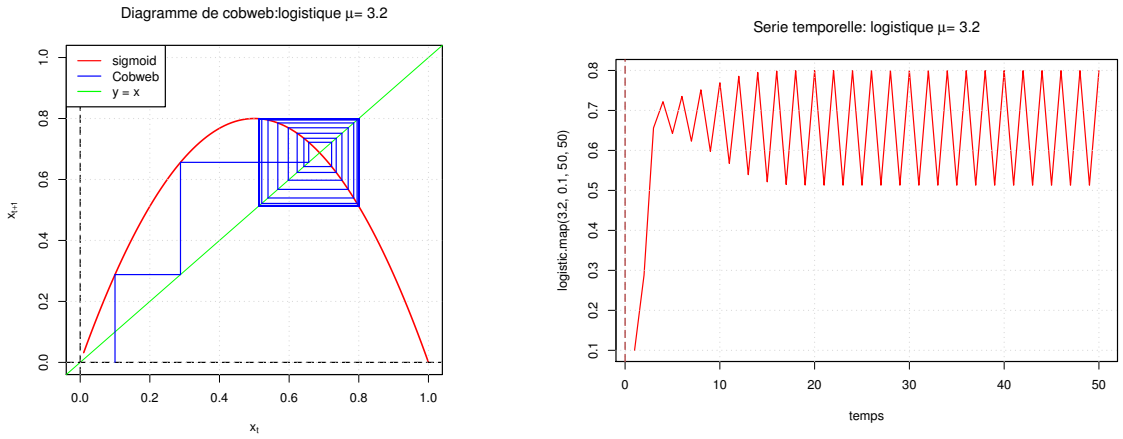


FIGURE 3.4 – Orbite $\{x_3, x_4\}$ attractive

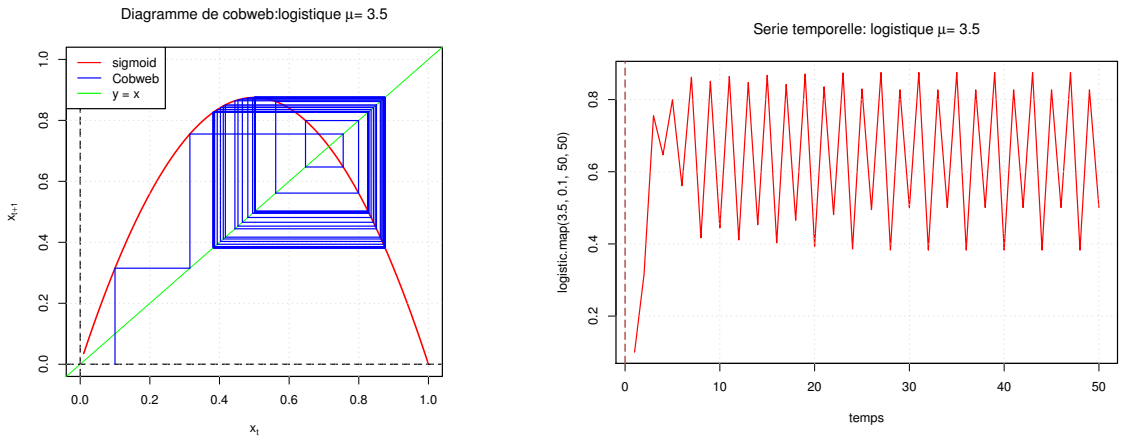


FIGURE 3.5 – Orbite $\{x_3, x_4\}$ répulsive

3.4 Les bifurcations

Une bifurcation est un changement d'état qualitatif du système en un autre pour une valeur de ses paramètres. Cette valeur est appelée valeur ou point de bifurcation. Notons deux types de bifurcations essentiels pour notre étude :

- Une bifurcation transcritique : le système subit ce type de bifurcation au point μ_0, s'il existe deux branches de points fixes qui se croisent en ce point.
- Une bifurcation de dédoublement de période : On dit qu'un système subit au point μ_0, une bifurcation de dédoublement de période s'il existe une branche de points fixes $\gamma_1(\mu)$ qui passe par le point (μ_0, x_0) de telle façon qu'elle est attractive pour $\mu < \mu_0$ ou répulsive pour $\mu > \mu_0$ et deux branches de points périodiques de période 2 prennent début au point (μ_0, x_0)

3.4.1 Branches de points fixes et de points périodiques

Toutes applications continues $\gamma : I \longmapsto \mathbb{R}$ qui vérifient

$$f(\mu, \gamma(\mu)) = \gamma(\mu) \tag{3.17}$$

sont appelées branches de points fixes du système.
Les branches de points périodiques sont aussi définies de la même manière. Quand $0 < \mu < 3$, le système possède deux points fixes donc deux branches de points fixes qui sont $\gamma_1(\mu) = 0$ et $\gamma_2(\mu) = 1 - \frac{1}{\mu}$

De même, pour $\mu > 3$, il existe deux points périodiques de période 2 donc deux branches de points périodiques qui sont :

$$\gamma_3(\mu) = \frac{\mu + 1 - \sqrt{(\mu - 3)(\mu + 1)}}{2\mu} \tag{3.18}$$

$$\gamma_4(\mu) = \frac{\mu + 1 + \sqrt{(\mu - 3)(\mu + 1)}}{2\mu} \tag{3.19}$$

Donc au point $\mu = 3$, le système subit une bifurcation transcritique.
Pour étudier la stabilité de ces orbites, calculons

$$\left(\frac{d}{dx} f(x_3)\right)\left(\frac{d}{dx} f(x_4)\right) = 4 + 2\mu - \mu^2 \tag{3.20}$$

L'orbite périodique $\{x_3, x_4\}$ est stable si $|4 + 2\mu - \mu^2| < 1$.
On trouve une seule valeur $\mu_1 = 1 + \sqrt{6}$ qui vérifie cette condition. Le système subit au point μ_1 une bifurcation de dédoublement de période et deux nouvelles orbites périodiques de période 2 apparaissent. Pour le système initial, ces orbites sont de période 4.

On remarque aussi que quand $\mu > \mu_1$,il existe un point $\mu_2 \approx 3,54$ tel que ces orbites deviennent répulsives et une autre orbite de période 8 apparait. Ce dédoublement de période continue. On peut trouver une suite de points de bifurcations $\mu_1 < \mu_2 < ... < \mu_n$. A chaque μ_i une orbite périodique de période 2^i apparait. Cette suite converge vers une limite μ_l telle que

$$\mu_l = \lim_{i \longrightarrow \infty} \mu_i \approx 3,5699 \tag{3.21}$$

Pour $\mu > \mu_l$, le système devient complexe et imprévisible donc le chaos total.

En résumé on a le diagramme ci-dessous appelé diagramme de bifurcation. Pour tracer un tel diagramme, on trace sur un plan toutes les branches de points fixes et périodiques du système.

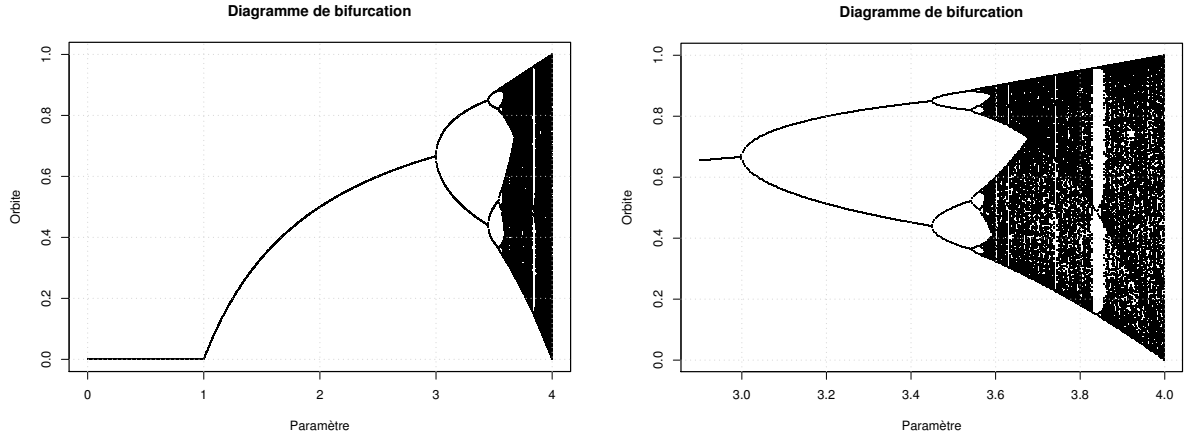


FIGURE 3.6 – Diagramme de bifurcation de l'application logistique

3.5 Les exposants de Lyapounov

Les exposants de Lyapounov servent en général pour détecter la présence de chaos dans un système dynamique. Plus précisément ils peuvent caractériser la sensibilité du système aux conditions initiales et le taux de divergence de l'évolution des trajectoires du système issu des conditions initiales proches.

Soient deux conditions initiales très proches x_0 et $x_0 + \varepsilon$. Si on cherche à évaluer la distance exponentielle entre les deux orbites formées par ces deux conditions, après n itérations, on a

$$\left| f^{(n)}(x_0) - f^{(n)}(x_0 + \varepsilon) \right| \approx \mathrm{e}^{n\lambda(x_0)}$$

En faisant tendre ε vers zéro

$$\mathrm{e}^{n\lambda(x_0)} \approx \left| \frac{d}{dx} f^{(n)}(x_0) \right|$$

Or

$$\frac{d}{dx} f^{(n)}(x_0) = \prod_{i=0}^{n-1} f'(x(i))$$

Finalement, on a les exposants de Lyapounov

$$\lambda(x_0) = \lim_{n \longrightarrow +\infty} \frac{1}{n} \sum_{i=0}^{n-1} \ln \left| f'(x(i)) \right| \tag{3.22}$$

— Si $\lambda(x_0) < 0$, les orbites du système sont stables.
— Si $\lambda(x_0) > 0$, les orbites du système sont instables et le taux de divergence entre deux orbites voisines augmente exponentiellement, donc vers le chaos.

Pour calculer les exposants de Lyapounov du système, prenons $n = 500$ et $x_0 = 0.1$ On a

$$\lambda(x_0) = \frac{1}{500} \sum_{i=1}^{500} \ln |\mu(1 - 2x_i)| \tag{3.23}$$

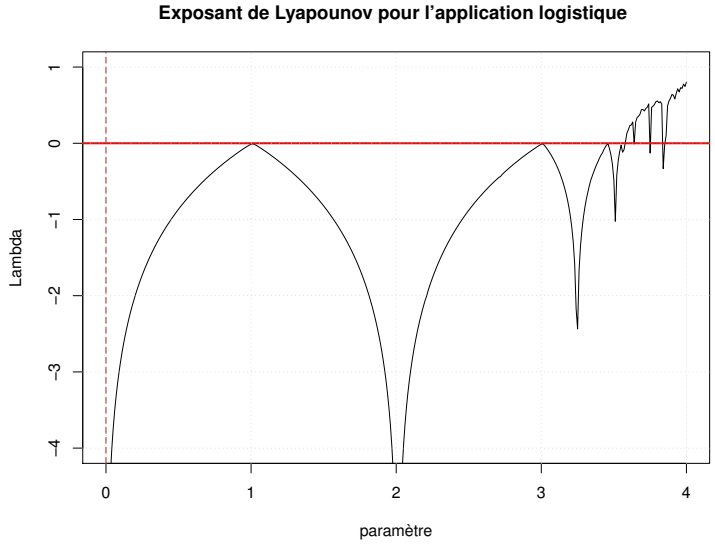


FIGURE 3.7 – Exposant de lyapounov de l'application logistique en fonction du paramètre μ

3.6 Conclusion

Dans ce chapitre, nous avons étudié théoriquement le comportement du système dynamique décrit par l'application logistique, suivant la valeur du paramètre μ. On peut donc résumer que : Si $0 < \mu < 1$, le système tend vers l'attracteur $x_1 = 0$. Pour $1 < \mu \leq 3$, le système converge vers le point $x_2 = 1 - \frac{1}{\mu}$. Si $3 < \mu \leq 3,5699$ le système subit des bifurcations de dédoublement de période et pour $3,5699 < \mu < 4$ le système devient chaotique.

Chapitre 4

Séries temporelles et prédiction

4.1 Introduction

L'étude des séries temporelles est devenue une discipline incontournable de la science. Elle est souvent abordée dans la branche de l'économétrie mais on trouve aussi des applications dans de nombreux domaines tels que l'étude de la consommation électrique, l'étude des séries astronomiques : le nombre de tâches solaires, l'étude des séries environnementales, etc. Dans ce chapitre voyons la définition, les types et l'objectif de l'analyse des séries temporelles. Illustrons ensuite avec un exemple et enfin voyons ce qu'est la prédiction des séries temporelles avec un réseau de neurones artificiel.

4.2 Définition

Une série temporelle ou série chronologique ou bien chronique est une succession d'observations au cours du temps d'un phénomène. On peut aussi la définir comme une suite finie $(x_1, x_2, ..., x_n)$ de donnés indexées par le temps. Le temps peut être le jour, l'année ou le mois... et n est appelé longueur de la série. Les séries temporelles sont souvent présentées dans un graphe nommé chronogramme, où l'on porte le temps en abscisse et la valeur observée en ordonnée.

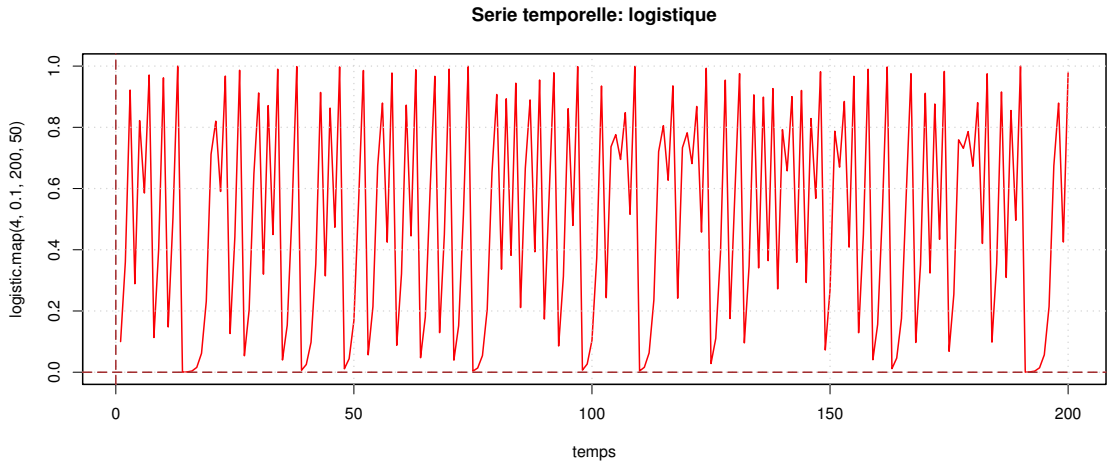


FIGURE 4.1 – Exemple de chronogramme généré par la dynamique logistique

4.3 Types de séries chronologiques

Les séries temporelles sont classées suivant la façon dont on les mesure dans le temps. Si elles peuvent être continues dans le temps, on dit par convention que la série temporelle est continue. Par contre si les mesures peuvent être prises comme un ensemble discret dans le temps, alors on dit que la série est discrète.

En général, dans une série chronologique continue, les variables observées sont des variables continues. Analyser une telle série consiste habituellement à l'échantillonner ou la numériser avec des intervalles de temps réguliers pour donner une série discrète. Pour les séries temporelles discrètes, elles peuvent être, soit échantillonnées à partir des séries continues, soit intrinsèquement comme une série discrète.

4.4 Prévision

En général, les principaux objectifs d'analyse des séries temporelles sont la description, la modélisation et la prédiction. La description consiste à décrire les données en utilisant des méthodes statistiques [34] et la modélisation a pour but de trouver un modèle statistique approprié dans le but de décrire le processus de génération des données [34]. Quant à la prédiction ou prévision, elle est un problème particulier qui implique le traitement d'échantillons évoluant dans le temps [36] et se réfère au processus par lequel les valeurs futures d'un système sont prédites en se basant sur les informations obtenues à partir des données passées et présentes [37].

4.4.1 Les méthodes

Plusieurs sont les méthodes statistiques capables d'appréhender le problème de prédiction des séries temporelles telles que les lissages exponentiels [35], le processus ARMA [35]. Il existe aussi des méthodes par apprentissage artificiel ou *Machine Learning* qui sont capables de résoudre ce problème de prédiction, tels que les réseaux de neurones que nous développons dans ce travail. Cependant il existe aussi plusieurs approches intelligentes pour analyser et faire la prédiction des séries temporelles, notamment la régression linéaire, Kalman filtering [39], les systèmes flous [38] ou bien les modèles de Markov ou les machines à vecteurs de support [40]. Mais pourquoi avons nous choisi le réseau de neurones artificiels ? Contrairement aux méthodes statistiques, les réseaux de neurones apprennent la non linéarité directement aux entrées. Ils peuvent aussi modéliser à la fois les structures linéaires et non linéaires de la série. Sans connaitre la description du modèle de la série, avec un RNA on peut directement faire de la prédiction.

4.5 La prévision par le perceptron multicouche

La prédiction des séries temporelles par les réseaux de neurones se divise en deux : la phase d'apprentissage du réseau et la prédiction proprement dite. Les données sont aussi divisées en deux, la première est utilisée pour l'apprentissage et la seconde pour la prédiction.

4.5.1 Recherche de l'architecture de notre PMC

La recherche d'une architecture optimale est essentielle lors de la résolution d'un problème de prédiction avec un RNA car peu de paramètres perdraient les informations et trop de paramètres consommeraient beaucoup de temps. Dans notre cas, puisque nous avons utilisé un PMC, le choix de l'architecture du réseau consiste à choisir le nombre d'unités d'entrées, le nombre de couches cachées et d'unités cachées, sachant que le nombre d'unités de sortie est égale à un.

4.5.1.1 Le nombre d'unités d'entrée

Pour trouver le nombre d'unités d'entrée du réseau, nous utilisons l'algorithme de Takens qui permet, en système dynamique, de déterminer la dimension de plongement de l'espace de phase construit [41]. Considérons la série temporelle de longueur n :$x(1), x(2), ..., x(n)$

1) On construit une séquence de vecteurs à partir de cette série que l'on note $\bar{x}(i)$.

On a

$$\bar{x}(i) = \begin{Bmatrix} x(i) \\ x(i+\tau) \\ x(i+2\tau) \\ . \\ . \\ . \\ x(i+(n-1)\tau) \end{Bmatrix}$$

τ est appelé paramètre de délais.

2) On définit une matrice de covariance θ formée par $\bar{x}(i) * \bar{x}(i)^T$. Où $\bar{x}(i)^t$ est la transposée de la matrice $\bar{x}(i)$ et $*$ désigne la multiplication de deux matrice.

$$\theta = \left\langle \bar{x}(i) * \bar{x}(i)^T \right\rangle$$

3) On calcule les vecteurs propres et les valeurs propres par ordre décroissant de la matrice θ.

On peut remarquer que la matrice θ est une matrice symétrique, alors on peut appliquer la méthode de Jacobi pour trouver les valeurs propres et vecteurs propres de la matrice.

4) On définit l'erreur d'approximation moyenne $\varepsilon_l = \sqrt{\lambda_l + 1}$. λ_l sont les valeurs propres de θ.

5) On trace ϵ_l en fonction de l.

6) La première valeur de l correspondant au premier plateau de la courbe donne le nombre d'unités d'entrée du réseau.

Notre classe Takens

```
1  //CLASSE TAKENS
2  package Outils;
3  /* @author Manantena Kiady*/
4  public class Takens {
5      double[] x_b;
6      double [] x;
7      int ndim;
8      //constructeur
9      public Takens(int d, double[] vect){
10         ndim = d;
11         x_b = new double[ndim];
12         x = new double[ndim];
13         for(int i = 0; i < d; i++)
```

```
14          x_b[i] = vect[i];
15      }
16      //covariance
17      public Matrix prod_vec(Matrix X, Matrix X_b){
18          Matrix mat = new Matrix(ndim,ndim);
19          for(int i = 0; i<ndim; i++){
20              for(int j = 0; j < ndim; j++)
21                mat.setValeur(i, j, X.getValeur(0, i)*X_b.getValeur(j, 0));
22          }
23          return mat;
24      }
25      //conversion vercteur en matrice 1d
26      public Matrix vect_col(double[] vect){
27          Matrix matri = new Matrix(ndim,1);
28              for (int k = 0; k<ndim ; k++)
29                  matri.setValeur(k, 0, vect[k]);
30      return matri;
31      }
32      public Matrix vect_lig(double[] vect){
33          Matrix matri = new Matrix(1,ndim);
34          for (int k = 0; k<ndim ; k++)
35              matri.setValeur(0, k, vect[k]);
36          return matri;
37      }
38  }
```

On a aussi besoin d'une autre classe que nous appellerons *Jacobi.java*. Cette classe contient non seulement la méthode de Jacobi et l'algorithme de Jacobi (Annexe B) mais aussi le calcul du nombre d'unités d'entrée.

```
1       //erreur d'approximation
2       public double[] erreur_approx(double [] mat){
3           double[] lambda = new double [mat.length];
4           for(int i = 0; i < mat.length; i++)
5           {lambda[i] = Math.sqrt(mat[i] + 1);
6           }
7          return lambda;
8       }
9       //retour au premier plateau de la courbe
10      public int nbUniteEntree(double[] lambda){
11          int count = 0;
12          double erreur, c1,c2;
13          c1 = lambda[0];
14          c2 = lambda[1];
15          for(int i = 2; i < lambda.length; i++){
16           erreur = Math.abs(c2 - c1);
17              if((erreur <= 0.1)&&(erreur > 0.00001)){
18                  count = i ;
19                  break;
20              }else{
21                  c1 = c2;
22                  c2 = lambda[i];
23              }
```

```
24          }
25          return count;
26      }
```

Le nombre de couches cachées

Le nombre de couche cachée est donnée par le théorème de Cybenko, disant qu'une seul couche cachée peut approximer toute fonction continue[16].

Le nombre d'unités cachées

On va réaliser plusieurs expériences en variant le nombre de neurones sur la couche cachée,le nombre d'époques maximal. On prendra comme nombre gagnant ce qui minimise la MSE et la NMSE durant l'apprentissage du réseau.

4.5.2 La phase d'apprentissage du réseau

On prend les 100 premières prototypes de la série pour faire apprendre le réseau de neurones. Un prototype est défini comme un vecteur de longueur égal au nombre d'unités d'entrée du réseau composé par des valeurs de la série temporelle. Si on pose n le nombre d'unités d'entrée alors les matrices d'entrée et de sortie se mettent sous les formes suivantes

$$\underbrace{\begin{pmatrix} 1.0 & x(0) & x(1) & ... & x(n-2) & x(n-1) \\ 1.0 & x(1) & x(2) & ... & x(n-1) & x(n) \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 1.0 & x(99) & x(100) & ... & x(97+n) & x(98+n) \end{pmatrix}}_{ENTREE} \underbrace{\begin{pmatrix} x(n) \\ x(n+1) \\ . \\ . \\ x(n+99) \end{pmatrix}}_{SORTIE}$$

FIGURE 4.2 – Structure des matrices d'entrée et de sortie du réseau pour l'apprentissage

La première colonne de la matrice d'entrée désigne les biais que l'on initialise à +1. La matrice de sortie est composée par les valeurs désirées du problème. Chaque ligne désigne un prototype et la première ligne donne le premier prototype.

4.5.3 La Prédiction

On distingue deux types de prédiction des séries temporelles avec un RNA. La prédiction à un pas en avant ou prédiction à court terme et la prédiction à plusieurs pas en avant ou prédiction à long terme.

4.5.3.1 Prédiction à un pas en avant

La prédiction à un pas en avant consiste à faire la prédiction d'une seule valeur future de la série en choisissant des prototypes autres que ceux utilisés pour l'apprentissage du réseau.

4.5.3.2 Prédiction à plusieurs pas en avant

Dans une prédiction à plusieurs pas en avant, le réseau est itéré en boucle. La sortie obtenue par le réseau doit être systématiquement rétro-propagé dans l'itération suivante. Si on prend la $p^{ème}$ prototype et supposons que le nombre d'unités d'entrée est n, alors le réseau donne une estimation de la $(p+n-1)^{ème}$ valeur de la série temporelle. Nous avons

$$\begin{bmatrix} 1^{ère} \\ 2^{ème} \\ 3^{ème} \\ . \\ k^{ème} \end{bmatrix} \underbrace{\begin{pmatrix} 1.0 & x(p-1) & x(p) & ... & x(p-2+n) \\ 1.0 & x(p) & x(p+1) & ... & \hat{x}(p+n-1) \\ 1.0 & x(p+1) & ... & \hat{x}(p+n-1) & \hat{x}(p+n) \\ . & . & . & . & . \\ 1.0 & ... & ... & ... & \hat{x}(p+n+k-2) \end{pmatrix}}_{ENTREE} \underbrace{\begin{pmatrix} \hat{x}(p+n-1) \\ \hat{x}(p+n) \\ \hat{x}(p+n+1) \\ \\ \hat{x}(p+n+k-1) \end{pmatrix}}_{SORTIE}$$

FIGURE 4.3 – Structure de la matrice d'entrée et de sortie pour une prédiction à plusieurs pas en avant

k désigne le nombre d'itérations ou le nombre de pas qu'on veut prédire. La matrice de sortie ici est formée par les valeurs estimées. $\hat{x}$ désigne une valeur estimée de x.

Contrairement à la prédiction à un pas en avant, la prédiction à plusieurs pas en avant donne directement plusieurs valeurs prédites consécutives de la série temporelle.

Pour évaluer l'erreur de prédiction, on calcul l'erreur de prédiction moyenne notée INMSE définie par :

$$INMSE = \frac{1}{\sigma^2 M} \sum_{t=1}^{M} (x_t - \hat{x}_i)^2 \tag{4.1}$$

Où M est le nombre d'itérations et σ^2 la variance des x_i. x_t et $\hat{x}_t$ sont respectivement la valeur désirée et la valeur prédite pour l'itération t.

Chapitre 5

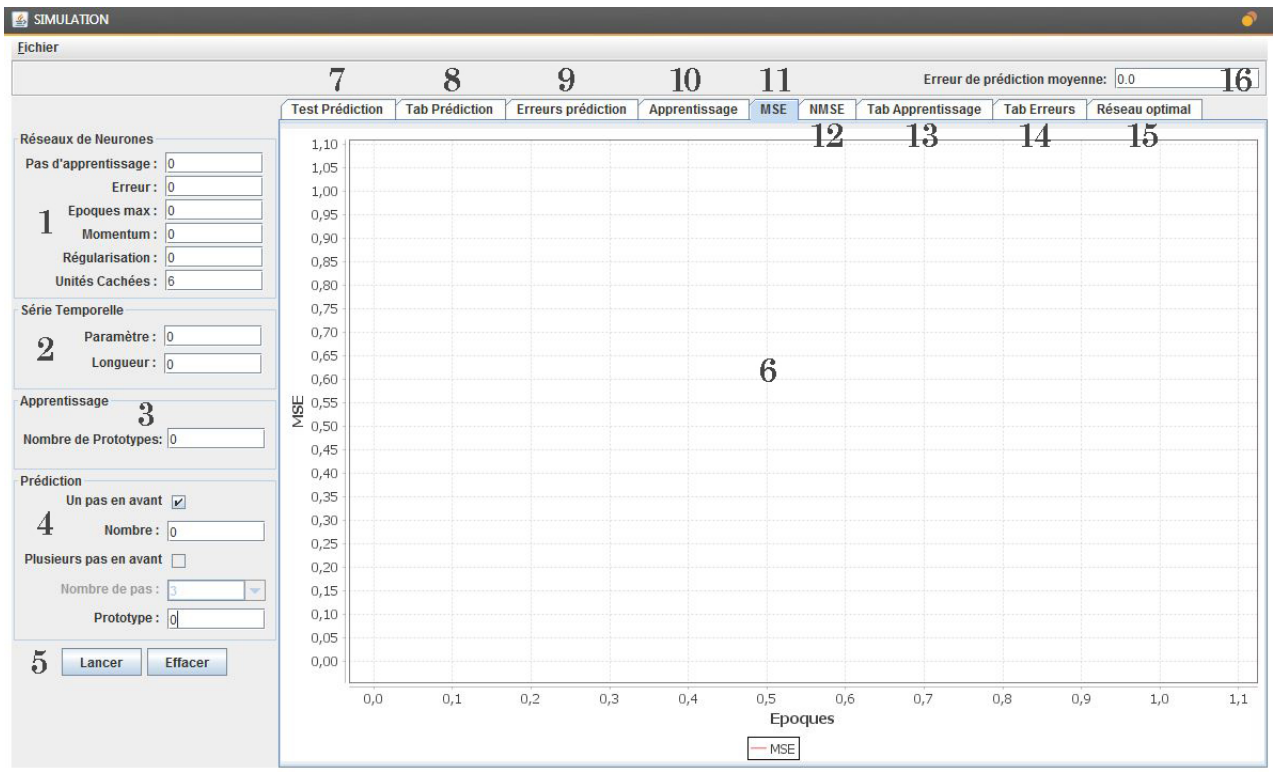
Applications et résultats

5.1 Présentation des données

Nous allons générer trois séries temporelles de longueur 500 avec l'application logistique. Pour la première, nous prendrons $\mu = 2$ avec $x_0 = 0.1$, pour la seconde,choisissons $\mu = 3.5$ et $x_0 = 0.1$ et pour la dernière prenons $\mu = 4$ et $x_0 = 0.1$. Nous avons observé dans le chapitre 3 que pour $\mu = 2$, le système tend vers l'attracteur 0.5 (3.6) ; si $\mu = 3.5$ le système oscille entre deux orbites répulsives (FIGURE 3.6) et pour $\mu = 4$ il devient chaotique (FIGURES 3.6 et 3.7).

5.2 Simulation

Pour ne pas perdre dans la compilation du programme, nous avons créé une interface de simulation



(1) Panneau de configuration du réseau de neurones. Il contient le pas d'apprentissage, l'époque maximale, le paramètre de régularisation, l'erreur, le momentum et le

nombre d'unité caché. L'erreur joue un rôle de limite pour l'apprentissage, si au cours de l'apprentissage la MSE atteint cette valeur alors l'apprentissage s'arrêtera.

(2) indique le panneau réservé à la série temporelle. On entre la longueur de la série et le paramètre de non linéarité de l'application logistique μ.

(3) On entre ici le nombre de prototypes nécessaires pour l'apprentissage du réseau.

(4) indique un panneau pour choisir le type de prédiction qu'on veut soit un pas en avant soit plusieurs pas en avant.

(5) désigne deux boutons, le lancement de la prédiction ou l'effacement des données sur les 3 panneaux 1,2 et 3.

(6) Le panneau d'affichage. Ce panneau affiche soit des graphes soit des tableaux ou bien des figures.

(7) à (15) désignent des onglets qui nous servent à parcourir et visualiser facilement les résultats de la simulation. (7) contient les résultats de la prédiction en tant que chronogrammes tandis que (8) affiche ces mêmes résultats mais sous forme de tableau.(9) nous donne la courbe des erreurs de prédictions en fonction du nombre d'itérations. (10) affiche une courbe représentative du comportement du réseau au cours de l'apprentissage. (11) et (12) contiennent des courbes affichant la MSE et la NMSE en fonction de l'époque. (13) et (14) contiennent des tableaux affichant les valeurs de (10),(11) et (12). Et (15) affiche l'architecture idéale pour la prédiction.

(16) Panneau d'affichage de l'erreur de prédiction moyenne INMSE.

5.3 Résultats

5.3.1 Séries non chaotiques : $\mu = 2$ et $\mu = 3.5$

a) Architecture optimale

Dans ces deux cas, l'algorithme de Takens (cf. 4.3.1) donne 2 comme nombre d'unité d'entrée optimal. En variant les neurones de la couche cachée, on trouve que le nombre gagnant qui donne la bonne prédiction est égal à 2. D'où l'architecture optimale de notre perceptron multicouches.

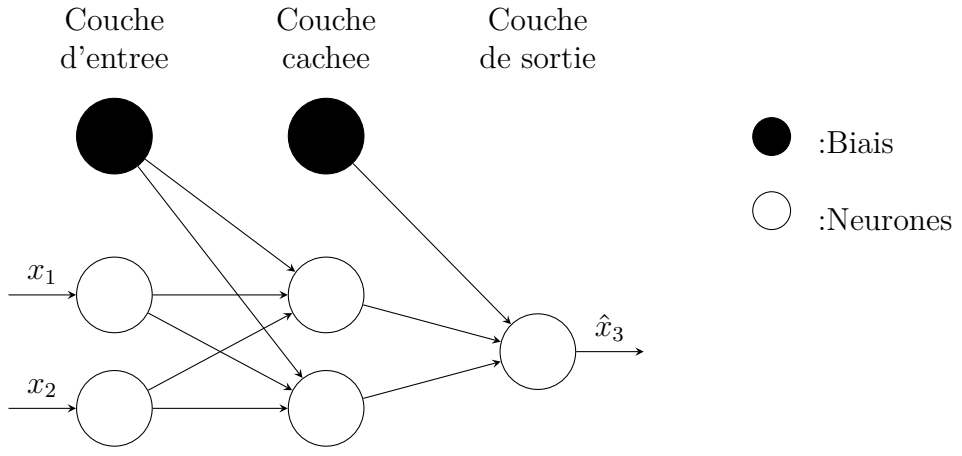


FIGURE 5.1 – Architecture optimale du réseau pour $\mu = 2$ et $\mu = 3.5$

b) Apprentissage de notre PMC

Pour choisir le pas d'apprentissage, le momentum, le nombre d'époques maximal et le paramètre de régularisation nous avons fait quelques expériences. Nous avons pris la

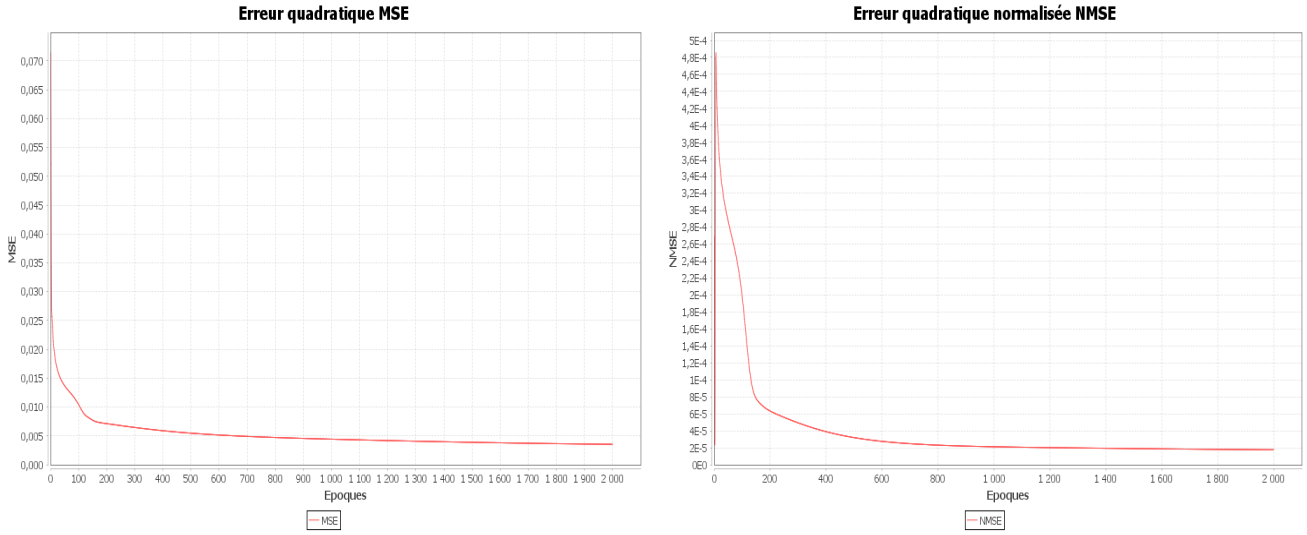
configuration qui minimise notre MSE et NMSE. Nous avons

	$\mu = 2$	$\mu = 3,5$
Pas de l'apprentissage	0.2	0.4
Momentum	0.5	0.5
Paramètre de régularisation	0.01	0.00001
Nombre d'époque maximal	1	2000

TABLE 5.1 – Configurations de l'apprentissage du perceptron pour $\mu = 2$ et $\mu = 3.5$

Pour $\mu = 2$, nous avons obtenu :$MSE = 0.00801275$ et $NMSE = 1.009162110^{-5}$.

Dans le cas de $\mu = 3.5$, nous avons les deux graphes suivantes qui représentent l'évolution de l'MSE et NMSE en fonction de l'époques au cours de l'apprentissage de notre réseau.



(a) (b)

FIGURE 5.2 – Evolution de l'MSE (a) et NMSE (b) durant l'apprentissage du réseau

c) Résultats des prédictions

• Prédictions à un pas en avant

Nous avons choisi le $300^{ème}$ prototype pour la prédiction, donc les valeurs prédites sont à partir de $x(301)$. Nous avons aussi choisi de faire 10 prédictions successives pour 10 valeurs existantes. C'est à dire $x(301)$ à $x(310)$.

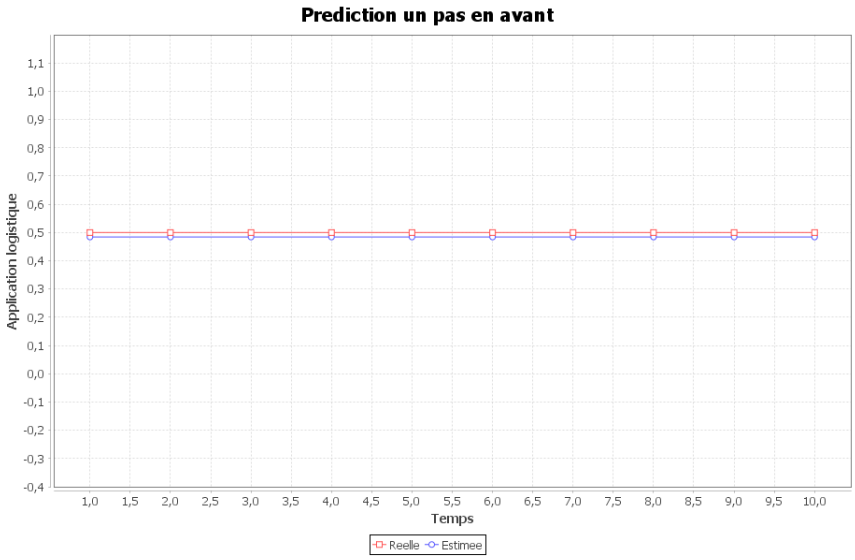


FIGURE 5.3 – Résultats des prédictions à un pas en avant $\mu = 2$

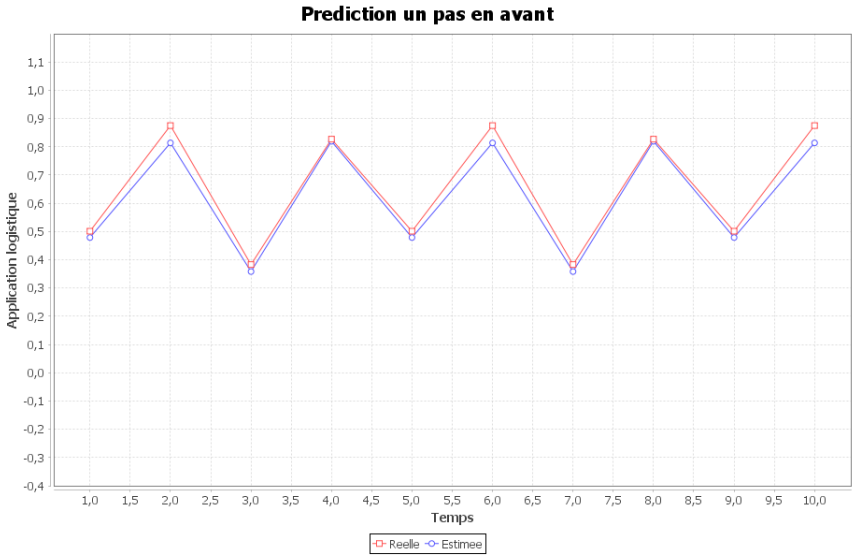


FIGURE 5.4 – Résultats des prédictions à un pas en avant $\mu = 3.5$

• Prédictions à plusieurs pas en avant

Pour chacun des cas, nous réaliserons 3 et 10 prédictions successives. On prendra la même prototype pour la prédiction c'est à dire le $300^{ème}$ prototype. Ce qui nous donne les prévisions de $x(301), x(302), x(303)$ et $x(301)$ à $x(310)$.

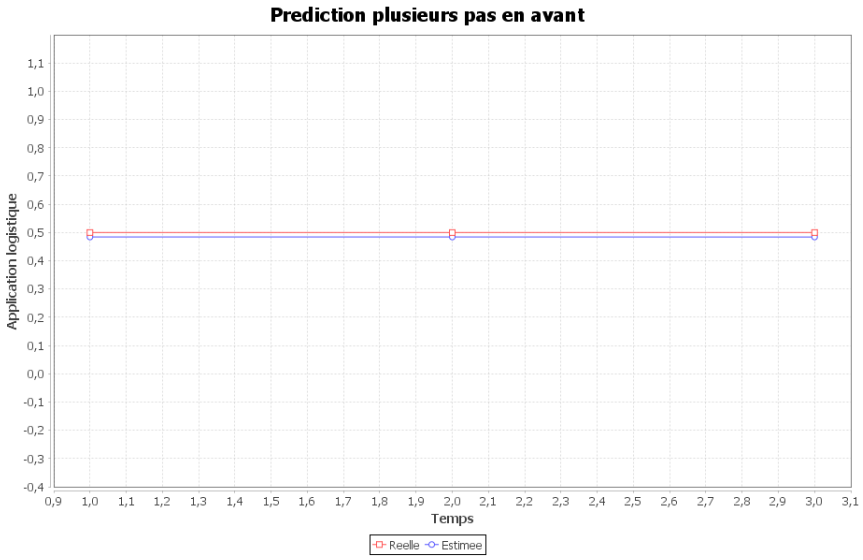


FIGURE 5.5 – Résultats des prédictions à trois pas en avant $\mu = 2$

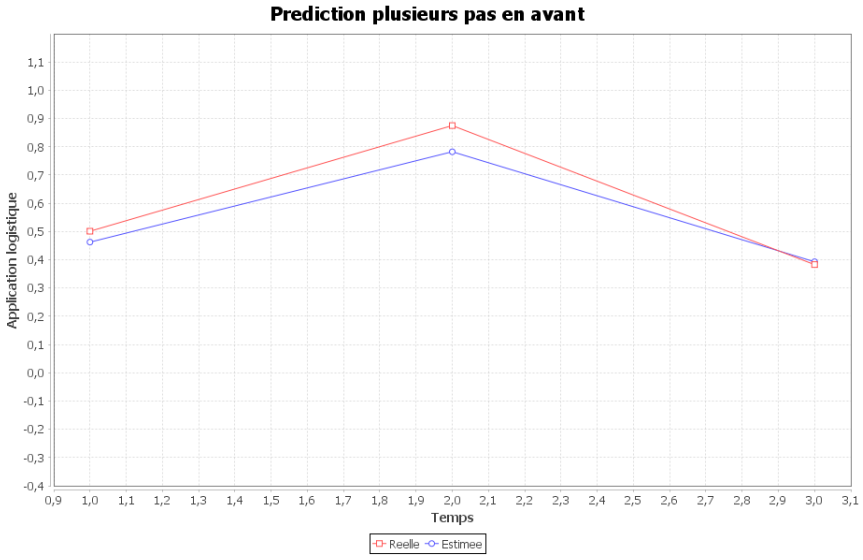


FIGURE 5.6 – Résultats des prédictions à trois pas en avant $\mu = 3.5$

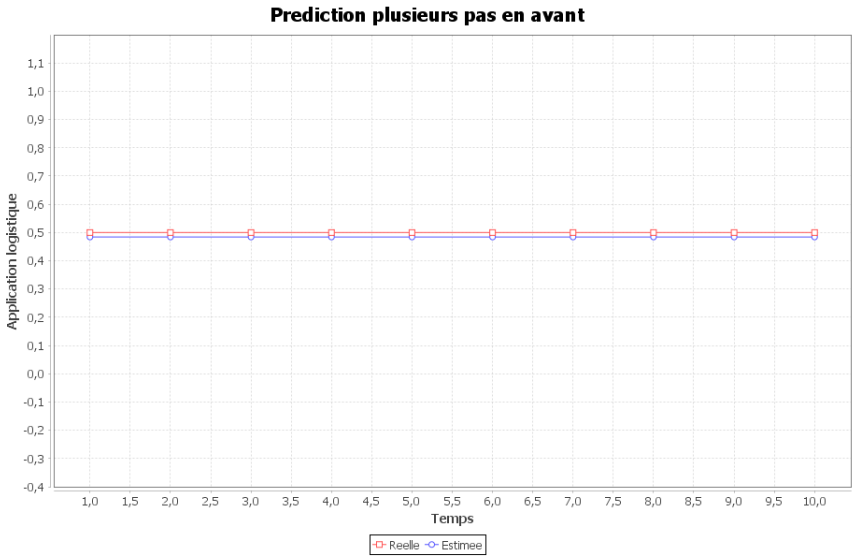


FIGURE 5.7 – Résultats des prédictions à dix pas en avant $\mu = 2$

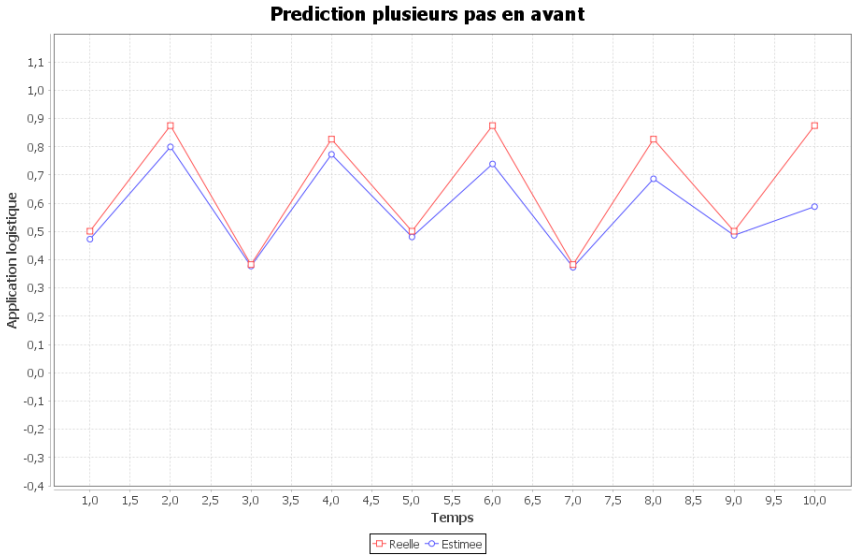


FIGURE 5.8 – Résultats des prédictions à dix pas en avant $\mu = 3.5$

L'erreur de prédiction moyenne pour chacun des quatre cas est donné par le tableau suivant

	Erreur de prédiction moyenne	
Nombre d'itérations	$\mu = 2$	$\mu = 3,5$
3	5.031679910^{-4}	0.00267200
10	9.128651810^{-4}	0.00326878

TABLE 5.2 – Erreurs de prédictions moyenne pour $\mu = 2$ et $\mu = 3.5$

Nous remarquons que l'erreur de prédiction moyenne augmente si le nombre d'itérations augmente.

5.3.2 Séries temporelles chaotiques $\mu = 4$

a) Architecture optimale du réseau

Le choix de l'architecture optimale du réseau est très difficile dans ce cas car le système adopte un comportement chaotique. Notre algorithme de Takens donne un nombre d'unité d'entrée égal à deux et d'après les tests que nous avons fait, nous avons l'architecture optimale suivante :

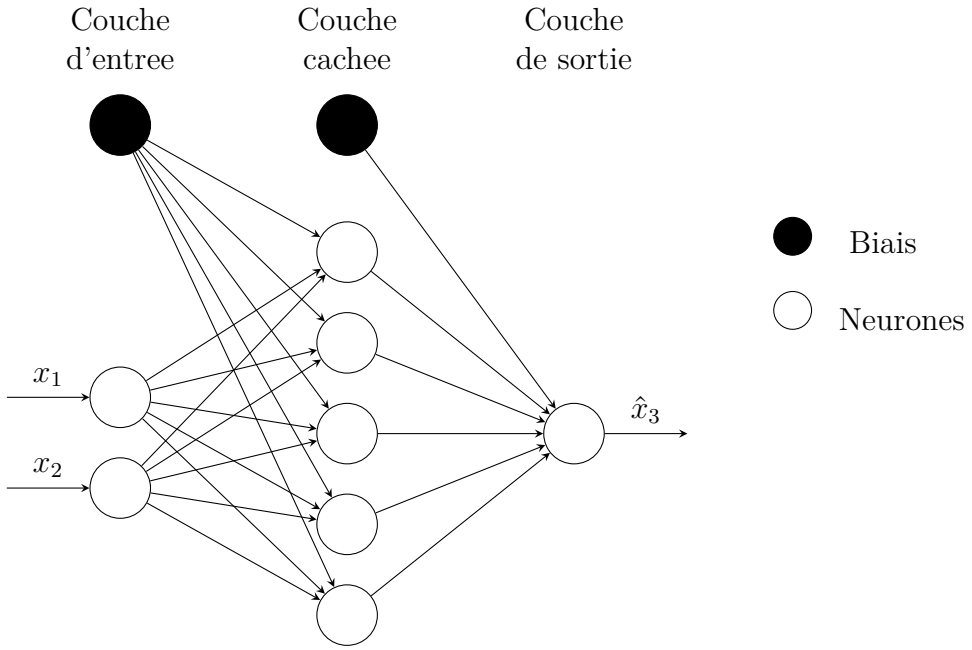


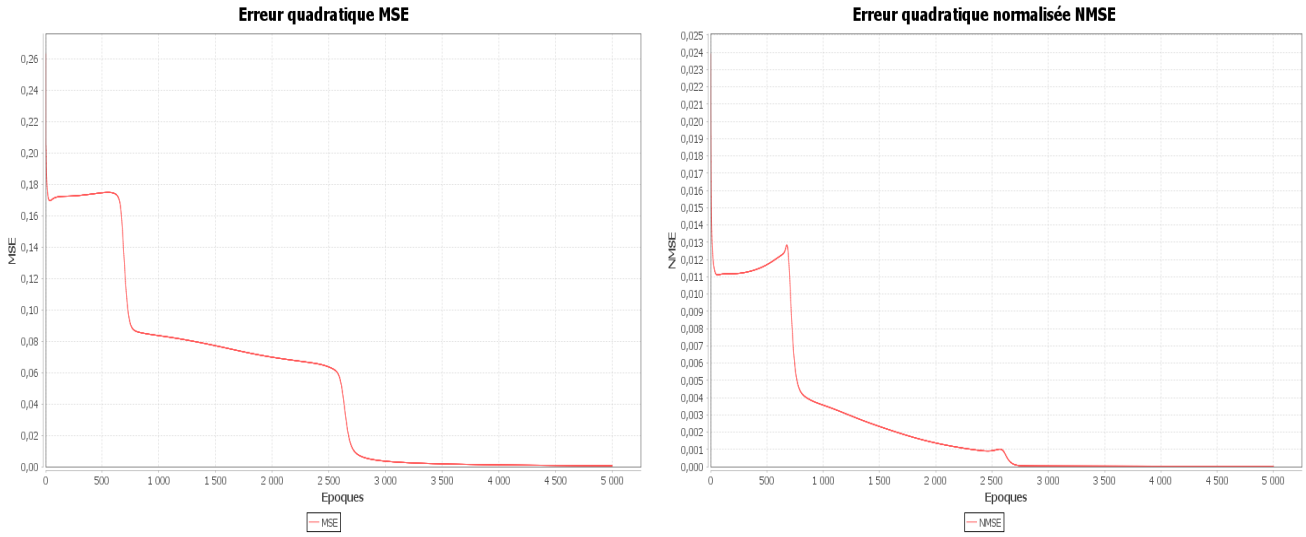
FIGURE 5.9 – Architecture optimale du réseau pour la série chaotique $\mu = 4$

b) Apprentissage

Après plusieurs testes, nous avons trouvé la configuration pour apprendre le réseau :

	$\mu = 4$
Pas de l'apprentissage	0.2
Momentum	0.5
Paramètre de régularisation	0.0000001
Nombre d'époque maximal	5000

TABLE 5.3 – Configurations de l'apprentissage du perceptron pour $\mu = 4$



(a) (b)

FIGURE 5.10 – Evolution de l'MSE (a) et NMSE (b) durant l'apprentissage du réseau $\mu = 4$

c) Résultats des prédictions

• Prédiction à un pas en avant

Nous avons choisi de faire 10 prédictions pour 10 valeurs existantes. Pour ce faire, nous prenons le $300^{ème}$ prototype, ce qui nous donne la prédiction de la $301^{ème}$ valeur de la série.

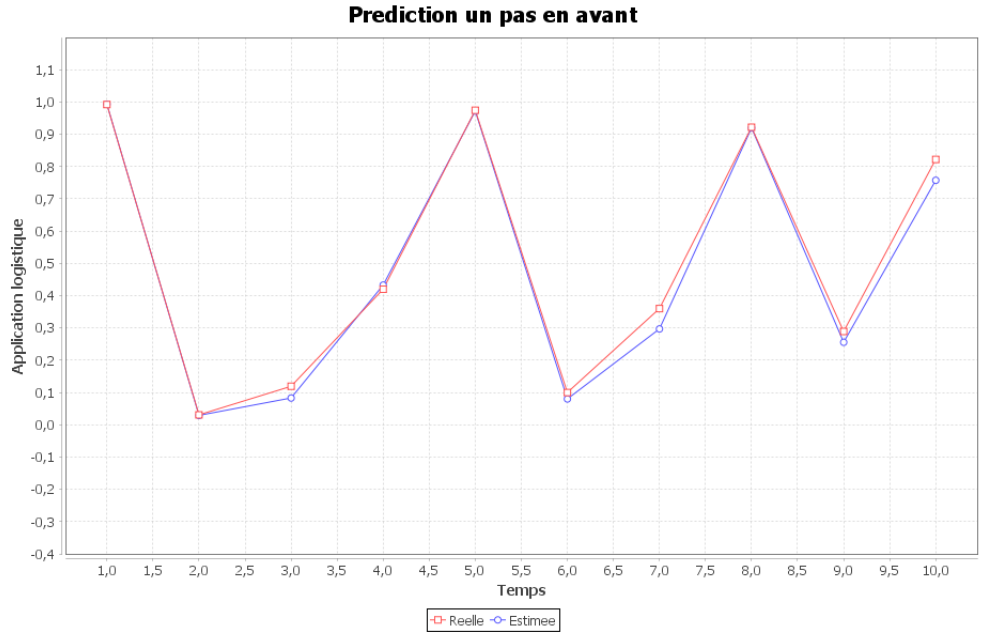


FIGURE 5.11 – Résultats des prédictions à un pas en avant $\mu = 4$

• Prédiction à plusieurs pas en avant

Dans ce cas, nous allons aussi prédire les valeurs de la série commençant par $x(301)$.

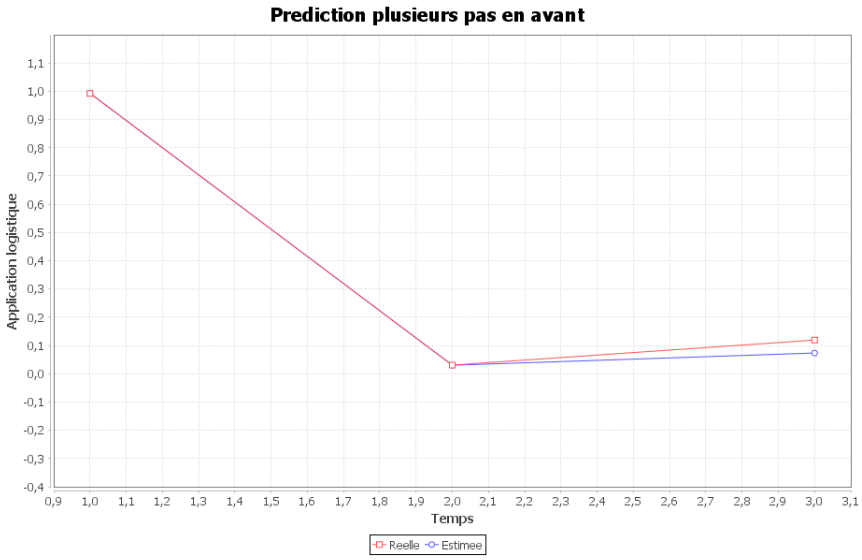


FIGURE 5.12 – Résultats des prédictions à trois pas en avant $\mu = 4$

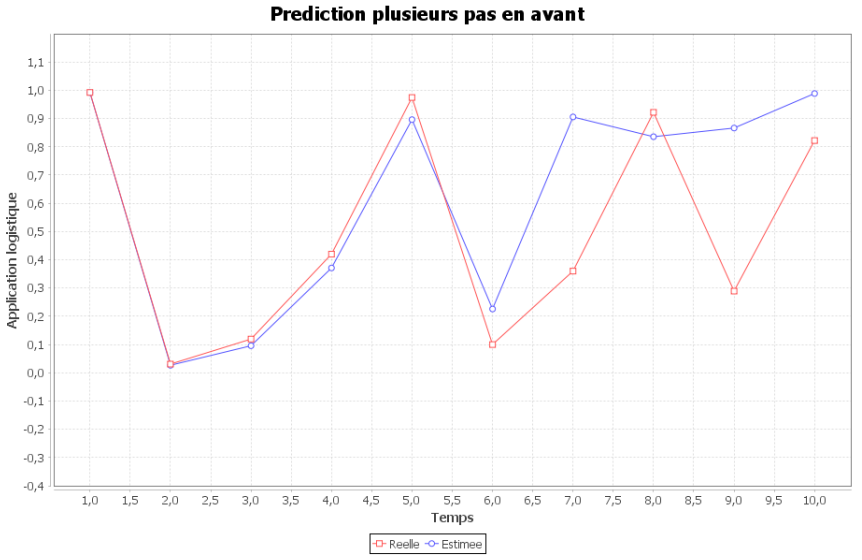


FIGURE 5.13 – Résultats des prédictions à dix pas en avant $\mu = 4$

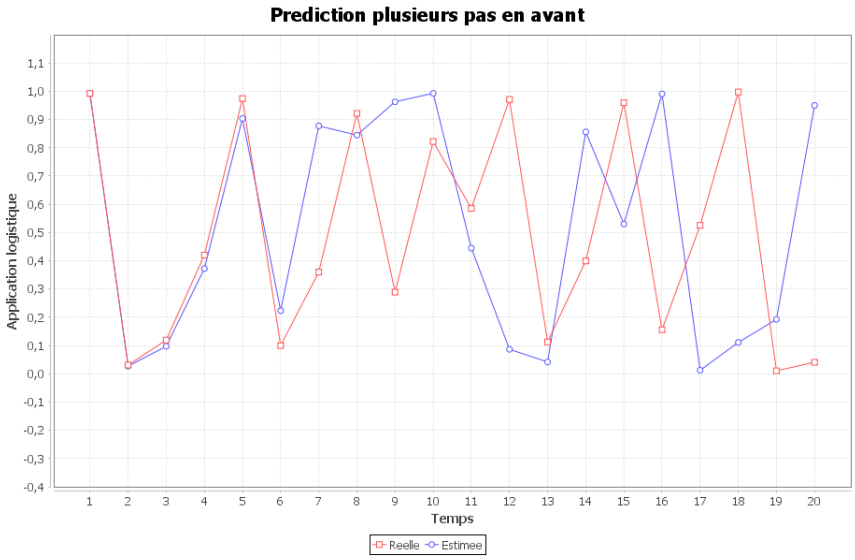


FIGURE 5.14 – Résultats des prédictions à vingt pas en avant $\mu = 4$

Les erreurs de prédiction moyennes pour les trois cas précédents sont :

	Erreur de prédiction moyenne
Nombre d'itérations	$\mu = 4$
3	8.1840649510^{-4}
10	0.01907589
20	0.03137464

TABLE 5.4 – Erreurs de prédictions moyennes pour $\mu = 4$

Conclusion

Ce chapitre a pour but de présenter les résultats des prédictions des trois séries temporelles générées par l'application logistique. Nous avons procédé à deux types de prédictions, la prédiction à un pas en avant ou prédiction à court terme et la prédiction à plusieurs pas en avant ou prédiction à long terme.

Dans le cas d'une prédiction à court terme, le résultat de la prédiction ne dépend pas du nombre de prédictions réalisées. Les valeurs estimées sont proches des valeurs réelles même dans le cas chaotique. Par contre, pour la prédiction à long terme, les résultats dépendent du nombre de pas. Si la série est non chaotique la prédiction est toujours bonne, mais si la série est chaotique, la prédiction à long terme présente une limite. La valeur de l'erreur de prédiction moyenne augmente avec le nombre d'itérations. La prévision est bonne jusqu'à un certain nombre d'itérations. Dans notre cas, nous observons que lorsque le nombre de pas dépasse six l'erreur devient grande et la valeur estimée diverge, ce qui est le signe du chaos.

Conclusion générale

Doté de nombreuses capacités intelligentes, les réseaux de neurones artificiels sont devenus un outil de prédiction puissant et mérite d'être exploité. Après une courte historique et théorie de base sur les réseaux de neurones, nous avons pu les exploiter pour résoudre le problème de prédiction des séries temporelles.

La contribution de nos travaux porte sur plusieurs aspects. Il s'agit en premier lieu de développer un modèle prédictif basé sur les réseaux de neurones artificiels à propagation de l'information vers l'avant ou les perceptrons multicouches. Ce fut l'objet du deuxième chapitre, qui traite d'une façon simple la création des réseaux de neurones utilisant le langage de programmation Java et l'environnement de développement NetBeans 8.1. Nous avons ensuite développé une interface de simulation qui regroupe les différents paquets de codes sources utilisés pour exploiter notre réseau et qui sert à représenter les résultats obtenus par les simulations.

Le deuxième point a porté sur l'application logistique et la théorie du chaos dans le but d'étudier tous les comportements que peut avoir cette application. En effet, le comportement de l'application logistique dépend d'un seul paramètre noté μ. En fonction de ce paramètre, les séries temporelles générées peuvent être chaotiques on non. L'étude des bifurcations nous montre les principaux points critiques et les différents comportements du système. Les exposants de Lyapounov nous donnent aussi la sensibilité de l'application à la condition initiale.

Le troisième aspect de nos travaux concerne la prédiction des séries temporelles générées par notre application logistique utilisant les perceptrons multicouches. Utilisant deux méthodes de prédiction : la prédiction à court terme et la prédiction à long terme, nous avons montré que les perceptrons multicouches peuvent appréhender le problème de prédiction des séries temporelles même dans le cas chaotique. Nous avons démontré que notre perceptron peut estimer les valeurs de la séries chaotique ou non chaotique, à 10^{-2} près utilisant la prédiction à court terme. Par contre, si nous utilisons la méthode de prédiction à long terme, notre perceptron présente des limites. En effet, si la série utilisée est chaotique, à partir d'un certain nombre d'itérations, les valeurs estimées divergent des valeurs désirées. C'est pour cela que nous avons calculé l'erreur de prédiction moyenne pour chaque prédiction à long terme fait. Nous avons montré que cette erreur augmente avec le nombre d'itérations.

Finalement, les résultats obtenus dépendent généralement de l'architecture de notre perceptron. Nous avons utilisé l'algorithme de Takens pour trouver le nombre de neurones de la couche d'entrée. Quant au nombre de neurones sur la couche cachée, aucune théorie n'est aujourd'hui fiable pour le définir. Nous souhaitons donc trouver un algorithme capable de résoudre ce problème. De plus plusieurs perspectives peuvent être envisagées

pour notre travail. Nous aurons pu comparer nos résultats avec ceux obtenus avec d'autres architectures comme le réseau d'Elman, de Jordan ou bien le « deep learning ».

Bibliographie

[1] W.S. McCulloch and W. Pitts, *A logical calculus of the ideas immanent in nervous activity*,Bulletin of Mathematical Biology , 5(4) :115–133, 1986.

[2] W.S. McCulloch and W. Pitts, *How we know universals the perception of auditory and visual forms*, Bulletin of Mathematical Biology 9(3) :127–147, 1989.

[3] Donald O. Hebb, *The Organization of Behavior : A Neuropsychological Theory* ,Wiley New York,1949.

[4] Frank Rosenblatt,*The perceptron : a probabilistic model for information storage and organization in the brain* Psychological review, 65(6) :386,1958

[5] B Widrow and M E Hoff *Adaptive switching circuits*, In Proceedings WESCON , pages 96–104, 1960.

[6] M. Minsky and S. Papert. *Perceptrons* . MIT Press, Cambridge, Mass, 1969.

[7] P. J. Werbos. *Beyond Regression : New Tools for Prediction and Analysis* in the Behavioral Sciences . PhD thesis, Harvard University, 1974.

[8] T. Kohonen.*Correlation matrix memories*. IEEEtC , C-21 :353–359, 1972.

[9] S. Grossberg.*Adaptive pattern classification and universal recoding, I Parallel development and coding of neural feature detectors* Biological Cybernetics , 23 :121–134, 1976.

[10] Teuvo Kohonen. *Self-organized formation of topologically correct feature maps*. Biological Cybernetics , 43 :59–69, 1982.

[11] Teuvo Kohonen. *Self-Organization and Associative Memory* . Springer-Verlag, Berlin, third edition, 1989.

[12] John J. Hopfield. *Neural networks and physical systems with emergent collective computational abilities*. Proc. of the National Academy of Science, USA , 79 :2554–2558, 1982.

[13] Haykin s *neural networks a comprehensive foundation*,Prentice Hall International, Inc. 1994

[14] Haykin s *neural networks and machine learning*,Prentice Hall International,Third edition, 2008.

[15] D. Anguita, G. Parodi, and R. Zunino. *Speed improvement of the back-propagation on current-generation workstations*. In WCNN'93, Portland :World Congress on Neural Networks, July 11-15, 1993, Oregon Convention Center, Portland, Oregon , volume 1. Lawrence Erlbaum, 1993.

[16] G. Cybenko. *Approximation by superpositions of a sigmoidal function*. Mathematics of Control, Signals, and Systems (MCSS) , 2(4) :303–314,1989.

[17] Jeffrey L. Elman. *Finding structure in time*. Cognitive Science , 14(2) :179–211, April 1990.

[18] M. I. Jordan.*Attractor dynamics and parallelism in a connectionist sequential machine*. In Proceedings of the Eighth Conference of the Cognitive Science Society , pages 531–546. Erlbaum, 1986.

[19] F. Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1962.

[20] J. L. McClelland and D. E. Rumelhart. *Parallel Distributed Processing* :Explorations in the Microstructure of Cognition , volume 2. MIT Press, Cambridge, 1986.

[21] David E. Rumelhart, Geoffrey E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*. In D. E. Rumelhart, J. L. McClelland, and the PDP research group., editors, Parallel distributed processing : Explorations in the microstructure of cognition, Volume 1 : Foundations . MIT Press, 1986.

[22] Jeff Heaton, *Introduction to Neural Networks with Java*, Second Edition, Heaton Research Inc. ISBN : 1-60439-008-5

[23] Herbert Schildt, *Java The Complete Reference*, Eighth Edition, Oracle Press, 2011.

[24] Fábio M Soares and Alan M.F Souza *Neural Network Programming with Java*, PACKT publishing, 2016

[25] EDWARD N. LORENZ *The Essence of Chaos*, Paperback – March 30, 1995

[26] Edward N. Lorenz, *Deterministic non-periodic flow*, Journal of the Atmospheric Sciences 20(2) 130–141, 1963.

[27] Ohtsubo J *Semiconductor Lasers Stability, Instability ans Chaos*, Springer, 2008.

[28] Cushing, J. M., R. F. Costantino, Brian Dennis, Robert A. Desharnais, and Shandelle M. Henson. 2003. Chaos in ecology. Vol. 1, Experimental nonlinear dynamics. Boston : Academic Press.

[29] Richard J Field and László Györgyi *Chaos in Chemistry and Biochemistry*,Mar 1993.

[30] D. Ruelle and F. Takens *On the Nature of turbulence*, Communications in Mathematical Physics, 20, 167-192, 1971.

[31] Jacques M. Bahi,Christophe Guyeux *Discrete Dynamical Systems and Chaotic Machines : Theory and Applications*, CRC Press, 2013.

[32] Pierre-François Verhulst *La première découverte de la fonction logistique*, 1804-1849.

[33] Benoit B. Mandelbrot, *The Fractal Geometry of Nature*, Henry Holt and Company, 15 août 1982.

[34] Chatfield, C. TIME-SERIES FORECASTING, Chapman and Hall CRC, United States of America, 2000.

[35] Régis Bourdonnais and Michel T, *Analyse des séries temporelles : Applications à l'économe et à la gestion*, 2ème édition, DUNOD, 2008.

[36] Coulibaly, P., Anctil, F. et Bobée, B. Prévision hydrologique par réseaux de neurones artificiels, état de l'art. Can. J. Civ. Eng. Vol. 26, pp. 293–304, 1999.

[37] Kayacan, E., Ulutas, B. et Kaynak, O. Grey system theory-based models in time series prediction, Expert Systems with Applications, Vol 37, pp. 1784–178, 2010.

[38] Kandel, A. Fuzzy expert systems. Florida, USA : CRC Press, 1991.

[39] Ma, J. and Teng, J.F. Predict chaotic time-series using unscented Kalman filter. In Proceedings of the third international conference on machine learning and cybernetics, Shanghai, China, Vol. 1, pp. 867- 890, 2004.

[40] Chen, B.J., Chang M.W. et Lin, C.J. Member, IEEE, Load Forecasting Using Support Vector Machines : A Study on EUNITE Competition 2001, IEEE TRANSACTIONS ON POWER SYSTEMS, VOL. 19, NO. 4, pp. 1821- 1830, NOVEMBER, 2004.

[41] Takens F, Detecting strange attractors in turbulence. *Springer Lecture Notes in Mathematics* 898 366-381, 1981.

[42] Bjarne Stroustrup, The Design an Evolution of C++, ADDISON WESLEY PUBLISHING COMPANY, 1994.

Annexe A

Codes sources

A.1 Classe série temporelle

```
1
2  package Outils;
3  import java.io.File;
4  import java.io.FileWriter;
5  import java.io.IOException;
6  import com.csvreader.CsvWriter;
7  import Outils.Takens;
8  import Outils.Jacobi;
9  import Outils.Matrix;
10
11
12 /**
13  *
14  * @author Manantena Kiady
15  */
16 public class Series_temporelles {
17
18     double [] X;
19     int dim;
20     double param;
21     int indicePrototype;
22     int nbUniteEntree;
23     private int Prototype_max;
24 public Series_temporelles(int d, double p, int pm){
25         dim = d;
26         param = p;
27         X = new double[dim + 1];
28         X[0] = 0.1;
29         for(int n =0; n < dim; n++)
30         X[n+1] = param*X[n]*(1 - X[n]);
31         Prototype_max = pm;
32         calcNombreUntitEntree();
33         CsvApprentissage();
34     }
35   static String toString(double d){
36          String retour = "";
37          retour += d;
```

```
38          return retour;
39      }
40    public void CsvApprentissage(){
41          int nbUniteEntreeSerie =this.getNbUniteEntreeSerie();
42          double[][] MatriceEntree = new double[Prototype_max + 1][
                nbUniteEntreeSerie + 3];
43          double[][] MatriceSortie = new double[Prototype_max + 1][Prototype_max
                + 1];
44          int nb = 0 ;
45          String data = "data/entreeApprentissage.csv";
46          String data1 = "data/sortieApprentissage.csv";
47          boolean existe = new File(data).exists();
48          boolean existe1 = new File(data1).exists();
49          try{
50              CsvWriter sortieCsv = new CsvWriter(new FileWriter(data1,false), ','
                    );
51              CsvWriter entreeCsv = new CsvWriter(new FileWriter(data,false), ',')
                    ;
52          for(int prototype_i = 0; prototype_i < Prototype_max; prototype_i++){
53                   for(int i = 1; i <= nbUniteEntreeSerie+1 ; i++){
54                              MatriceEntree[prototype_i][0] = 1; //----biais----
55                              MatriceEntree[prototype_i][i] = this.getX()[
                                    prototype_i+i - 1]; //[nb]
56                              entreeCsv.write(toString(MatriceEntree[prototype_i][
                                    i - 1]));
57                      }
58            entreeCsv.endRecord();
59          }
60          entreeCsv.close();
61
62          for(int i = 0; i < Prototype_max; i++)
63          {
64              MatriceSortie[i][0] = this.getX()[i+nbUniteEntreeSerie];
65              sortieCsv.write(toString(MatriceSortie[i][0]));
66              sortieCsv.endRecord();
67          }
68          sortieCsv.close();
69          }
70          catch (IOException e)
71          {
72            e.printStackTrace();
73          }
74       }
75    public void calcNombreUntitEntree(){
76          int d = 500;
77          double[] X = new double[d];
78          for(int i =0; i< d; i++){
79              X[i] = this.getX()[i];
80          }
81          Takens tks = new Takens(X.length,X);
82          Matrix m1 = new Matrix(X.length,X.length);
83          m1 = tks.vect_lig(X);
84          Matrix m2 = new Matrix(X.length,X.length);
```

```
        m2 = tks.vect_col(X);
        Matrix M = new Matrix(X.length,X.length);
        M = tks.prod_vec(m1, m2);
        Jacobi jcb = new Jacobi();
        double[] valprop = jcb.val_prop(M);
        double[] erreur = jcb.erreur_approx(valprop);
        nbUniteEntree = jcb.nbUniteEntree(erreur);
    }
    public int getNbUniteEntreeSerie()
    {
        return nbUniteEntree;
    }
    public double[] getX()
      {
        return X;
      }
    public double getParametre()
    {
        return param;
    }
    public void setParamtetre(double param)
    {
        this.param = param;
    }
    public int getDimension()
    {
        return dim;
    }
    public void setDimension(int dimension)
    {
        this.dim = dimension;
    }
    public int getIndicePrototype()
    {
        return indicePrototype;
    }
    public void setIndicePrototype(int indice)
    {
        this.indicePrototype = indice;
    }
      public int getPrototype_max() {
        return Prototype_max;
      }
      public void setPrototype_max(int Prototype_max) {
        this.Prototype_max = Prototype_max;
      }
}
```

A.2 Classe Prédiction

```
package Prediction;
```

```
2
3  import reseaux_neuronal_master.Reseaux;
4  import Apprentissage.Apprentissage;
5  import Data_Chart.Chart;
6  import Data_Chart.Data;
7  import Outils.Series_temporelles;
8  import Swing.Convert;
9  import com.csvreader.CsvWriter;
10 import java.awt.BorderLayout;
11 import java.awt.Color;
12 import java.awt.Font;
13 import java.io.FileWriter;
14 import java.io.IOException;
15 import java.util.ArrayList;
16 import javax.swing.JFrame;
17 import javax.swing.JPanel;
18 import javax.swing.JTable;
19 /*@author Manantena Kiady*/
20 public class Prediction extends Apprentissage{
21     //reseaux definie ici est seulement l'architecture
22     //sans entree ni sortie
23     //reseaux avec max_epochs, erreur,
24    private double[][] MatriceEntree;
25    private double[][] MatriceSortie;
26    private double[][] matriceEntree;
27    private double[] ValeursPredites;
28    private double INMSE;
29    private int prototype_max;
30    private JPanel p1,p2,p3,p4,p5,p6,p7,p8;
31    private JTable t1,t2,t3;
32    private ArrayList<Double> listErreurPredic ;
33
34    public Prediction(){
35         t1 = new JTable();
36         t3 = new JTable();
37         p1 = new JPanel();
38         p2 = new JPanel();
39         p3 = new JPanel();
40         p4 = new JPanel();
41         t2 = new JTable();
42         p6 = new JPanel();
43    }
44    public Reseaux Apprendre(Series_temporelles serie, Reseaux rna){
45            Reseaux rnaApp = new Reseaux();
46            Data entree = new Data("data", "entreeApprentissage.csv");
47            Data sortie = new Data("data", "sortieApprentissage.csv");
48            try{
49                double[][] matriceEntree = entree.rawData2Matrix(entree);
50                double[][] matriceSortie = sortie.rawData2Matrix(sortie);
51                rna.setMatriceEntree(matriceEntree);
52                rna.setMatriceReelleSortie(matriceSortie);
53                rnaApp = rna.apprendreReseaux(rna);
54                Chart c1 = new Chart();
```

```
55              Chart c4 = new Chart();
56              p1 =(c1.afficheDonneeXY(rna.getListMSE().toArray(), "Erreur
                    quadratique MSE", "Epoques", "MSE"));
57              p1.setVisible(true);
58              p2 = (c4.afficheDonneeXY(rna.getM_listNMSE().toArray(), "Erreur
                    quadratique normalisee NMSE", "Epoques", "NMSE"));
59              p2.setVisible(true);
60               JTable tab = new JTable();
61          String[] nam = {"Epochs","MSE","NMSE"};
62          Object[][] object1 = new Object[rna.getListMSE().size()][3];
63          Font font1 = new Font("",Font.BOLD,12);
64          for(int i = 0; i < rna.getListMSE().size(); i++)
65          { for(int j = 0; j < 3; j++)
66              {
67                  if(j==0)
68                  {
69                  object1[i][j] = Convert.toString(i+1);
70                  font1 = new Font(nam[j],Font.BOLD,12);
71                  }
72                  else if(j==1)
73                  {
74                  font1 = new Font(nam[j],Font.BOLD,12);
75                  object1[i][j] = Convert.toString(rna.getListMSE().get(i));
76                  }
77                  else
78                  {
79                  font1 = new Font(nam[j],Font.BOLD,12);
80                  object1[i][j] = Convert.toString(rna.getM_listNMSE().get(i));
81                  }
82                 tab = new JTable(object1,nam);
83              }
84          }
85          tab.getTableHeader().setBackground(Color.white);
86          tab.getTableHeader().setFont(font1);
87          tab.setGridColor(Color.LIGHT_GRAY);
88          tab.setBackground(Color.white);
89          t3=(tab);
90             double[][] sortieRna = rnaApp.getSortieReseau(rnaApp);
91             ArrayList<double[][]> listOfArraysToJoin = new ArrayList<double
                   [][]>();
92             listOfArraysToJoin.add( matriceSortie );
93             listOfArraysToJoin.add( sortieRna);
94             double[][] matrixOutputsJoined = new Data().joinArrays(
                   listOfArraysToJoin);
95             Chart c2 = new Chart();
96             setP3(c2.afficheDonneeXY(matrixOutputsJoined, "Apprentissage du
                   reseau", "Temps", "Application logistique", Chart.
                   TypeAffichageChartEnum.COMPARAISON));
97             JTable table = new JTable();
98          String[] names = {"N:","Valeurs Reseau","Valeurs reelles","Erreur"};
99          Object[][] object = new Object[sortieRna.length][4];
100         Font font = new Font("",Font.BOLD,12);
101         for(int i = 0; i < sortieRna.length; i++)
```

```
102         { for(int j = 0; j < 4; j++)
103             {
104                 if(j==0)
105                 {
106                 object[i][j] = Convert.toString(i+1);
107                 font = new Font(names[j],Font.BOLD,12);
108                 }
109                 else if(j==1)
110                 {
111                 font = new Font(names[j],Font.BOLD,12);
112                 object[i][j] = Convert.toString(sortieRna[i][0]);
113                 }
114                 else if(j==2)
115                 {
116                 font = new Font(names[j],Font.BOLD,12);
117                 object[i][j] = Convert.toString(matriceSortie[i][0]);
118                 }
119                 else
120                 {
121                 font = new Font(names[j],Font.BOLD,12);
122                 object[i][j] = Convert.toString(matriceSortie[i][0]-sortieRna[i
                        ][0]);
123                 }
124                table = new JTable(object,names);
125             }
126         }
127         table.getTableHeader().setBackground(Color.white);
128         table.getTableHeader().setFont(font);
129         table.setGridColor(Color.LIGHT_GRAY);
130         table.setBackground(Color.white);
131         t1=(table);
132            }
133            catch(IOException e){
134                e.printStackTrace();
135            }
136          return rnaApp;
137     }
138     public void predireUnPas(Reseaux r,int indicePrototype,Series_temporelles
        serie, int nb)
139     { listErreurPredic = new ArrayList<Double>();
140         double[][] sortieR = new double[nb][1];
141         double[][] sortieD = new double[nb][1];
142         p4 = new JPanel();
143          t2 = new JTable();
144          p6 = new JPanel();
145         String data = "data/entreeTestunpas.csv";
146         String data1 = "data/sortieTestunpas.csv";
147         int nbUniteEntree = r.getCoucheEntree().getNbNeuronSurCouche() - 1;
148         for(int k = 0; k < nb ; k++)
149         {
150         MatriceEntree = new double[1][nbUniteEntree + 2];
151         MatriceSortie = new double[1][1];
152         try{
```

```
153             CsvWriter sortieCsv = new CsvWriter(new FileWriter(data1,false), ','
                    );
154             CsvWriter entreeCsv = new CsvWriter(new FileWriter(data,false), ',')
                    ;
155             MatriceEntree[0][0] = 1; //----biais----
156          for(int i = 1; i <= nbUniteEntree + 1 ; i++){
157                             MatriceEntree[0][i] = serie.getX()[indicePrototype +
                                    i - 2 + k];
158                             entreeCsv.write(toString(MatriceEntree[0][i-1]));
159                     }
160         entreeCsv.endRecord();
161         entreeCsv.close();
162         for(int j = 0; j <r.getCoucheSortie().getNbNeuronSurCouche();j++){
163                      MatriceSortie[0][j] = serie.getX()[nbUniteEntree +
                             indicePrototype + k - 1];
164                      sortieCsv.write(toString(MatriceSortie[0][j]));
165                      sortieCsv.endRecord();
166              }
167         sortieCsv.close();
168         }
169         catch (IOException e)
170         {
171           e.printStackTrace();
172         }
173         Data entreetest = new Data("data", "entreeTestunpas.csv");
174         Data sortietest = new Data("data", "sortieTestunpas.csv");
175         try{
176          double[][] matriceEntree = sortietest.rawData2Matrix(entreetest);
177          double[][] matriceSortie = sortietest.rawData2Matrix(sortietest);
178
179         r.setMatriceEntree(matriceEntree);
180         r.setMatriceReelleSortie(matriceSortie);
181         double[][] sortieRna = r.getSortieReseau(r);
182         sortieR[k][0] = sortieRna[0][0];
183         sortieD[k][0] = matriceSortie[0][0];
184         System.out.print( matriceSortie[0][0]+" :----VS----: "+sortieRna[0][0]+
                "\t");
185         } catch (IOException e)
186         {
187           e.printStackTrace();
188         }
189         }
190         ArrayList<double[][]> listOfArraysToJoin = new ArrayList<double[][]>();
191               listOfArraysToJoin.add( sortieD );
192               listOfArraysToJoin.add( sortieR );
193               double[][] matrixOutputsJoined = new Data().joinArrays(
                        listOfArraysToJoin);
194               Chart c2 = new Chart();
195            p4=(c2.afficheDonneeXY(matrixOutputsJoined, "Prediction un pas en
                    avant", "Temps", "Application logistique", Chart.
                    TypeAffichageChartEnum.COMPARAISON));
196            JTable table = new JTable();
197         String[] names = {"N:","Valeurs predites","Valeurs reelles","Erreurs"};
```

```
198         Object[][] object = new Object[sortieR.length][4];
199         Font font = new Font("",Font.BOLD,12);
200         for(int i = 0; i < sortieR.length; i++)
201         { for(int j = 0; j < 4; j++)
202             {
203                 if(j==0)
204                 {
205                 object[i][j] = Convert.toString(i+1);
206                 font = new Font(names[j],Font.BOLD,12);
207                 }
208                 else if(j==1)
209                 {
210                 font = new Font(names[j],Font.BOLD,12);
211                 object[i][j] = Convert.toString(sortieR[i][0]);
212                 }
213                 else if(j==2)
214                 {
215                 font = new Font(names[j],Font.BOLD,12);
216                 object[i][j] = Convert.toString(sortieD[i][0]);
217                 }
218                 else
219                 {
220                double q = Math.abs(sortieD[i][0]-sortieR[i][0]);
221                 font = new Font(names[j],Font.BOLD,12);
222                 object[i][j] = Convert.toString(q);
223                 listErreurPredic.add(q);
224                 }
225                 table = new JTable(object,names);
226             }
227         }
228         table.getTableHeader().setBackground(Color.white);
229         table.getTableHeader().setFont(font);
230         table.setGridColor(Color.LIGHT_GRAY);
231         table.setBackground(Color.white);
232         Chart c3 = new Chart();
233         setP6(c3.afficheDonneeXY( listErreurPredic.toArray(), "Erreur
                prediction", "Nombre d'iteration", "Erreurs"));
234         t2=(table);
235     }
236      public void predirePlusieursPas(Reseaux r,int indicePrototype,
            Series_temporelles serie, int nbPas)
237     { listErreurPredic = new ArrayList<Double>();
238         double[][] sortieR = new double[nbPas][1];
239         double[][] sortieD = new double[nbPas][1];
240         p4 = new JPanel();
241         t2 = new JTable();
242           p6 = new JPanel();
243         String data = "data/entreeTestpluspasb2.csv";
244         String data1 = "data/sortieTestpluspasb2.csv";
245         int nbUniteEntree = r.getCoucheEntree().getNbNeuronSurCouche() - 1;
246         int k;
247         for(int pas_i = 0; pas_i < nbPas ; pas_i++)
248         {
```

```
249         MatriceEntree = new double[1][nbUniteEntree + 2];
250         MatriceSortie = new double[1][1];
251         try{
252             CsvWriter sortieCsv = new CsvWriter(new FileWriter(data1,false), ','
                );
253             CsvWriter entreeCsv = new CsvWriter(new FileWriter(data,false), ',')
                ;
254          if(pas_i==0)
255         {
256            for(int i = 0; i <= nbUniteEntree ; i++){
257                        if(i==0) MatriceEntree[0][0] = 1;//biais
258                        else MatriceEntree[0][i] = serie.getX()[indicePrototype +
                            i - 2 + pas_i];
259                        entreeCsv.write(toString(MatriceEntree[0][i]));
260                       }
261            entreeCsv.endRecord();
262            entreeCsv.close();
263         for(int j = 0; j <r.getCoucheSortie().getNbNeuronSurCouche();j++){
264                        MatriceSortie[0][j] = serie.getX()[nbUniteEntree +
                            indicePrototype + pas_i - 1];
265                        sortieCsv.write(toString(MatriceSortie[0][j]));
266                        sortieCsv.endRecord();
267               }
268         sortieCsv.close();
269         }
270          else
271         {
272          k = pas_i;
273          for(int i = nbUniteEntree; i >=0 ; i--){
274               if(i==0) MatriceEntree[0][0] = 1.0; //biais
275               else
276               {
277                  if(i<=nbUniteEntree - pas_i){
278                        MatriceEntree[0][i] = serie.getX()[indicePrototype +i - 2+
                             pas_i];
279                     }
280                 else{
281                        MatriceEntree[0][i] = sortieR[k-1][0];
282                     }
283               }
284                    k--;
285              }
286          for(int i =0; i <= nbUniteEntree; i++ )
287          {
288               entreeCsv.write(toString(MatriceEntree[0][i]));
289          }
290         entreeCsv.endRecord();
291         entreeCsv.close();
292         for(int j = 0; j <r.getCoucheSortie().getNbNeuronSurCouche();j++){
293                        MatriceSortie[0][j] = serie.getX()[nbUniteEntree +
                            indicePrototype + pas_i - 1];
294                        sortieCsv.write(toString(MatriceSortie[0][j]));
295                        sortieCsv.endRecord();
```

```
296             }
297         sortieCsv.close();
298         }
299         }
300         catch (IOException e)
301         {
302           e.printStackTrace();
303         }
304         Data entreetest = new Data("data", "entreeTestpluspasb2.csv");
305         Data sortietest = new Data("data", "sortieTestpluspasb2.csv");
306         try{
307          double[][] matriceEntree = sortietest.rawData2Matrix(entreetest);
308          double[][] matriceSortie = sortietest.rawData2Matrix(sortietest);
309          for(int j = 0; j<matriceEntree[0].length;j++)
310          {
311              System.out.println("Entree "+pas_i+"= "+matriceEntree[0][j]);
312          }
313         r.setMatriceEntree(matriceEntree);
314         r.setMatriceReelleSortie(matriceSortie);
315         double[][] sortieRna = r.getSortieReseau(r);
316         sortieR[pas_i][0] = sortieRna[0][0];
317         sortieD[pas_i][0] = matriceSortie[0][0];
318         System.out.print( matriceSortie[0][0]+" :----VS----: "+sortieRna[0][0]+
                "\t");
319
320         } catch (IOException e)
321         {
322           e.printStackTrace();
323         }
324         }
325         ArrayList<double[][]> listOfArraysToJoin = new ArrayList<double[][]>();
326               listOfArraysToJoin.add( sortieD );
327               listOfArraysToJoin.add( sortieR );
328
329               double[][] matrixOutputsJoined = new Data().joinArrays(
                    listOfArraysToJoin);
330               double[] erreur = new double[sortieD.length];
331               double moyenne = 0.0;
332               double variance =0.0;
333                 for(int i = 0; i < sortieD.length; i++)
334                  {
335                        moyenne = moyenne + sortieD[i][0];
336                        variance+= Math.pow(sortieD[i][0],2.0);
337
338                  }
339                    moyenne = moyenne / sortieD.length;
340                    variance = variance - Math.pow(moyenne, 2.0);
341               double som = 0.0;
342               for(int i = 0; i < sortieD.length;i++)
343               {
344                   erreur[i] = (sortieD[i][0] - sortieR[i][0]);
345                   som+=erreur[i]*erreur[i];
346               }
```

```
347             INMSE = som/(variance*nbPas);
348             Chart c2 = new Chart();
349             p4 = (c2.afficheDonneeXY(matrixOutputsJoined, "Prediction
                    plusieurs pas en avant", "Temps", "Application logistique",
                    Chart.TypeAffichageChartEnum.COMPARAISON));
350             System.out.println("INMSE = "+INMSE);
351               JTable table = new JTable();
352       String[] names = {"N:","Valeurs predites","Valeurs reelles","Erreurs"};
353       Object[][] object = new Object[sortieR.length][4];
354       Font font = new Font("",Font.BOLD,12);
355       for(int i = 0; i < sortieR.length; i++)
356       { for(int j = 0; j < 4; j++)
357           {
358               if(j==0)
359               {
360               object[i][j] = Convert.toString(i+1);
361               font = new Font(names[j],Font.BOLD,12);
362               }
363               else if(j==1)
364               {
365               font = new Font(names[j],Font.BOLD,12);
366               object[i][j] = Convert.toString(sortieR[i][0]);
367               }
368               else if(j==2)
369               {
370               font = new Font(names[j],Font.BOLD,12);
371               object[i][j] = Convert.toString(sortieD[i][0]);
372               }
373               else
374               {
375               double q = Math.abs(sortieD[i][0]-sortieR[i][0]);
376               font = new Font(names[j],Font.BOLD,12);
377               object[i][j] = Convert.toString(q);
378               listErreurPredic.add(q);
379               }
380               table = new JTable(object,names);
381           }
382       }
383       table.getTableHeader().setBackground(Color.white);
384       table.getTableHeader().setFont(font);
385       table.setGridColor(Color.LIGHT_GRAY);
386       table.setBackground(Color.white);
387       Chart c3 = new Chart();
388       p6=(c3.afficheDonneeXY( listErreurPredic.toArray(), "Erreur prediction"
              , "Nombre d'iteration", "Erreurs"));
389       t2=(table);
390   }
391     static String toString(double d){
392        String retour = "";
393        retour += d;
394        return retour;
395   }
396   public double[][] getMatriceEntree() {
```

```
397         return MatriceEntree;
398     }
399     public void show(){
400     }
401     public void setMatriceEntree(double[][] MatriceEntree) {
402         this.MatriceEntree = MatriceEntree;
403     }
404     public double[][] getMatriceSortie() {
405         return MatriceSortie;
406     }
407     public void setMatriceSortie(double[][] MatriceSortie) {
408         this.MatriceSortie = MatriceSortie;
409     }
410     public double[] getValeursPredites() {
411         return ValeursPredites;
412     }
413     public void setValeursPredites(double[] ValeursPredites) {
414         this.ValeursPredites = ValeursPredites;
415     }
416     public double getINMSE() {
417         return INMSE;
418     }
419     public void setINMSE(double INMSE) {
420         this.INMSE = INMSE;
421     }
422     public int getPrototype_max() {
423         return prototype_max;
424     }
425     public void setPrototype_max(int prototype_max) {
426         this.prototype_max = prototype_max;
427     }
428 }
```

A.3 Classe Jacobi

```
1  package Outils;
2
3  /* @author Manantena Kiady*/
4  public class Jacobi {
5      public Jacobi(){}
6      //tri decroissante
7      private double[] dec_sort(double[] tab){
8          double tmp;
9          for(int i=0; i<tab.length;i++)
10             for(int j = 0; j < tab.length; j++)
11                 if(tab[i] < tab [j]){
12                     tmp = tab[i];
13                     tab[i] = tab[j];
14                     tab[j] = tmp;
15                 }
16         return tab;
```

```
17      }
18      //cosinus et sinus
19      private double cos(double a, double b){
20          return (Math.sqrt(0.5*(1 + (b / Math.sqrt(a*a + b*b)))));
21      }
22      private double sin(double a, double b){
23          return (a/(2 * cos(a,b)*Math.sqrt(a*a + b*b)));
24      }
25       //diagonalization
26      public Matrix diagonalization(Matrix m){
27          double c, d;
28         // Matrix mat = new Matrix(m.getNbLignes(),m.getNbColonnes());
29          int lmax = m.max_ligne(m);
30          System.out.println("ligne max = " + lmax);
31          int cmax = m.max_colonne(m);
32          System.out.println("colonne max = " + cmax);
33          Matrix P = new Matrix(m.getNbColonnes(),m.getNbColonnes()).MatriceUnit(
                m.getNbColonnes());
34          double a =0, b =0;
35          a = 2*m.getValeur(lmax, cmax);
36          b = m.getValeur(lmax, lmax) - m.getValeur(cmax, cmax);
37          if(m.getValeur(lmax, lmax)!= m.getValeur(cmax, cmax)){
38              d = sin(a,b);
39              c = cos(a,b);
40          }else{
41              c = Math.sqrt(2)/2;
42              d = Math.sqrt(2)/2;
43          }
44          for(int i =0; i<P.getNbLignes(); i++){
45              for(int j =0; j<P.getNbColonnes(); j++){
46                  if((i==j)&&(j!=lmax)&&(j!=cmax))
47                      P.setValeur(i, j, 1);
48                  else if((i==cmax)&&(j==lmax))
49                      P.setValeur(i, j, -d);
50                  else if((i==lmax)&&(j==lmax) || (i==cmax)&&(j==cmax))
51                      P.setValeur(i, j, c);
52                  else if((i==lmax)&&(j==cmax))
53                      P.setValeur(i, j, d);
54                  else P.setValeur(i, j,0);
55              }
56          }
57          Matrix trans_P = new Matrix(P.transpose());
58          Matrix M = new Matrix(m.getNbLignes(),m.getNbColonnes());
59          M = M.multiplie(M.multiplie(trans_P,m),P);
60          return M;
61      }
62      public double [] val_prop(Matrix mat){
63          double [] valprop = new double[mat.getNbLignes()];
64          Matrix matrix = new Matrix(mat.getNbLignes(), mat.getNbColonnes());
65          Jacobi jc = new Jacobi();
66          matrix = jc.diagonalization(mat);
67          for(int i = 0; i<matrix.getNbLignes(); i++)
68          for(int j = 0; j<matrix.getNbColonnes(); j++)
```

```
69          if(i==j){
70              valprop[i] = matrix.getValeur(i, i);
71          }
72          return dec_sort(valprop);
73      }
74      public double[] erreur_approx(double [] mat){
75          double[] lambda = new double [mat.length];
76          for(int i = 0; i < mat.length; i++)
77          {lambda[i] = Math.sqrt(mat[i] + 1);
78          }
79         return lambda;
80      }
81      public int nbUniteEntree(double[] lambda){
82          int count = 1;
83          double erreur, c1,c2;
84          c1 = lambda[0];
85          c2 = lambda[1];
86          for(int i = 2; i < lambda.length; i++)
87          {
88           erreur = Math.abs(c2 - c1);
89              if((erreur <= 0.00000001)&&(erreur > 0.00000000001)){
90                  count = i ;
91                  break;
92              }else{
93                  c1 = c2;
94                  c2 = lambda[i];
95              }
96          }
97          return count;
98      }
99  }
```

A.4 Classe Matrix

```
1   package Outils;
2   import java.util.HashSet;
3   import java.util.Set;
4   /** @author Manantena Kiady */
5   public class Matrix {
6       private double [][] m_contenue;
7       private int m_nbLignes;
8       private int m_nbColonnes;
9       public Matrix(int lign, int col)
10      {
11          m_nbLignes = lign;
12          m_nbColonnes = col;
13          m_contenue = new double[m_nbLignes][m_nbColonnes];
14      }
15      public Matrix(double[][] mat)
16      {
17          m_nbLignes = mat.length;
```

```
18          m_nbColonnes = mat[0].length;
19          m_contenue = mat;
20      }
21      public Matrix(double[] vecteur)
22      {
23          m_nbLignes = 1;
24          m_nbColonnes = vecteur.length;
25          m_contenue = new double[m_nbLignes][m_nbColonnes];
26          m_contenue[0] = vecteur;
27      }
28      public Matrix(Matrix a)
29      {
30          m_nbLignes = a.getNbLignes();
31          m_nbColonnes = a.getNbColonnes();
32          m_contenue = new double[m_nbLignes][m_nbColonnes];
33          for(int i=0; i < m_nbLignes; i++)
34              for(int j=0; j< m_nbColonnes; j++)
35              {
36                  setValeur(i,j,a.getValeur(i, j));
37              }
38      }
39      public Matrix multiplie(Matrix a, Matrix b)
40      {
41          Matrix resultat = new Matrix(a.getNbLignes(), b.getNbColonnes());
42          if(a.getNbColonnes()!=b.getNbLignes())
43               throw new IllegalArgumentException("On ne pas multipier les deux
                     matrices");
44
45          for(int i=0;i<a.getNbLignes();i++){
46              for(int j=0;j<b.getNbColonnes();j++){
47                  double value = 0;
48                  for(int k=0;k<b.getNbLignes();k++){
49                      value+=a.getValeur(i,k)*b.getValeur(k,j);
50                  }
51                  resultat.setValeur(i, j, value);
52              }
53          }
54          return resultat;
55      }
56
57         public Matrix multiplie(Matrix a, double b){
58          Matrix resultat = new Matrix(a.getNbLignes(),a.getNbColonnes());
59
60          for(int i=0;i<a.getNbLignes();i++){
61              for(int j=0;j<a.getNbColonnes();j++){
62                  resultat.setValeur(i, j, a.getValeur(i,j)*b);
63              }
64          }
65
66          return resultat;
67      }
68       public Matrix transpose(){
69          Matrix resultat = new Matrix(m_nbColonnes,m_nbLignes);
```

```
70          for(int i=0;i<m_nbLignes;i++){
71              for(int j=0;j<m_nbColonnes;j++){
72                  resultat.setValeur(j, i, getValeur(i,j));
73              }
74          }
75          return resultat;
76      }
77
78      public Matrix transpose(Matrix a){
79          Matrix resultat = new Matrix(a.getNbColonnes(),a.getNbLignes());
80          for(int i=0;i<a.getNbLignes();i++){
81              for(int j=0;j<a.getNbColonnes();j++){
82                  resultat.setValeur(j, i, a.getValeur(i,j));
83              }
84          }
85          return resultat;
86      }
87      public Matrix MatriceId(int dim){
88       m_nbLignes = dim;
89       m_nbColonnes = dim;
90       Matrix Id = new Matrix(dim,dim);
91          for(int i=0;i<m_nbLignes;i++)
92              for(int j=0;j< m_nbColonnes;j++)
93                  Id.setValeur(i,j,(i==j)?1:0);
94
95          return Id;
96      }
97      //-------------matrice unitaire----------------------
98        public Matrix MatriceUnit(int dim){
99       m_nbLignes = dim;
100      m_nbColonnes = dim;
101      Matrix Unit = new Matrix(dim,dim);
102         for(int i=0;i<m_nbLignes;i++)
103             for(int j=0;j< m_nbColonnes;j++)
104                 Unit.setValeur(i,j,1.0);
105         return Unit;
106     }
107     //-------------max ligne et max colonne--------------
108      public int max_ligne(Matrix m)
109      {
110          double max = m.getValeur(0,1);
111          int r_max = 0;
112          for(int i =0; i<m.getNbLignes(); i++)
113              for(int j=0; j<m.getNbColonnes();j++)
114                  if(i!=j){
115                      if(m.getValeur(i, j) >= max){
116                          max = m.getValeur(i, j);
117                          r_max = i;
118                      }
119                  }
120          return r_max;
121      }
122       public int max_colonne(Matrix m)
```

```
123      {
124          double max = m.getValeur(0,1);
125          int c_max = 0;
126          for(int i =0; i<m.getNbLignes(); i++)
127              for(int j=0; j<m.getNbColonnes();j++)
128                  if(i!=j){
129                      if(m.getValeur(i, j) >= max){
130                          max = m.getValeur(i, j);
131                          c_max = j;
132                      }
133                  }
134          return c_max;
135      }
136      public double[][] getContenue(){
137          return m_contenue;
138      }
139      public void setContenue(double[][] ct){
140          this.m_contenue = ct;
141      }
142      public int getNbLignes(){
143          return m_nbLignes;
144      }
145      public void setNbLignes(int lgn){
146          this.m_nbLignes = lgn;
147      }
148      public int getNbColonnes(){
149          return m_nbColonnes;
150      }
151      public void setNbColonnes(int col){
152          this.m_nbColonnes = col;
153      }
154      public double getValeur(int i,int j){
155          if(i>=m_nbLignes)
156              throw new IllegalArgumentException("Nombre de lignes en dehors de la
                     limite");
157          if(j>=m_nbColonnes)
158              throw new IllegalArgumentException("Nombre de colonnes en dehors de
                    la limite");
159
160          return m_contenue[i][j];
161      }
162      public void setValeur(int i,int j,double value){
163          if(i>=m_nbLignes)
164              throw new IllegalArgumentException("Nombre de lignes en dehors de la
                     limite");
165          if(j>=m_nbColonnes)
166              throw new IllegalArgumentException("Nombre de colonnes en dehors de
                    la limite");
167
168          m_contenue[i][j]=value;
169      }
170
171  }
```

A.5 Classe Chart

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Data_Chart;

//----------------import-----------------
import java.awt.BasicStroke;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import javax.swing.JPanel;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;
import org.jfree.ui.RefineryUtilities;
/** @author Manantena Kiady*/
public class Chart {
    public enum TypeAffichageChartEnum{
        DONNEE_COMPLET, COMPARAISON;
    }
    public JPanel afficheDonneeXY(Object[] vector, String titre, String
        titreAxeX,String titreAxeY)
    {
        int longueur = vector.length;
        XYSeriesCollection donnee = new XYSeriesCollection();
        XYSeries series = new XYSeries(titreAxeY);
        for(int i = 0; i < longueur; i++){
            series.add((i+1), (double) vector[i]);
        }
        donnee.addSeries(series);
        JFreeChart chart = ChartFactory.createXYLineChart(
                    titre ,
                    titreAxeX,
                    titreAxeY,
                    donnee,
                    PlotOrientation.VERTICAL,
                    true, //legendes
                    true,
                    false
        );
       XYPlot plot = chart.getXYPlot();
       plot.setBackgroundPaint(Color.WHITE);
```

```
51          plot.setDomainGridlinesVisible(true);
52         plot.setRangeGridlinesVisible(true);
53         plot.setRangeGridlinePaint(Color.LIGHT_GRAY);
54         plot.setDomainGridlinePaint(Color.LIGHT_GRAY);
55
56          ChartPanel chartPanel = new ChartPanel(chart);
57          chartPanel.setPreferredSize(new Dimension(800,600));
58          chartPanel.setMouseWheelEnabled(true);
59          JPanel panel = new JPanel();
60
61          panel.add(chartPanel, BorderLayout.CENTER);
62          panel.validate();
63
64          return panel;
65      }
66      public JPanel afficheDonneeXY(double[][] matrice, String titre, String
            titreAxeX, String titreAxeY,TypeAffichageChartEnum typeAffichageChart )
67      {
68          int lign = matrice.length;
69          int col = matrice[0].length;
70          XYSeriesCollection donnee = new XYSeriesCollection();
71          for(int col_i = 0; col_i < col; col_i++) {
72              XYSeries seriesMin = null;
73              XYSeries seriesMax = null;
74              XYSeries series = null;
75              switch (typeAffichageChart){
76                  case DONNEE_COMPLET:
77                      series = new XYSeries(selectSerieTemporelle(col_i));
78                      break;
79                  case COMPARAISON:
80                      series = new XYSeries(selectComparaisonSerieTemp(col_i));
81                      break;
82                  default:
83                      throw new IllegalArgumentException(typeAffichageChart + " N'
                            existe pas dans la liste");
84              }
85              if (col_i > 0){
86                  seriesMin = new XYSeries("min");
87                  seriesMax = new XYSeries("max");
88              }
89            for(int lign_i = 0; lign_i < lign ; lign_i++){
90                  if(col_i > 0){
91                      seriesMin.add((lign_i + 1), matrice[lign_i][col_i] - 1.0);
92                      seriesMax.add((lign_i+ 1),matrice[lign_i][col_i]+ 1.0);
93                  }
94                  series.add((lign_i+1), matrice[lign_i][col_i]);
95              }
96              donnee.addSeries(series);
97          }
98         JFreeChart chart = ChartFactory.createXYLineChart(titre, titreAxeX,
               titreAxeY, donnee,
99                                                          PlotOrientation.VERTICAL,
100                                                         true,
```

```
                                                        true,
                                                        false);
        XYPlot plot = chart.getXYPlot();
        chart.getXYPlot().getRangeAxis().setRange(-0.4,1.2);
        plot.setBackgroundPaint(Color.WHITE);
        plot.setDomainGridlinesVisible(true);
        plot.setRangeGridlinesVisible(true);
        plot.setRangeGridlinePaint(Color.LIGHT_GRAY);
        plot.setDomainGridlinePaint(Color.LIGHT_GRAY);
        XYLineAndShapeRenderer renderer = (XYLineAndShapeRenderer) plot.
            getRenderer();
        renderer.setBaseShapesVisible(true);
               ChartPanel chartPanel = new ChartPanel(chart);
               chartPanel.setPreferredSize(new Dimension(800,600));
               chartPanel.setMouseWheelEnabled(true);
               JPanel panel = new JPanel();
               panel.add(chartPanel, BorderLayout.CENTER);
               panel.validate();
               return panel;
    }
    private String selectComparaisonSerieTemp(int index)
    {
        switch (index)
        {
            case 0:
                return "Reelle";
            case 1:
                return "Estimee";
            default:
                return "Non definie";
        }
    }
   public JPanel getChartPanel(JFreeChart c) {
        ChartPanel chartPanel = new ChartPanel(c);
        chartPanel.setPreferredSize(new Dimension(800,600));
        chartPanel.setMouseWheelEnabled(true);
        JPanel panel = new JPanel();
        panel.add(chartPanel, BorderLayout.CENTER);
        panel.validate();

        return panel;
    }

}
```

A.6 Classe Data

```
package Data_Chart;

import java.io.BufferedReader;
import java.io.File;
```

```
5  import java.io.FileReader;
6  import java.io.IOException;
7  import java.util.ArrayList;
8  import java.util.Collections;
9  import java.util.Scanner;
10 /*@author Manantena Kiady */
11 public class Data {
12         private String path;
13         private String fileName;
14         public Data(String path, String fileName){
15                 this.path = path;
16                 this.fileName = fileName;
17         }
18         public Data(){
19         }
20         public double[][] joinArrays(ArrayList<double[][]> listOfArraysToJoin){
21
22                 int rows = listOfArraysToJoin.get(0).length;
23                 int cols = listOfArraysToJoin.size();
24                 double[][] matrix = new double[rows][cols];
25                 for (int cols_i = 0; cols_i < cols; cols_i++) {
26
27                         double[][] a = listOfArraysToJoin.get(cols_i);
28
29                         for (int rows_i = 0; rows_i < rows; rows_i++) {
30
31                                 matrix[rows_i][cols_i] = a[rows_i][0];
32                         }
33                 }
34                 return matrix;
35         }
36         public double[][] rawData2Matrix(Data r) throws IOException {
37
38                 String fullPath = defineAbsoluteFilePath( r );
39                 BufferedReader buffer = new BufferedReader(new FileReader(
                       fullPath));
40                 try {
41                         StringBuilder builder = new StringBuilder();
42                         String line = buffer.readLine();
43                         int columns = line.split(",").length;
44                         int rows = 0;
45                         while (line != null) {
46                                 builder.append(line);
47                                 builder.append(System.lineSeparator());
48                                 line = buffer.readLine();
49                                 rows++;
50                         }
51                         double matrix[][] = new double[rows][columns];
52                         String everything = builder.toString();
53
54                         Scanner scan = new Scanner( everything );
55                         rows = 0;
56                         while(scan.hasNextLine()){
```

```
57                              String[] strVector = scan.nextLine().split(",");
58                              for (int i = 0; i < strVector.length; i++) {
59                                      matrix[rows][i] = Double.parseDouble(
                                            strVector[i]);
60                              }
61                              rows++;
62                      }
63                      scan.close();
64                      return matrix;
65              } finally {
66                      buffer.close();
67              }
68      }
69      private String defineAbsoluteFilePath(Data r) throws IOException {
70              String absoluteFilePath = "";
71              String workingDir = System.getProperty("user.dir");
72              String OS = System.getProperty("os.name").toLowerCase();
73              if (OS.indexOf("win") >= 0) {
74                      absoluteFilePath = workingDir + "\\" + r.getPath() + "\\"
                            + r.getFileName();
75              } else {
76                      absoluteFilePath = workingDir + "/" + r.getPath() + "/" +
                            r.getFileName();
77              }
78              File file = new File(absoluteFilePath);
79
80              if (file.exists()) {
81                      System.out.println("File found!");
82                      System.out.println(absoluteFilePath);
83              } else {
84                      System.err.println("File did not find...");
85              }
86              return absoluteFilePath;
87      }
88      public String getPath() {
89              return path;
90      }
91      public void setPath(String path) {
92              this.path = path;
93      }
94      public String getFileName() {
95              return fileName;
96      }
97      public void setFileName(String fileName) {
98              this.fileName = fileName;
99      }
100
101 }
```

Annexe B

Méthode de Jacobi

Soit θ une matrice symétrique alors d'après la propriété des matrices symétriques elle est aussi diagonalisable. Donc on peut trouver une matrice orthogonale U tel que

$$D = U^T \theta U \tag{B.1}$$

Les diagonales de D sont les valeurs propres de θ. La diagonalisation consiste donc à trouver U, une base dans laquelle θ est diagonale.

Algorithme de Jacobi

- Posons $U = P$ où P est une matrice de rotation d'angle α et $\theta^{(0)} = \theta$.
- Cherchons les composantes maximales de la matrice θ tel que si on note p_0 la ligne où se situe la valeur maximal et q_0 sa colonne on a $p_0 \neq q_0$.
- Construisons une matrice orthogonale $P^{(k)} = P_{p_0 q_0}$

En appliquant la rotation de Given, une rotation d'angle α dans le plans est donnée par

$$(P_{p_0 q_0})_{ij} = \begin{cases} 1 & si\ i = j\ avec\ j \neq p_0, j \neq q_0 \\ \cos(\alpha) & si\ i = j = p_0\ ou\ i = j = q_0 \\ \sin(\alpha) & si\ i = p_0\ et\ j = q_0 \\ -\sin(\alpha) & si\ i = q_0\ et\ j = q_0 \\ 0 & sinon \end{cases} \tag{B.2}$$

Où $\cos(\alpha) = \sqrt{0.5(1 + \frac{b}{\sqrt{a^2 + b^2}})}$ $\sin(\alpha) = \frac{a}{2\cos(\alpha)\sqrt{a^2 + b^2}}$

Avec $a = \theta(p_0, q_0)$ la composante de la matrice située à (p_0, q_0) et $b = \theta(p_0, p_0) - \theta(q_0, q_0)$

- Finalement on construit la matrice

$$A = (P^{(k)})^T * \theta * P^{(k)}$$

Les valeurs propres sont les diagonales de A.

Titre : Création d'un perceptron multicouche et application à la prédiction des séries temporelles chaotiques générées par l'application logistique

Résumé : Dans ce travail, nous avons pour objectif de développer un modèle prédictif basé sur les réseaux de neurones à propagation de l'information vers l'avant ou les perceptrons multicouches, adoptant l'algorithme de rétro-propagation comme algorithme d'apprentissage. Le réseau ainsi construit est appliqué à la prédiction des séries temporelles chaotiques qui sont générées par l'application logistique. Dans ce cadre, nous avons utilisé deux méthodes de prédiction : la prédiction à court terme, qui consiste, en connaissant la valeur de la série à l'instant t, à estimer sa valeur à un instant futur $t+1$ et la prédiction à long terme ou à plusieurs pas en avant qui donne plusieurs estimations des valeurs futures successives de la série.Les séries générées par l'application logistique peuvent être chaotiques ou non. Si $0 < \mu \leq 3$, la série est stable, si $3 < \mu \leq 3,5699$ la série oscille entre deux valeurs et si $3,5699 < \mu < 4$, la série devient chaotique. De ce fait, nous avons pris une série pour chacun des trois cas précédent. Nous montrons que le perceptron multicouche minimise l'erreur de prédiction jusqu'à 10^{-2} près dans les deux cas, à court et à long terme.

Mots clés : Réseaux de neurones artificiels, perceptron multicouche, rétro-propagation, séries temporelles chaotiques, application logistique, prédiction.

Title : Building of multilayer perceptron and application to chaotic logistic map time series prediction.

Abstract : In this work, we aim to develop a predictive model based on feed-forward neural networks or multilayer perceptrons, adopting the backpropagation as learning algorithm. The built neural network is applied in chaotic time series prediction that are generated by the well known logistic map. In this context, we used two prediction methods : short-term prediction, knowing values of the series at time t , estimating its value at a future time $t + 1$, and long-term prediction or multiple step prediction that gives several estimates future values of the series.The series generated by the logistic may be chaotic or not. If $0 < \mu \leq 3$, the series is stable, if $3 < \mu \leq 3,5699$ series oscillates between two values and if $3,5699 < \mu \leq 4$, the series becomes chaotic. Thus, we have taken a series for each of the three previous cases We show that the multilayer perceptron minimizes prediction error up to 10^{-2} in cases of short and long-term predictions.

Keywords : Artificial neural networks, multi-layer perceptron, backpropagation, chaotic time series, prediction, logistic map.

Encadreur :
M. RABOANARY Roland
Professeur Titulaire
Physique et Applications
Sciences et Technologies

Impétrant :
ANDRY MANANTENA Kiady Mahefa
Email : andrymanantenak@outlook.com
Tel :(+261) 34 64 166 95
Lot I 152 bis Lohanosy Ambohijanaka
Tana 102, Madagasikara