ETUDE DYNAMIQUE DE LA SERIE DE HENON PAR PREDICTION DE LA SERIE TEMPORELLE GENEREE PAR LA SERIE EN UTILISANT LES RESEAUX DE NEURONES ARTIFICIELS MULTICOUCHES

RALAIKOTO Mamitiana Angelo

30 juin 2023

REMERCIEMENTS

Ce mémoire est le fruit des efforts fournis et des sacrifices consentis par plusieurs personnes que je ne pourrai oublier de remercier.

Mes remerciements s'adressent d'abord à Dieu, créateur de toutes choses, pour son souffle et tous ses innombrables bienfaits.

Aussi, je remercie mon Directeur de mémoire, le Professeur RABOANARY Roland, Professeur titulaire à l'Université d'Antananarivo, d'avoir accepté de m'encadrer dans la conception et l'élaboration de ce travail, et aussi pour le dévouement manifesté malgré toutes ses nombreuses occupations.

Un grand remerciement est aussi adressé à Monsieur HANITRIARIVO Rakotoson, professeur en physique à l'université d'Antananarivo de bien vouloir présider la commission de jury de cette soutenance.

Je tiens aussi à remercier Monsieur RASOANAIVO Andriniaina, Maître de conférence en physique à l'université d'Antananarivo et qui a aimablement accepté d'être l'examinateur de ce mémoire.

Je me dois de remercier plus particulièrement mes parents : Mon père, RALAIKOTO Marolahy Mamy Cantos et ma mère RAHARILALAO Marie Regine pour tous les conseils, pour tous les encouragements et pour tous les incommensurables sacrifices consentis pour toutes mes formations académiques.

Je saisis l'occasion pour remercier ma soeur : RALAIKOTO Mamilalao Sitraka Nadia, étudiante en 2ème année en Mathématique, pour ses encouragements dans chaque dure journée que j'ai traversé et aussi pour tous ses appuies en connaissance mathématique.

Enfin, que tous ceux qui, de loin ou de près, ont participé à la réalisation de ce travail trouvent ici l'expression de ma sincère gratitude.

Table des matières

liste des figures

Liste des abréviations

BSD : Berkeley Software Distribution *(Distribution de logiciels Berkeley)*

HTTP : Hypertext Transfer Protocol *(Protocole de transfert hypertexte)*

IA : Intelligence Artificielle

MIME : Multipurpose Internet Mail Extensions *(Extensions de messagerie Internet polyvalentes)*

NMSE : Normalised Mean Square Error *(Erreur quadratique moyenne normalisée)*

NumPy : Numerical Python

RNA : Réseau de Neurone Artificiel

INTRODUCTION

Le terme "intelligence artificielle", créé par John McCarthy, est souvent abrégé par le sigle "IA". Il est défini par l'un de ses créateurs, Marvin Lee Minsky, comme "la construction de programmes informatiques qui s'adonnent à des tâches qui sont, pour l'instant, accomplies de façon plus satisfaisante par des êtres humains car elles demandent des processus mentaux de haut niveau tels que : l'apprentissage perceptuel, l'organisation de la mémoire et le raisonnement logique" [1]. Historiquement, l'idée d'intelligence artificielle semble émerger dans les années 1950 quand Alan Turing se demande si une machine peut "penser", dans l'article "Computing Machinery and Intelligence" [2]. On y trouve le côté "artificiel" atteint par l'usage des ordinateurs ou de processus électroniques élaborés et le côté "intelligence" associé à son but d'imiter le comportement.

Les systèmes dynamiques sont désignés pour décrire tout ce qui peut être en évolution dans le monde par rapport à une notion. Parmi ces systèmes, il'y a ce qu'on appelle des systèmes dynamiques d'équations non linéaires qui se comportent de façon chaotique, ce dernier ayant été un des sujets de recherche d'Henri Poincaré en 1881, dont il avait senti que quelque chose y cloche. Il pose cependant les bases et invente des outils utilisés encore aujourd'hui pour l'exploration de ce domaine. De ce fait, même les mathématiques sont l'objet d'innombrables "expériences", à savoir des simulations numériques sur ordinateur. Cependant, l'utilisation intensive de l'ordinateur comme outil permet toutefois une démarche certes moins "analytique" que l'approche classique, mais autrement plus "concrète".

L'objectif de ce mémoire est l'étude des séries de Hénon qui est un système dynamique d'équations non linéaires et qui se comportent de façon chaotique. Nous avons choisi l'application de Hénon en raison de sa simplicité et du fait qu'elle soit un système dynamique non-linéaire et chaotique, c'est un exemple intéressant à étudier à l'aide des réseaux de neurones. De ce fait, on va utiliser une approche expérimentale basée sur la prédiction par l'intelligence artificielle plus précisément, prédiction par les réseaux de neurones artificiels. Cette technique permet de reproduire le fonctionnement des modèles des neurones biologiques afin de rechercher dans l'espace des états les prochains états d'un état initial. Alors, pourra-t-on réellement déduire le comportement chaotique de ce système à l'aide des réseaux de neurones ?

Ce travail se divise en plusieurs chapitres : chapitre 1, "Généralités sur les systèmes dynamiques", chapitre 2, "L' application de HÉNON". Dans le chapitre 3, nous allons définir les "Réseaux de neurones artificiels". Le chapitre 4 donne les détails sur le langage "Python : numpy, matplotlib et Tkinter" que nous allons utiliser pour la construction des réseaux de neurones. Le

chapitre 5 est relatif à la présentation numérique de la "série de Hénon" à l'aide de python. Chapitre 6, "prédiction" où nous allons construire les réseaux de neurones et faire la prédiction de la série de Hénon. Et enfin le travail se termine sur le chapitre 7 qui se rapporte sur les "discussions" des résultats obtenus.

Chapitre 1

Généralités sur les systèmes dynamiques

1.1 Introduction

En mathématiques, en chimie ou en physique, un système dynamique est la donnée d'un système et d'une loi décrivant l'évolution temporelle de ce système. Ce peut être l'évolution d'une réaction chimique au cours du temps, le mouvement des planètes dans le système solaire (régi par la loi universelle de la gravitation de Newton) ou encore l'évolution de la mémoire d'un ordinateur sous l'action d'un programme informatique. Formellement on distingue les systèmes dynamiques à temps discrets (comme un programme informatique) des systèmes dynamiques à temps continu (comme une réaction chimique).

En physique, un système dynamique est décrit comme "une particule ou un ensemble de particules dont l'état varie dans le temps" [3]. Pour étudier son évolution, on devrait avoir la connaissance de son état à l'instant initial ainsi que sa loi d'évolution. Les conditions initiales d'un système sont la cause de son évolution ultérieure et la déterminent complètement [4].

Deux aspects importants d'un système dynamique sont qu'il est :

Causal : son avenir ne dépend que de phénomènes du passé ou du présent

Déterministe : c'est-à-dire qu'à une « condition initiale » donnée à l'instant « présent » va correspondre à chaque instant ultérieur un et un seul état « futur » possible.

1.2 Système dynamique à temps discret et système dynamique à temps continu

1.2.1 Notion d'état dynamique : aspect philosophique

Avant de parler de la notion d'état d'un système dynamique, parlons d'abord du paradoxe de Zénon qui présente la difficulté de l'étude d'un système dynamique sans notion d'étude rigoureux. Zénon demandait : « Soit une flèche en vol. À un instant, est-ce qu'elle est au repos ou en mouvement ? » Si on répondait qu'elle est en mouvement, il disait « Mais être en mouvement, c'est changer de position. À un instant, la flèche a une position, elle n'en change pas. Elle n'est donc pas en mouvement. » Si on répondait qu'elle est au repos, il disait « Mais si elle est au repos à cet instant, elle est aussi au repos à tous les autres instants, elle est donc toujours au repos. Elle n'est jamais en mouvement. Mais comment alors peut-elle passer d'une position à une autre ?

» Il en concluait qu'il n'est pas possible de dire des vérités sur ce qui est en mouvement. Tout ce qui est en mouvement serait par nature mensonger et il n'y aurait pas de vérités à propos de la matière mais seulement à propos des grandes idées, pourvu qu'elles soient immuables. Le sens commun est exactement inverse. On croit plus couramment à la vérité de ce qu'on voit qu'aux vérités métaphysiques. La théorie des systèmes dynamiques rejoint le sens commun sur ce point.

La notion d'état dynamique fournit une solution au paradoxe de Zénon : à un instant, la flèche est en mouvement, elle a une position mais elle est en train de changer de position, elle a une vitesse instantanée. Les nombres qui mesurent sa position et sa vitesse sont les valeurs de ses variables d'état. Les variables d'état sont toutes les grandeurs physiques qui déterminent l'état instantané du système et qui ne sont pas constantes a priori. On les appelle aussi les variables dynamiques. Si on prend une photo au flash, on ne voit pas que la flèche est en mouvement, mais on peut le détecter par d'autres moyens, par l'effet Doppler par exemple, sans avoir à mesurer un changement de position. L'état dynamique d'un système est un état instantané, mais c'est un état de mouvement. Il est déterminé par les valeurs de toutes les variables d'état à cet instant.

1.2.2 Espace des phases

L'espace des phases est une structure correspondant à l'ensemble de tous les états possibles du système considéré. Ce peut être un espace vectoriel, une variété différentielle ou un fibré vectoriel, un espace mesurable ...
Pour un système possédant n degrés de liberté, par exemple, l'espace des phases Γ du système possède n dimensions, de telle sorte que l'état complet $x(t) \in \Gamma$ du système à l'instant t est en général un vecteur à n composantes.

Classification des dynamiques

On distingue plusieurs grands types de dynamiques en fonction de la nature mathématique de l'espace des phases :

- lorsque Γ est un espace topologique et l'application $\phi : \Gamma \to \Gamma$ un homéomorphisme, on parle de **dynamique topologique** ;
- lorsque Γ est une variété différentielle et l'application $\phi : \Gamma \to \Gamma$ un difféomorphisme, on parle de **dynamique différentielle** ;
- lorsque Γ est une variété analytique complexe et l'application $\phi : \Gamma \to \Gamma$ est holomorphe, on parle de **dynamique holomorphe** ou **dynamique complexe** ;
- lorsque Γ est un espace mesuré et $\phi : \Gamma \to \Gamma$ une application qui préserve la mesure, on entre dans le domaine de la **théorie ergodique**, et on parle de **système dynamique mesuré**.

1.2.3 Dynamique à temps discret

Un système dynamique discret est généralement défini par une application bijective $\Phi : \Gamma \to \Gamma$ de l'espace des phases sur lui-même. Elle opère de la façon suivante : étant donnée une condition initiale x_0 de l'état du système, le premier état suivant est :

$$x_1 = \Phi(x_0) \tag{1.1}$$

Le second état, qui suit immédiatement le premier, est :

$$x_2 = \Phi(x_1) = \Phi(\Phi(x_0)) = (\Phi \circ \Phi)(x_0) = \Phi^2(x_0) \tag{1.2}$$

et ainsi de suite, de telle sorte que le n-ième état est donné par :

$$x_n = \Phi(x_{n-1}) = ... = \Phi^n(x_0) \tag{1.3}$$

Pour remonter dans le passé, il suffit d'inverser la fonction Φ , ce qui est toujours possible pour une bijection.

1.2.4 Dynamique à temps continu

Système dynamique différentiel

On considère typiquement un système différentiel du premier ordre du type :

$$\frac{d}{dt}x(t) = f(x(t), t)$$

où la fonction f définit le système dynamique étudié (pour un système à n degrés de liberté, il s'agit à proprement parler d'un champ de vecteurs à n dimensions, c'est-à-dire, d'un point de vue prosaïque, un ensemble de n fonctions scalaires).

Définition

Dans le cas où le composant temps est continu, le système dynamique est présenté par un système d'équation différentielles de la forme :

$$X' = F(X,t) \Longleftrightarrow \left\{ \begin{array}{rcll} x_1' & = & f_1(x_1, ..., x_n, t) & \\ x_2' & = & f_2(x_1, ..., x_n, t) & \\ \cdot & & & X \in D \subset \mathbb{R}^n, t \in I \subset \mathbb{R}^+ \\ \cdot & & & \\ \cdot & & & \\ x_n' & = & f_n(x_1, ..., x_n, t) & \end{array} \right. \tag{1.4}$$

Si le temps est exprimé explicitement dans la fonction F , le système est dit "non autonome" ; Autrement il est autonome.

1.3 Système linéaire et non linéaire

Nous distinguons les **systèmes dynamiques linéaires** des **systèmes dynamiques non linéaires**. Dans les premiers, le membre de droite de l'équation est une fonction dépendant linéairement de x, telle que :

$$x_{t+1} = 3x_t$$

La somme de deux solutions d'un système linéaire est également solution (« principe de superposition »). Les solutions d'une équation linéaire forment un espace vectoriel, ce qui permet l'utilisation de l'algèbre linéaire et simplifie considérablement l'analyse. Pour les systèmes à temps continu, la

transformée de Laplace permet de transformer les équations différentielles en des équations algébriques.

Par ailleurs, les systèmes non linéaires ont souvent des comportements dits chaotiques, leur analyse est en général très difficile, ce qui les rend apparemment imprévisibles.

1.4 Définition générale

Un **système dynamique** est un triplet (T, M, Φ) où T est un monoïde, noté additivement, M un ensemble et Φ une fonction

$$\Phi : U \subset T \times M \to M$$

avec :

$$\begin{aligned} I(x) &= t \in T : (t, x) \in U & (1.5)\\ \Phi(0, x) &= x & (1.6)\\ \Phi(t_2, \Phi(t_1, x)) &= \Phi(t_1 + t_2, x) & (1.7) \end{aligned}$$

pour tout $t_1, t_2, t_1 + t_2 \in I(x)$.

La fonction $\Phi(t, x)$ est appelée la **fonction d'évolution du système dynamique** : elle associe à chaque point de M une image unique, dépendamment de la variable t appelée **paramètre d'évolution**. M est appelé ***espace des phases*** ou ***espace des états***, et la variable x représente l'***état initial*** du système.

1.5 Les systèmes dynamiques et la théorie du chaos

Des systèmes dynamiques non linéaires, ou simplement linéaires par morceau, peuvent faire preuve de comportements complètement imprévisibles, qui peuvent même sembler aléatoires (alors qu'il s'agit de systèmes parfaitement déterministes). Cette imprédictibilité est appelée **chaos**. La branche des systèmes dynamiques qui s'attache à définir clairement et à étudier le chaos s'appelle la **théorie du chaos**.

Cette branche des mathématiques décrit qualitativement les comportements à long terme des systèmes dynamiques. Dans ce cadre, on ne met pas l'accent sur la recherche de solutions précises aux équations du système dynamique (ce qui, de toute façon, est souvent sans espoir), mais plutôt sur la réponse à des questions comme « Le système convergera-t-il vers un état stationnaire à long terme, et dans ce cas, quels sont les états stationnaires possibles ? » ou « Le comportement à long terme du système dépend-il des conditions initiales ? ».

Un objectif important est la description des **points fixes**, ou états stationnaires, du système ; ce sont les valeurs de la variable pour lesquelles elle n'évolue plus avec le temps. Certains de ces points fixes sont **attractifs**, ce qui veut dire que si le système parvient à leur voisinage, *il va converger vers le point fixe.*

De même, on s'intéresse aux **points périodiques**, les états du système qui se répètent au bout d'un certain nombre de pas (leur période). Les points périodiques peuvent également être

attractifs. Le théorème de Charkovski donne une contrainte sur l'ensemble des périodes possibles des points d'un système dynamique à variable réelle et fonction d'évolution continue ; notamment, s'il existe un point de période 3, il existe des points de période quelconque [5].

Le comportement chaotique de systèmes complexes n'est pas une surprise, on sait depuis longtemps que la météorologie comprend des comportements complexes et même chaotiques. La véritable surprise est plutôt la découverte de chaos dans des systèmes presque triviaux.

1.6 Attracteur

On définit la notion d'attracteur dans le contexte d'un système dynamique d'équations dont les variables sont autant de dimensions d'un espace appelé espace des phases. L'évolution d'un système à partir d'un certain état initial (correspondant à un point de l'espace des phases) est symbolisée par une trajectoire plus ou moins complexe ; chaque point de cette trajectoire représente l'"état du système" à un instant donné (si l'on suppose que la variable de paramétrage est le temps). Le terme **attracteur** désigne le comportement asymptotique (s'il est défini) en temps infini du système. Il est formellement conçu comme un ensemble (fini ou non) de points vers lesquels les trajectoires se dirigent généralement :

"*On appellera* ***attracteur*** *d'un système dynamique (à temps continu ou discret) un sous-ensemble fermé invariant de l'espace de phase, attirant tous(presque tous seraient plus raisonnable) les points d'un voisinage (le bassin de l'attracteur) lorsque le temps croît, et ayant une certaine forme d'indécomposabilité*" [6].

Il existe plusieurs catégories d'attracteurs : attracteurs ponctuels (par exemple, l'attracteur du pendule amorti),les cycles-limite,les attracteurs périodiques,les attracteurs quasi-périodiques et les **attracteurs étranges**.

Ce sont ces derniers qui nous intéressent. Plusieurs sont célèbres, parmi lesquels il faut mentionner l'attracteur de Lorentz, premier attracteur étrange connu, découvert en 1963 par Edward Lorenz à la suite de l'étude d'un sytème dynamique différentiable lié à la météorologie, et l'attracteur de Hénon, découvert en 1976 par Michel Hénon. Il cache une complexité magnifique derrière deux équations récursives d'une simplicité surprenante.

1.7 Théorie des bifurcations

La théorie des bifurcations, en mathématiques et en physique est l'étude de certains aspects des systèmes dynamiques. Une bifurcation intervient lorsqu'un petit changement d'un paramètre physique produit un changement majeur dans l'organisation du système.

1.7.1 Cas concrets

Des exemples classiques d'une bifurcation en sciences pures sont par exemple les rythmes circadiens de populations animales en biologie théorique et les solutions de météo en mathématique

et physique non linéaire, en sciences de l'ingénieur il y a aussi le flambage d'une poutre élastique (l'expérience peut être faite avec une règle d'écolier) ou les transitions de phase de matériaux (température critique de bifurcation, concentration critique).
Cas de la poutre : si l'on compresse la poutre légèrement, elle va rester droite. Tout à coup, au-delà d'une limite bien définie, la poutre va se plier de plus en plus lorsqu'on augmente la force exercée. Il y a donc bifurcation, ou brisure de symétrie, où l'on passe de l'état "poutre droite" à l'état "poutre courbée", soit dans un sens, soit dans l'autre, avec une certaine probabilité (d'où l'idée de bifurcation). Avant la bifurcation, l'état "poutre droite" était stable, après la bifurcation, il devient instable.
Un exemple en magnétisme est la température de Curie pour une transition de phase : au-delà de cette température, un matériau perd son aimantation spontanée.
La prise de la mayonnaise : le système, à l'état liquide, contient de l'huile et de l'eau (dans le jaune d'œuf). En mélangeant on forme une émulsion d'huile et eau où la quantité relative des deux varie lentement au fur et à mesure qu'on ajoute l'huile. La mayonnaise prend lorsque le liquide se transforme en gel (bifurcation appelée changement de phase en physique), et cela se produit pour un rapport eau/huile critique [7].

1.7.2 Définition

Soit un système d'équations différentielles :

$$\dot{x} = f(x, \lambda) \quad f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$$

λ est ici le paramètre contrôlant la bifurcation (la force exercée dans l'exemple précédent). On dit qu'il y a bifurcation en λ_0 si, en une valeur de λ arbitrairement proche de λ_0, il existe une dynamique topologiquement non équivalente à celle en λ_0.

Chapitre 2

L'application de HÉNON

2.1 L'application de Hénon

L'application de Michel Hénon [8], est une bijection du carré $[0,1][0,1]$ dans lui-même définie par :

$$x_{n+1} = y_n + 1 - ax_n^2 \tag{2.1}$$

$$y_{n+1} = bx_n \tag{2.2}$$

où a et b sont deux paramètres, dont des valeurs typiques sont a=**1,4** et b = **0,3**. Avec ces valeurs, la dynamique présente un attracteur étrange de nature fractale, de type Cantor (Avec a = 1,4 et b = 0,3, l'attracteur étrange disparaît totalement au profit d'un attracteur en forme d'orbite périodique, de période 7).

Hénon a obtenu ses équations en cherchant une version simplifiée du système dynamique de Lorenz à temps continu introduit en 1963.

2.2 Décomposition

L'attracteur de Hénon peut être décomposé dans une zone qui conserve la surface. Il s'agit de la *transformation de Hénon* [9] :

$$T(x,y) = (y - ax^2 + 1, bx).$$

La Transformation de Hénon provient de la composition de trois transformations successives du plan. La première est un repliement du plan dans le sens de l'axe des ordonnées *(Figure 2.1)* :

$$T_1(x,y) = (x, y - ax^2 + 1).$$

La deuxième est une symétrie autour de la bissectrice du plan *(Figure 2.2)*, selon :

$$T_2(x,y) = (y,x)$$

La troisième et dernière transformation est une contraction (ou un étirement, dans le cas de b $>$ 1) de l'axe des ordonnées *(Figure 2.3)*, selon :

$$T_3(x,y) = (x, by).$$

2.2.1 Composition de la transformation de Hénon

Repliement-étirement selon l'axe des ordonnées

Pour cette transformation on applique la transformation :

$$T_1(x, y) = (x, y - ax^2 + 1).$$

et on obtient la *(Figure 2.1)*

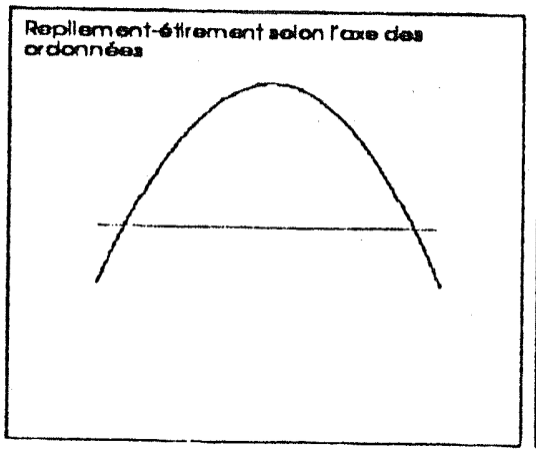

FIGURE 2.1 – Repliement-étirement selon l'axe des ordonnées

Symétrie autour de la bissectrice du plan

pour celui ci on applique la transformation :

$$T_2(x, y) = (y, x)$$

et on obtient la *(Figure 2.2)*

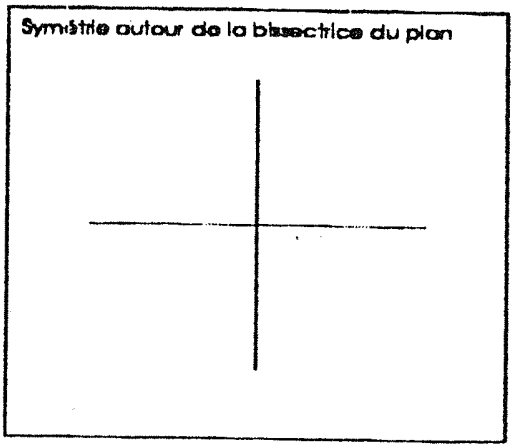

FIGURE 2.2 – Symétrie autour de la bissectrice du plan

Contraction de l'axe des ordonnées (invisible ici puisque y=0)

Où l'on applique la transformation :

$$T_3(x, y) = (x, by).$$

et on obtient la *(Figure 2.3)*

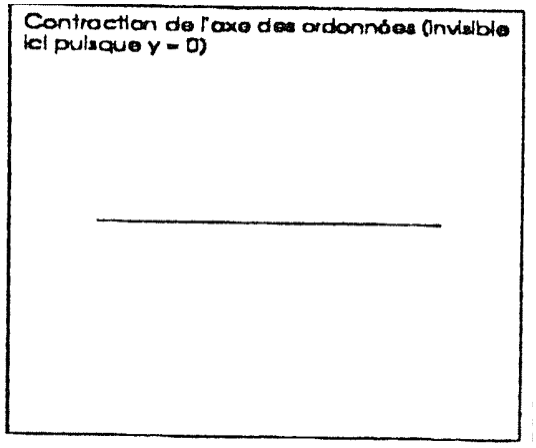

FIGURE 2.3 – Contraction de l'axe des ordonnées (invisible ici puisque y=0)

Transformation de Hénon

Et si on compose ces trois transformations on obtient la *(Figure 2.4)*

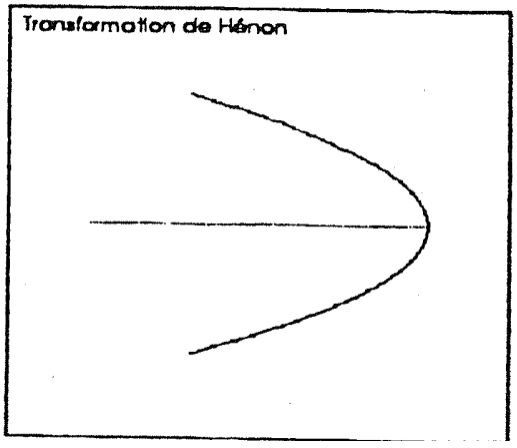

FIGURE 2.4 – Transformation de Hénon

2.2.2 Rôle de "a" et de "b" *(Figure 2.5)*

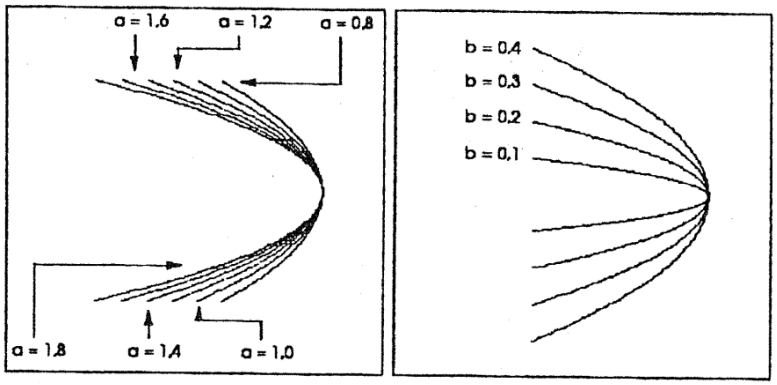

FIGURE 2.5 – Rôle de "a" et de "b"

Si l'on augmente a, la parabole s'allonge selon l'axe horizontal ; d'un autre côté, un accroissement de b entraîne un écartement vertical des branches [10].

2.2.3 Les 4 Premiers transformés de $y = 0$

$$I_1 : (x, y) = (-at^2 + 1, bt);$$

Ce qui donne :

$$I_1' : x = -\frac{a}{b^2}y^2 + 1; \quad (Figure 2.6)$$

D'où on obtient la figure :

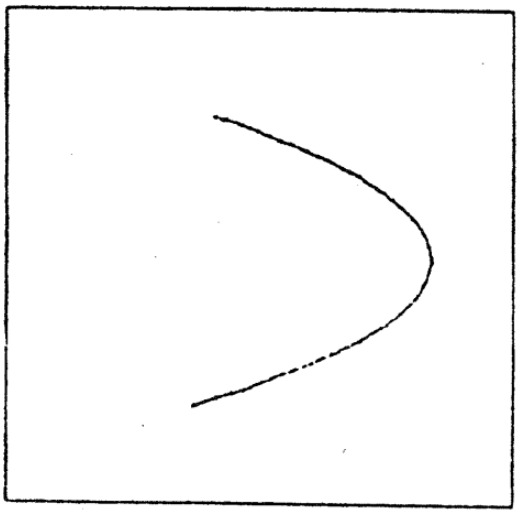

FIGURE 2.6 – T(x, y)

$$I_2 : (x, y) = [bt - a(-at^2 + 1)^2 + 1, b(-at^2 + 1)]$$

Ce qui donne :

$$I_2' : x = \pm\sqrt{\frac{b^2 - by}{a}} - \frac{a}{b^2}y^2 + 1; \quad (Figure 2.7)$$

D'où on obtient la figure :

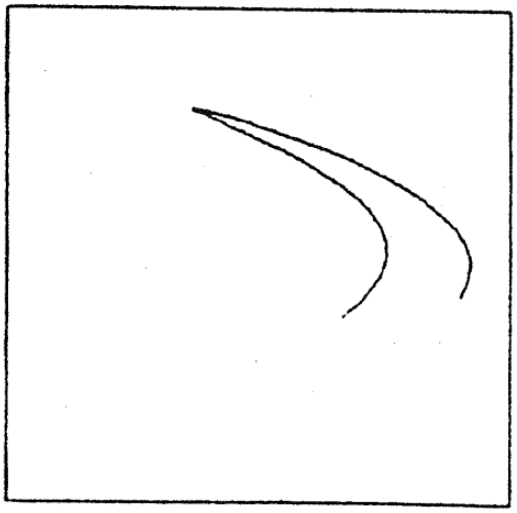

FIGURE 2.7 – T(T(x, y))

$$I_3 : (x, y) = b(-at^2 + 1) - a[bt - a(-at^2 + 1)^2 + 1]^2 + 1, b[bt - 1(-at^2 + 1)^2 + 1]];$$

Après avoir calculé son équation cartésienne on obtient la figure suivante *(Figure 2.8)* :

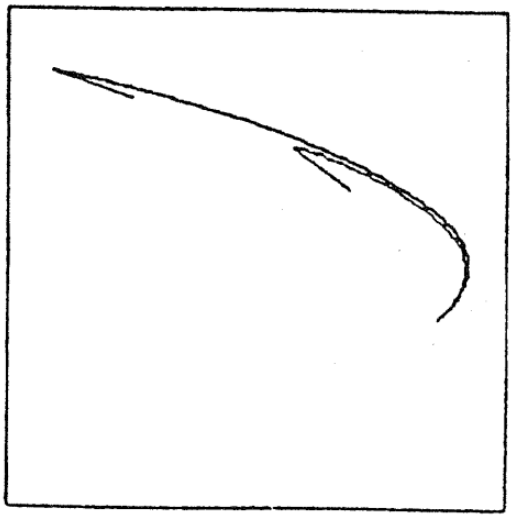

FIGURE 2.8 – T(T(T(x, y)))

On procède de la même façon pour I_4 *(Figure 2.9)* et on obtient la figure suivante :

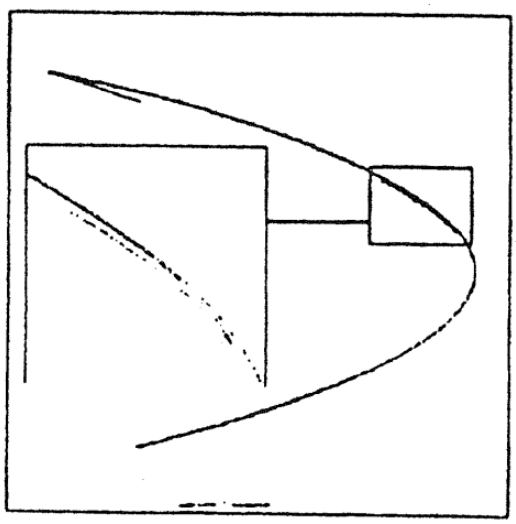

FIGURE 2.9 – T(T(T(T(x, y))))

2.3 Attracteur

2.3.1 Exploration de l'attracteur de Hénon

L'accès à l'attracteur de Hénon se fait par l'intermédiaire d'un système conceptuellement très simple. Il suffit de choisir un point P_0 du plan et d'y appliquer la transformation de Hénon définie plus tôt ; le point P_1 obtenu (le premier itéré de P_0) est alors lui-même transformé (en deuxième itéré de P_0) et ainsi de suite *(Figure 2.10)*. On écrit plus mathématiquement le processus de la façon suivante :

$$P_n = (x_n, y_n) = \begin{cases} (x_0, y_0) \quad si \quad n = 0 \\ (y_{n-1} - ax_{n-1}^2 + 1, bx_{n-1}) \end{cases} \tag{2.3}$$

processus que l'on appellera *système de Hénon*. L'ensemble P_n des itérés de P_0 se nomme orbite de P_0 par le système de Hénon. Enfin, pour étudier le comportement des orbites, on porte les points sur un plan cartésien.

Deux facteurs (et deux facteurs seulement) ont une influence sur les orbites générées : le point de départ P_0 et les paramètres a et b. Comme dans les sciences expérimentales, on étudie les effets de l'une de ces variables en fixant l'autre à une valeur donnée, ce qui permet de les traiter indépendamment.

Figure de l'agrandissement de la pointe de l'attracteur

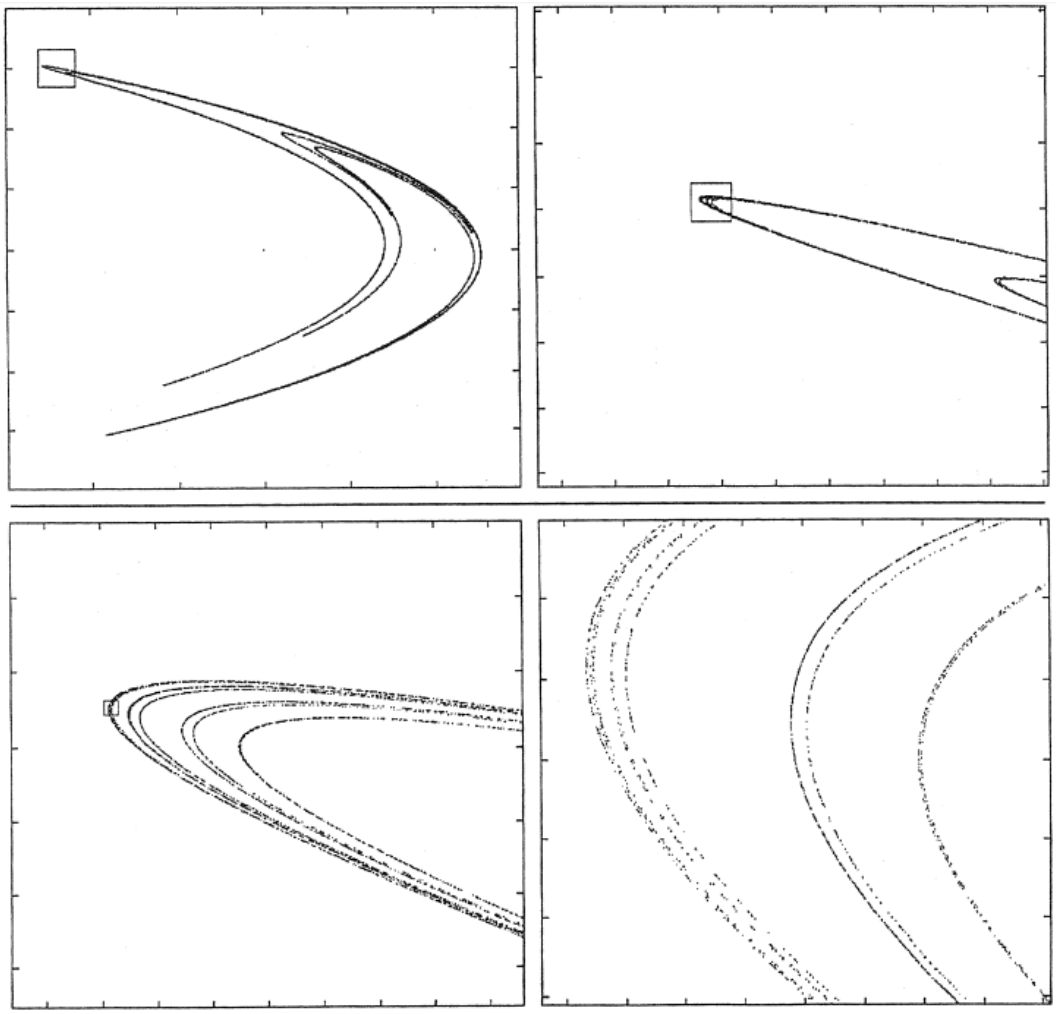

FIGURE 2.10 – Agrandissement de la pointe de l'attracteur

Chapitre 3

Les Réseaux de neurones artificiels

3.1 Origine Biologique

3.1.1 Modèle du neurone biologique

Les **modèles de neurones biologiques** *(Figure 3.1)*, sont des descriptions mathématiques des propriétés des neurones, un type de cellules du système nerveux qui génèrent des potentiels électriques d'une durée d'environ une milliseconde à travers leur membrane cellulaire. Ces potentiels électriques sont appelés potentiels d'action, pics ou impulsions.

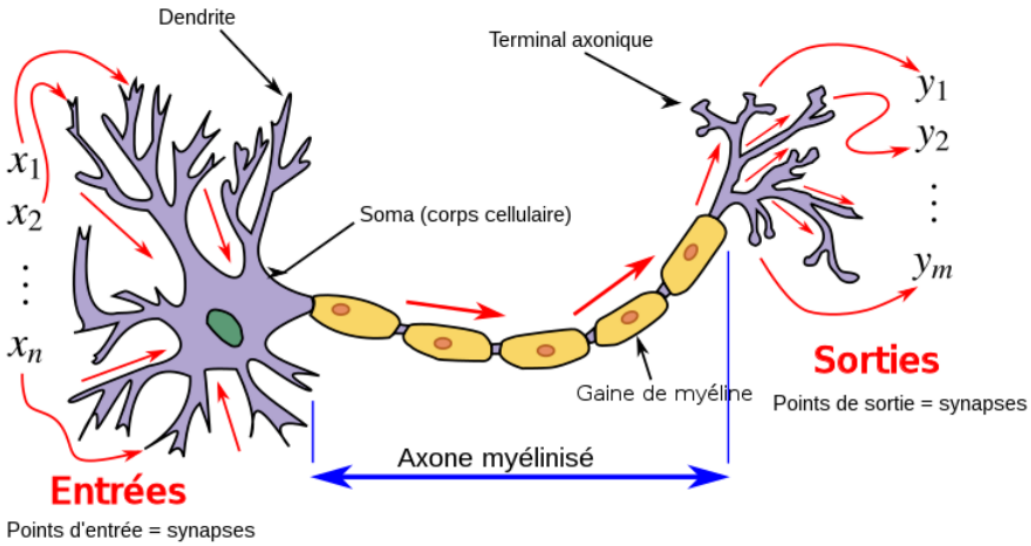

FIGURE 3.1 – Modèle du neurone biologique

3.1.2 Traitement de l'information au niveau du cerveau

Le traitement de l'information au niveau du cerveau humain est de nature électrochimique. En effet, l'information circule dans le système nerveux sous la forme de signaux électriques. La membrane d'un neurone est maintenue à une différence de potentiel d'environ -70 mV entre l'intérieur et l'extérieur du neurone. Un neurone peut transmettre un signal à des centaines d'autres selon les

conditions d'une impulsion électrique appelé "potentiel d'action". Il est généré par le neurone selon un mode "tout ou rien" lors de la dépolarisation de la membrane à partir d'une certaine valeur seuil et se propage le long de l'axone. À l'extrémité de celui-ci au niveau d'une synapse, le potentiel d'action déclenche la libération d'un transmetteur chimique se diffusant vers la membrane post-synaptique et peut être excitateur (s'il facilite le déclenchement du potentiel d'action d'un autre neurone), ou inhibiteur dans le cas contraire. les réseaux de neurones biologiques transmettent les signaux électriques à une vitesse beaucoup plus lente que dans les fils électriques.

Le *tableau 3.1* représente l'analogie entre les réseaux de neurones biologiques et les réseaux de neurone formels.

Réseaux de neurones biologiques	Réseaux de neurones artificiels
Neurones	Nœuds
Dendrite	Entrée
Synapses	Poids : associées aux connexions
Potentiel d'activation	Fonction d'activation
Axone	Sortie

TABLE 3.1 – Réseaux de neurones biologiques et artificiels

3.1.3 Perceptron

Perceptron linéaire

Le principe du perceptron linéaire est de prendre des valeurs en entrées, de faire un calcul simple et de renvoyer une valeur en sortie. Les calculs dépendent de paramètres propres à chaque perceptron [11].

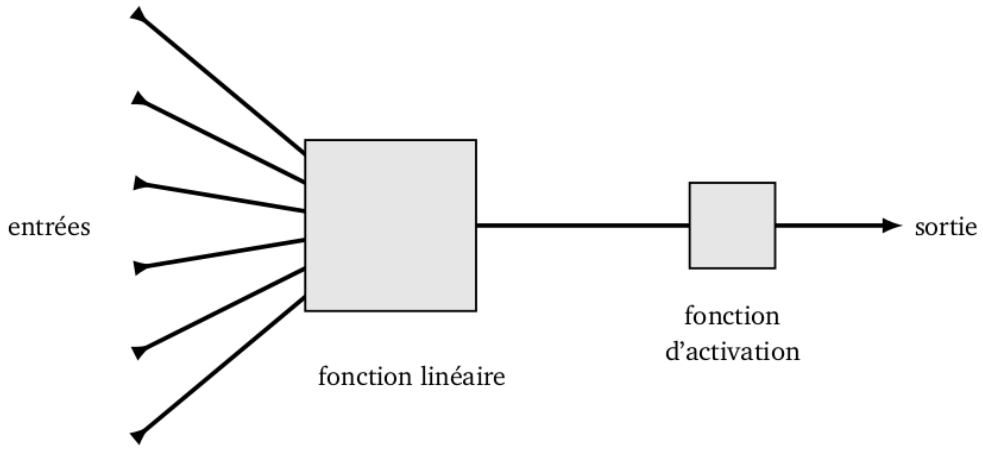

FIGURE 3.2 – Perceptron linéaire

Le calcul effectué par un perceptron se décompose en deux phases : un calcul par une fonction linéaire f, suivi d'une fonction d'activation ϕ.

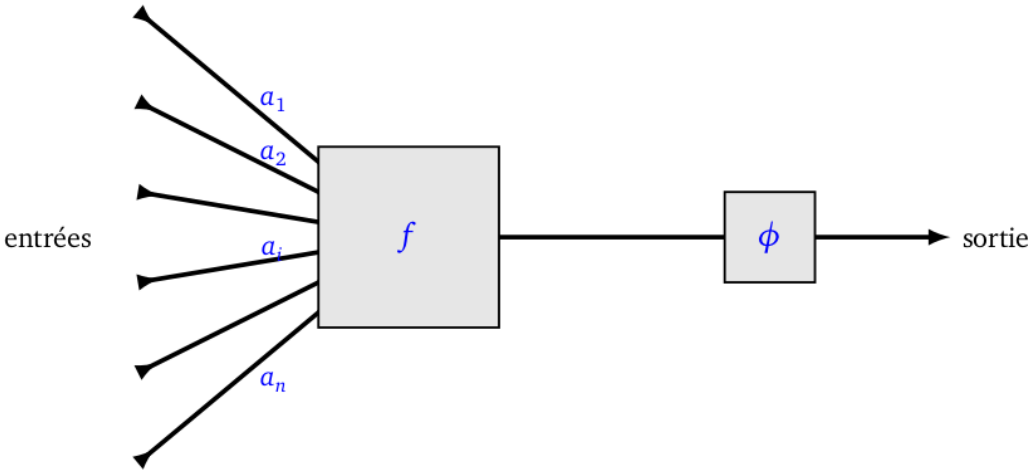

FIGURE 3.3 – Perceptron linéaire

Détaillons chaque phase du *figure 3.3*.

- **Partie linéaire.** Le perceptron est d'abord muni de **poids** $a_1, .., a_n$ qui déterminent une fonction linéaire
$$f(x_1, ..., x_n) = a_1x_1 + a_2x_2 + ... + a_nx_n$$
- **Fonction d'activation.** La valeur renvoyée par la fonction linéaire f est ensuite composée par une fonction d'activationϕ.
- **Sortie.La valeur de sortie est donc** $\phi(a_1x_1 + a_2x_2 + ... + a_nx_n)$**.**

3.1.4 Apprentissage

Mode supervisé ou non

Un apprentissage est dit supervisé lorsque le réseau est forcé à converger vers un état final précis, en même temps qu'un motif lui est présenté.

À l'inverse, lors d'un apprentissage non-supervisé, le réseau est laissé libre de converger vers n'importe quel état final lorsqu'un motif lui est présenté.

Surapprentissage

Il arrive souvent que les exemples de la base d'apprentissage comportent des valeurs approximatives ou bruitées. Si on oblige le réseau à répondre de façon quasi parfaite relativement à ces exemples, on peut obtenir un réseau qui est biaisé par des valeurs erronées.

Pour éviter le surapprentissage, il existe une méthode simple : il suffit de partager la base d'exemples en deux sous-ensembles. Le premier sert à l'apprentissage et le second sert à l'évaluation de l'apprentissage. Tant que l'erreur obtenue sur le deuxième ensemble diminue, on peut continuer l'apprentissage, sinon on arrête.

Rétropropagation

La rétropropagation consiste à rétropropager l'erreur commise par un neurone à ses synapses et aux neurones qui y sont reliés. Pour les réseaux de neurones, on utilise habituellement la *rétropropagation du gradient de l'erreur*, qui consiste à corriger les erreurs selon l'importance des éléments qui ont justement participé à la réalisation de ces erreurs : les poids synaptiques qui contribuent à engendrer une erreur importante se verront modifiés de manière plus significative que les poids qui ont engendré une erreur marginale.

3.2 Les réseaux multicouches à propagation de l'information vers l'avant et à rétro-propagation de l'erreur

3.2.1 Introduction

Les unités de ces réseaux sont organisées en couches. Elles sont connectées aux unités des niveaux suivants et seulement à celles-là. Le réseau est sans rétroaction [12].
Pour illustrer un tel type de réseau, prenons par exemple un réseau à 3 couches c'est à-dire comprenant une couche d'entrée, une couche cachée et une couche de sortie. La couche d'entrée comporte 5 unités, la couche cachée 2 unités et la couche de sortie 2 unités *(Figure 3.4)*.

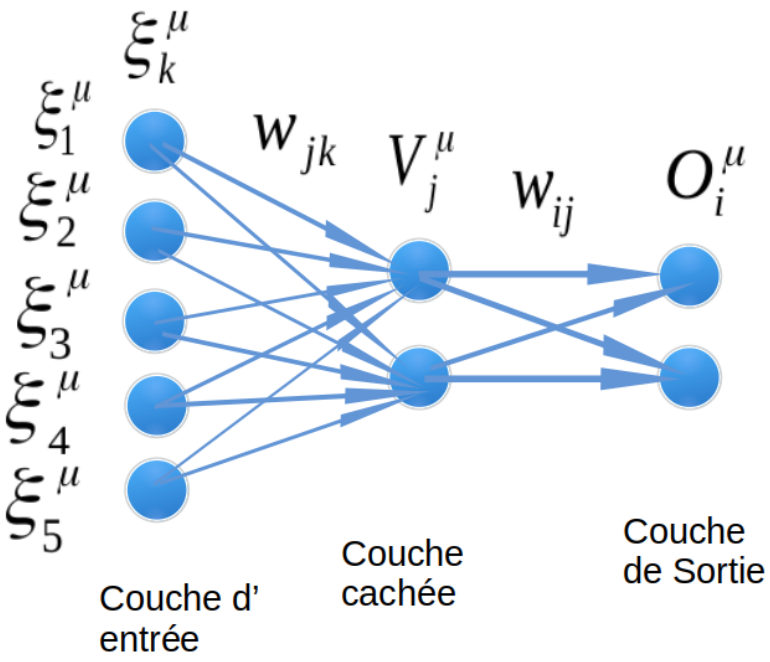

FIGURE 3.4 – réseau(5,2,2)

μ **:** indices de prototype. Un prototype est l'ensemble des données d'entrée que l'on présente à chaque fois sur les nœuds d'entrée. S'il y a p prototypes ou motifs, $\mu : 1, ..., p$

k **:** indices des nœuds d'entrée. S'il y a N unités d'entrée, $k = 1, ..., N$

j **:** indices des nœuds cachés

i **:** indices des nœuds de sortie

Pour un prototype μ quelconque, l'unité cachée j reçoit une entrée :

$$h_j^\mu = \sum_k W_{jk}\xi_k^\mu \tag{3.1}$$

et produit la sortie :

$$V_j^\mu = g(h_j^\mu) = g(\sum_k W_{jk}\xi_k^\mu) \tag{3.2}$$

où g est la fonction de transfert.
L'unité de sortie reçoit alors :

$$h_i^\mu = \sum_j W_{ij}V_j^\mu = \sum_j W_{ij}\ g(\sum_k W_{jk}\xi_k^\mu) \tag{3.3}$$

et produit la sortie finale :

$$O_i^\mu = g(h_i^\mu) = g(\sum_j W_{ij}V_j^\mu) = g(\sum_j W_{ij}\ g(\sum_k W_{jk}\xi_k^\mu)) \tag{3.4}$$

3.2.2 Modélisation de l'apprentissage (supervisé) par la méthode de descente de gradient

La méthode de descente de gradient est une technique d'optimisation issue de la recherche opérationnelle.
On définit une fonction d'erreur ou fonction de coût :

$$E(\overline{W}) = \frac{1}{2}\sum_{i\mu}(\zeta_i^\mu - O_i^\mu)^2 \tag{3.5}$$

où :

ζ_i^μ **:** est la sortie désirée

et O_i^μ **:** la sortie donnée par le réseau.

Compte tenu de l'expression de O_i^μ

$$E(\overline{W}) = \frac{1}{2}\sum_{i\mu}(\zeta_i^\mu - g(\sum_j W_{ij}\ g(\sum_k W_{jk}\xi_k^\mu)))^2 \tag{3.6}$$

La fonction de coût est une fonction continue, dérivable de chaque poids pour que nous puissions utiliser l'algorithme de descente de gradient. Il s'agit alors de minimiser la fonction d'erreur $E(\overline{W})$

Connexion couche cachée - couche de sortie

La méthode de descente de gradient donne :

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}} \tag{3.7}$$

avec η le pas d'apprentissage. Pour calculer $\dfrac{\partial E}{\partial W_{ij}}$ nous utilisons la règle de chaîne :

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial O_i^\mu} \frac{\partial O_i^\mu}{\partial W_{ij}} \tag{3.8}$$

avec $E = \dfrac{1}{2} \sum_{i\mu} (\zeta_i^\mu - O_i^\mu)^2$ et $O_i^\mu = g(\sum_j W_{ij} V_j^\mu)$

$$\Rightarrow \Delta W_{ij} = \eta \sum_\mu (\xi_i^\mu - O_i^\mu) g^{'}(h_i^\mu) V_j^\mu \tag{3.9}$$

En posant :

$$\delta_i^\mu = g^{'}(h_i^\mu)(\zeta_i^\mu - O_i^\mu) \tag{3.10}$$

On a finalement :

$$\Delta W_{ij} = \eta \sum_\mu \delta_i^\mu V_j^\mu \tag{3.11}$$

$$\boxed{\Delta W_{ij} = \eta \textstyle\sum_\mu \delta_i^\mu V_j^\mu}$$

Connexion couche d'entrée - couche cachée

Nous avons :

$$\Delta W_{jk} = -\eta \frac{\partial E}{\partial W_{jk}} \tag{3.12}$$

Utilisons de nouveau la règle de chaîne :

$$\frac{\partial E}{\partial W jk} = \frac{\partial E}{\partial O_i^\mu} \frac{\partial O_i^\mu}{\partial V_j^\mu} \frac{\partial V_j^\mu}{\partial W jk} \tag{3.13}$$

Nous obtenons :

$$\Delta W_{jk} = \eta \sum_{i\mu} (\zeta_i^\mu - O_i^\mu)\ g^{'}(h_i^\mu) W_{ij}\ g^{'}(h_j^\mu) \xi_k^\mu \tag{3.14}$$

En remarquant que $\delta_i^\mu = g^{'}(h_i^\mu(\zeta_i^\mu - O_i^\mu))$, nous pouvons écrire :

$$\Delta W_{jk} = \eta \sum_{i\mu} \delta_i^\mu W_{ij}\ g^{'}(h_j^\mu) \xi_k^\mu \tag{3.15}$$

Finalement, en posant

$$\delta_j^\mu = g^{'}(h_j^\mu) \sum_i W_{ij} \delta_i^\mu \tag{3.16}$$

nous obtenons :

$$\Delta W_{jk} = \eta \sum_{\mu} \delta_j^{\mu} \xi_k^{\mu} \tag{3.17}$$

$$\boxed{\Delta W_{jk} = \eta \sum_{\mu} \delta_j^{\mu} \xi_k^{\mu}}$$

Remarque :

- Les informations vont de la couche d'entrée vers la couche de sortie.
- Pour calculer ΔW_{jk} , on détermine d'abord ΔW_{ij} (par l'intermédiaire de δ_i^{μ} puis ΔW_{jk} qui est fonction de δ_j^{μ} , elle-même fonction de δ_i^{μ} . D'où la dénomination de « Réseau multicouche à propagation de l'information vers l'avant et à rétro-propagation de l'erreur ».

Remarque :

Pour la fonction d'activation g, il est normal d'utiliser une fonction sigmoïde.
Comme fonction sigmoïde on peut utiliser :
soit :

$$g : x \to g(x) = \frac{1}{1 + e^{-x}} \quad (Figure\ 3.5)$$

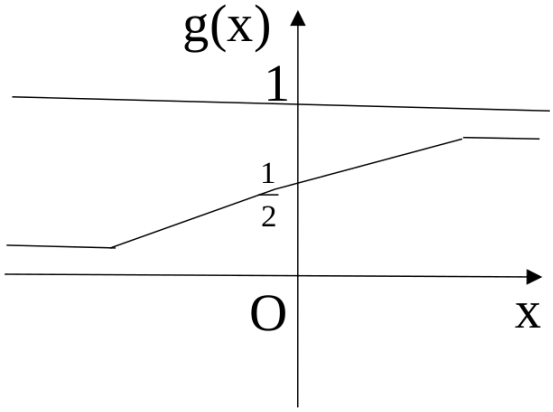

FIGURE 3.5 – Courbe de la fonction g(x)

soit :

$$g : x \to g(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad (Figure\ 3.6)$$

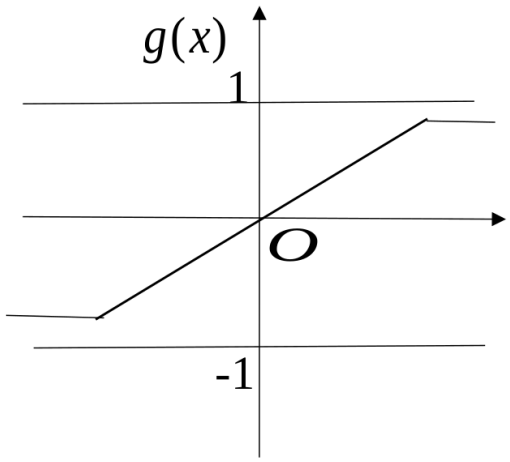

FIGURE 3.6 – Courbe de la fonction g(x)

3.2.3 Algorithme d'apprentissage par descente de gradient pour réseaux multicouches

- On considère un réseau à M couches.

Notation :

- V_i^m : sortie de la i^{me} unité dans la m^{me} couche.
  i : indice d'unité
  m : indice de couche. m = 1, 2, ..., M.
- W_{ij}^m **:** poids connexion de V_j^{m-1} à V_i^m.
- μ **:** Indice de prototype.
  S'il y a p prototypes $\mu = 1, \ldots\ldots, p$.

- On fait entrer un prototype à la fois.
- **Le procédé de propagation est le suivant :**
  - **Étape 1 :** Initialiser les poids à de petites valeurs aléatoires.
  - **Étape 2 :** Choisir un prototype ξ_k^μ et l'appliquer à la couche d'entrée ($m = 1$), c'est-à-dire : $V_k^1 = \xi_k^\mu$, $\forall k$
  - **Étape 3 :** Propager le signal vers l'avant à travers le réseau en utilisant :

$$V_i^m = g(h_i^m) = g(\sum_j W_{ij}^m V_j^{m-1})$$

    pour chaque i et m jusqu'à ce que les sorties finales V_i^M aient été toutes calculées.
  - **Étape 4 :** Calculer les deltas pour la couche de sortie :

$$\delta_i^m = g^{'}(h_i^M)(\zeta_i^\mu - V_i^M)$$

    en comparant les sorties actuelles V_i^M avec les sorties désirées ζ_i^μ pour le prototype μ.

- **Étape 5 :** Calculer les delta pour les couches précédentes en propageant les erreurs vers l'arrière :

$$\delta_i^{m-1} = g^{'}(h_i^{m-1}) \sum_j w_{ji}^m \delta_j^m \quad pour \quad m = M,\ M-1,\ M-2,\ \ldots\ldots,\ 3$$

  jusqu'à ce qu'un delta ait été calculé pour chaque unité.
- **Étape 6 :** Utiliser : $\Delta W_{ij}^m = \eta \delta_i^m V_j^{m-1}$ pour mettre à jour toutes les connexions suivant : $W_{ij}^{new} = W_{ij}^{old} + \Delta W_{ij}$.
- **Étape 7 :** Retourner à l'étape 2 et répéter les processus pour les autres prototypes.

3.3 Application des réseaux multicouches à propagation de l'information vers l'avant à la prédiction des séries temporelles

3.3.1 Architecture optimale du réseau :

Pour un problème donné, il est nécessaire d'avoir un réseau optimal car trop peu de paramètres perdrait les informations de la série et trop de paramètres consommeraient beaucoup de temps.

a) Nombre de couches cachées nécessaires : En fait, un RNA implémente un ensemble de fonctions F :

où $F : \{x_k\} \to y_i = F(\{x_k\})$ sont les données d'entrée.

On utilise entre la couche d'entrée et la première couche cachée et entre les couches cachées elles-mêmes la fonction sigmoïde

$$g : u \to g(u) = \frac{1}{1 + e^{-u}}$$

et entre la dernière couche cachée et la couche de sortie la fonction identité

$$I_d : u \to I_d(u) = u$$

Question : Pour approcher F combien aura-t-on besoin de couches cachées ?

Réponse : donnée par le théorème de Cybenko. Le théorème de Cybenko montre qu'une seule couche cachée est suffisante pour approcher toute fonction continue [13].

Démonstration : Dans un intervalle donné $[a, b]$ on peut approximer toute fonction continue f en une fonction escalier E tout en bien choisissant la subdivision de $[a, b]$ *(Figure 3.7)*

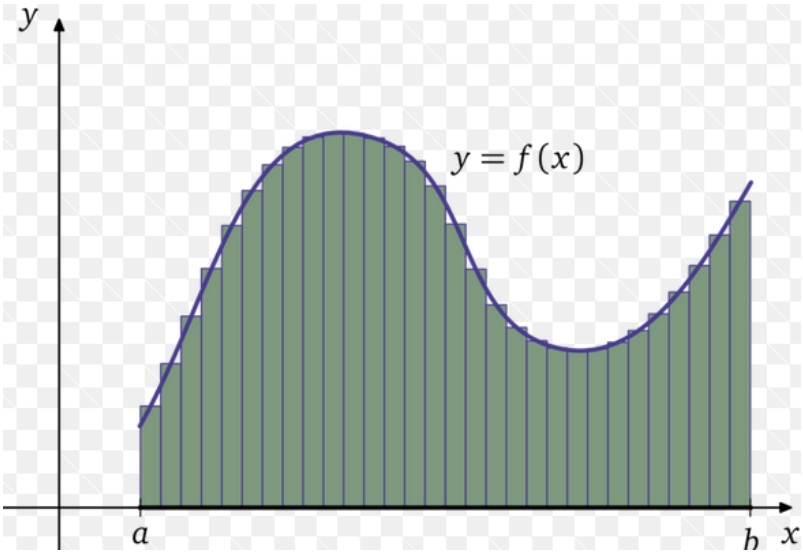

FIGURE 3.7 – Approximation d'une fonction continue

Donc il ne nous reste plus qu'à trouver le RNA qui peut construire une fonction en escalier.

- A l'aide d'un RNA à 2 couches, avec 1 unité d'entrer et 1 unité de sortie *(Figure 3.8)* on peut avoir :

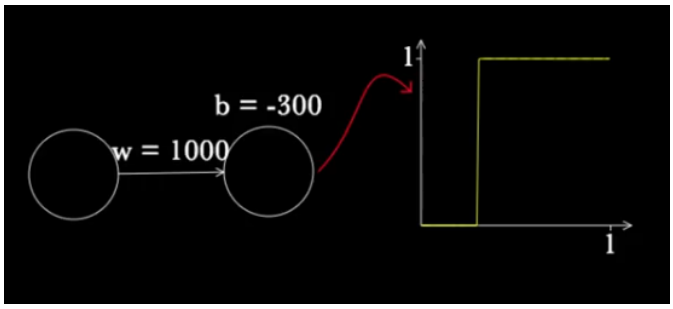

FIGURE 3.8 – construction progressive d'une fonction escalier première figure

- A l'aide d'un RNA à 2 couches, avec 1 unité d'entrer et 2 unités de sorties *(Figure 3.9)* on peut avoir :

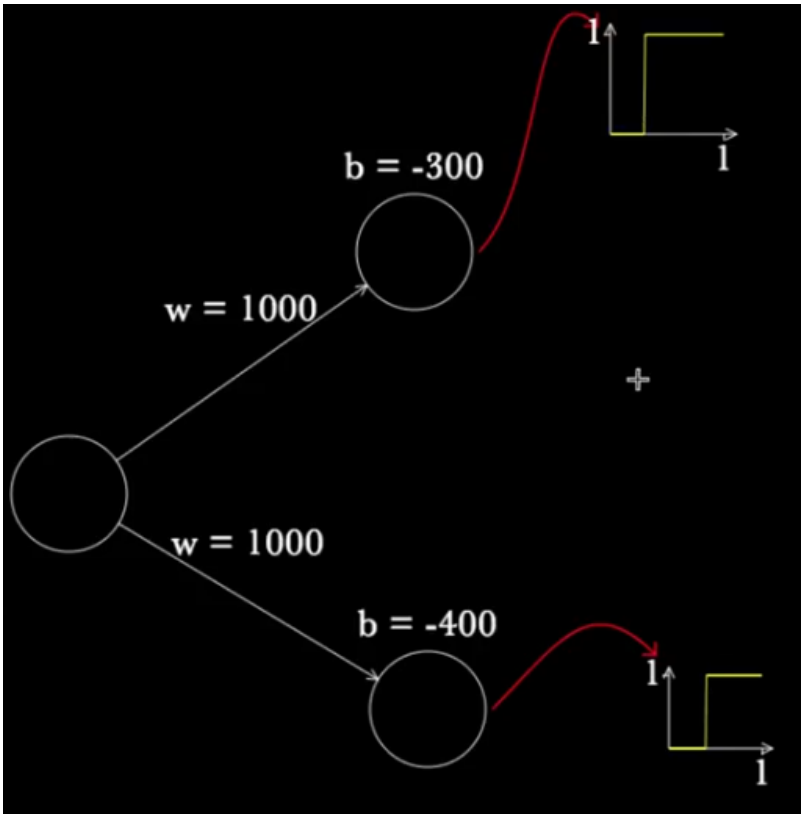

FIGURE 3.9 – construction progressive d'une fonction escalier deuxième figure

- Et avec un RNA à 3 couches *(Figure 3.10)* avec 1 unité d'entrée, 2 unités dans la couche cachée et 1 unité de sortie nous obtenons :

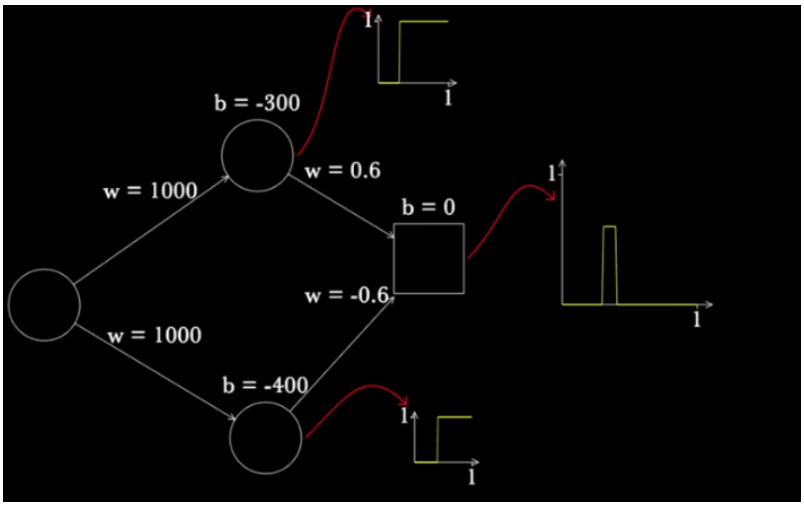

FIGURE 3.10 – construction progressive d'une fonction escalier troisième figure

- et en faisant varier le nombre d'unité de couche cachée *(Figure 3.11)* tout en prenant bien soin de choisir les " POIDS" nous pouvons approximer n'importe quelle fonction continue :

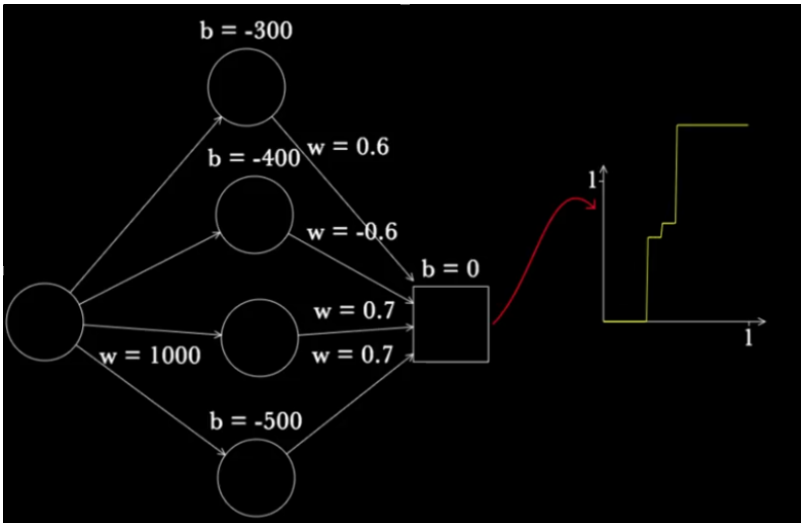

FIGURE 3.11 – construction progressive d'une fonction escalier figure 4

- Nous pouvons donc conclure qu' une seule couche cachée suffit pour approximer des fonctions continues, il suffit de faire varier les unités de la couche cachée pour ajuster les fonctions escaliers qui approximent la fonction continue.

b) Nombre d'unités d'entrée :

Ce nombre dépend du problème à traiter. Il est donné par l'algorithme de Takens qui fait appel au système dynamique en Physique.

Algorithme de Takens :

Cet algorithme permet la détermination de la dimension de plongement de l'espace de phase reconstruit et qui donne le nombre d'unités d'entrée du réseau.

Étape 1 : Soit une suite de données (série temporelle) :

$$u(1), u(2), ..., u(i), ..., u(n)$$

Étape 2 : Construire une séquence de vecteurs à partir de cette suite :

$$\bar{x}(i) = \begin{bmatrix} u(i) \\ u(i+\tau) \\ u(i+2\tau) \\ . \\ . \\ . \\ u(i+n\tau) \end{bmatrix}$$

τ : paramètre de délais (délais de mesure).
Dans la pratique, on prend n assez grand.

Étape 3 : Définir la matrice de covariance :

$$\Theta = \langle \bar{x}(i) \cdot \bar{x}(i)^t \rangle$$

Étape 4 : Calculer les vecteurs propres et les valeurs propres de Θ et ranger ces dernières par ordre décroissant.

Étape 5 : L'erreur d'approximation moyenne est :

$$\varepsilon_l = \sqrt{\lambda_{l+1}}$$

Étape 6 : Tracer ε_l en fonction de l.

Étape 7 : La première valeur de l correspondant au premier plateau de la courbe donne la dimension de plongement de l'espace de phase reconstruit et qui est égale au nombre d'unités d'entrée du RNA.

c) Nombre d'unités de Sortie :

Pour un problème de prédiction, on a besoin d'une seule unité de sortie.

d) Nombre d'unités cachées :

De façon pratique, les nombre d'unités d'entrée et de sortie ayant été fixés, on fait varier la structure du réseau en donnant différentes valeurs au nombre d'unités cachées. Le nombre d'unités cachées du réseau optimal que l'on adoptera sera celui qui donnera l'erreur d'apprentissage la plus faible.

Conclusion : Finalement nous adopterons dans un problème de prédiction, un réseau à 3 couches $(e, c, 1)$, où e désigne le nombre d'unités d'entrée, c le nombre d'unités cachées et 1 est le nombre d'unité de sortie

3.3.2 Apprentissage :

On introduit par exemple les 50 premiers prototypes qui forment une première époque. La deuxième époque sera formée par les mêmes prototypes, et ainsi de suite ...
Introduction des 50 premiers prototypes formant une première époque

N°	Prototypes	Valeurs attendues	Valeurs données par le réseau
1	x(1), x(2), x(3), x(4)	x(5)	$\hat{x}(5)$
2	x(2), x(3), x(4), x(5)	x(6)	$\hat{x}(6)$
3	x(3), x(4), x(5), x(6)	x(7)	$\hat{x}(7)$
...	.....	....	.....
50	x(50), x(51), x(52), x(53)	x(54)	$\hat{x}(54)$

FIGURE 3.12 – Exemple de disposition de l'apprentissage pour les 50 premiers prototypes

Puis on calcule l'erreur quadratique normalisée NMSE :

$$NMSE = \frac{1}{N\sigma^2}\sum_{k=1}^{N}((x(k) - \widehat{x}(k)))^2 \tag{3.18}$$

où N est le nombre de prototypes, σ^2 : variance de la série, $x(k)$: valeur attendue, $\widehat{x}(k)$: valeur donnée par le réseau.
On reprend les mêmes prototypes pour une seconde époque en utilisant les valeurs des poids trouvées, puis on calcule le NMSE correspondant, et ainsi de suite jusqu'à ce que l'erreur soit minimale.
On peut trouver ce minimum en traçant les NMSE en fonction des époques.

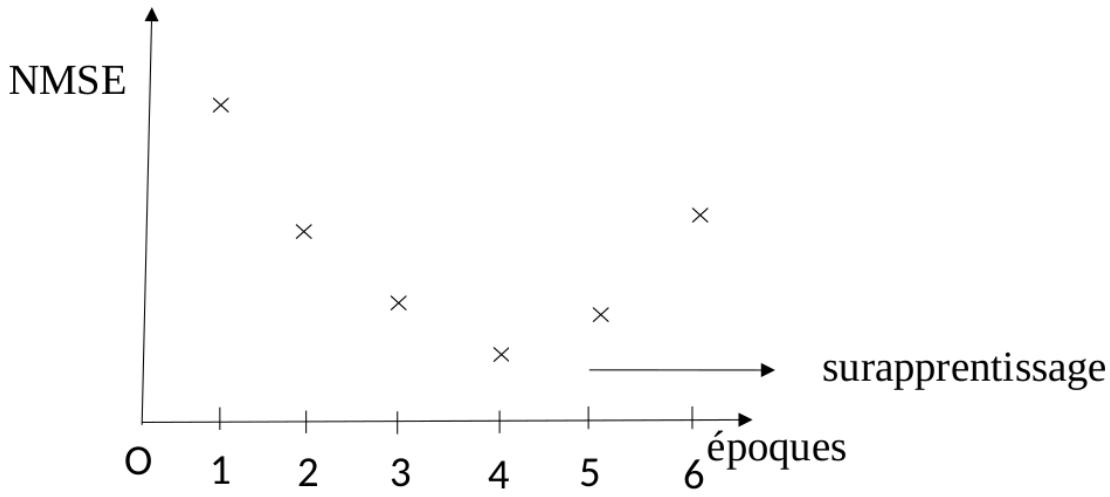

FIGURE 3.13 – Courbe NMSE

3.3.3 Prédiction :

a) Prédiction à un pas en avant :

Il s'agit de prédire à chaque fois la valeur suivante. Le mode d'introduction des données est identique à celui de l'apprentissage *(Figure 3.12)*.

Exemple :

N°	Prototypes	Valeurs à prédire	Valeurs données par le réseau
1	x(54), x(55), x(56), x(57)	x(58)	$\hat{x}(58)$
2	x(55), x(56), x(57), x(58)	x(59)	$\hat{x}(59)$

FIGURE 3.14 – Exemple de disposition d'une prédiction à un pas en avant

b) Prédiction à plusieurs pas en avant :

Le réseau est itéré en bouclage fermé. Autrement dit, la sortie donnée par le réseau est systématiquement rétro-propagée en entrée à l'itération suivante *(Figure 3.13)*.

Exemple :

Itérations	Prototypes	Valeurs données par le réseau
1	x(54), x(55), x(56), x(57)	$\hat{x}(58)$
2	$\hat{x}(58)$, x(54), x(55), x(56)	$\hat{x}(59)$
3	$\hat{x}(59)$, $\hat{x}(58)$, x(54), x(55)	$\hat{x}(60)$

FIGURE 3.15 – Exemple de disposition d'une prédiction à plusieurs pas en avant

Chapitre 4

Python : NumPy, Matplotlib et Tkinter

4.1 Python

Python *(Figure 4.1)* est un langage de programmation interprété (crée par Guido van Rossum), multiparadigme et multiplateformes [14]. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions.
Le langage Python est placé sous une licence libre proche de la licence BSD et fonctionne sur la plupart des plateformes informatiques.

4.1.1 Historique

À la fin des années 1980, le programmeur Guido van Rossum participe au développement du langage de programmation ABC au Centrum voor Wiskunde en Informatica (CWI) d'Amsterdam, aux Pays-Bas. Il travaille alors dans l'équipe du système d'exploitation Amoeba dont les appels systèmes sont difficilement interfaçables avec le Bourne shell utilisé comme interface utilisateur. Il estime alors qu'un langage de script inspiré d'ABC pourrait être intéressant comme interpréteur de commandes pour Amoeba [15].
En 1989, profitant d'une semaine de vacances durant les fêtes de Noël, il utilise son ordinateur personnel8 pour écrire la première version du langage. Fan de la série télévisée Monty Python's Flying Circus, il décide de baptiser ce projet Python [16]. Il s'est principalement inspiré d'ABC, par exemple pour l'indentation comme syntaxe ou les types de haut niveau mais aussi de Modula-3 pour la gestion des exceptions, du langage C et des outils UNIX [17].

4.1.2 Utilisation

Python est un langage de programmation qui peut s'utiliser dans de nombreux contextes et s'adapter à tout type d'utilisation grâce à des bibliothèques spécialisées. Il est cependant particulièrement utilisé comme langage de script pour automatiser des tâches simples mais fastidieuses, comme un script qui récupérerait la météo sur Internet ou qui s'intégrerait dans un logiciel de conception assistée par ordinateur afin d'automatiser certains enchaînements d'actions répétitives (voir la section Adoption). On l'utilise également comme langage de développement de prototype lorsqu'on a besoin d'une application fonctionnelle avant de l'optimiser avec un langage de plus bas niveau. Il est particulièrement répandu dans le monde scientifique, et possède de nombreuses bibliothèques optimisées destinées au calcul numérique (NumPy, SciPy, ...).

4.1.3 Bibliothèque standard

Python possède une grande bibliothèque standard, fournissant des outils convenant à de nombreuses tâches diverses. Le nombre de modules de la bibliothèque standard peut être augmenté avec des modules spécifiques écrits en C ou en Python.

La bibliothèque standard est particulièrement bien conçue pour écrire des applications utilisant Internet, avec un grand nombre de formats et de protocoles standards gérés (tels que MIME et HTTP). Des modules pour créer des interfaces graphiques et manipuler des expressions rationnelles sont également fournis. Python inclut également un framework de tests unitaires (unittest, anciennement PyUnit avant version 2.1) pour créer des suites de tests exhaustives.

4.1.4 Interfaces graphiques

Python possède plusieurs modules disponibles pour la création de logiciels avec une interface graphique. Le plus répandu est Tkinter. Ce module convient à beaucoup d'applications et peut être considéré comme suffisant dans la plupart des cas.

4.2 NumPy

NumPy *(Figure 4.2)* est une bibliothèque pour langage de programmation Python, destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux.
Plus précisément, cette bibliothèque logicielle libre et open source fournit de multiples fonctions permettant notamment de créer directement un tableau depuis un fichier ou au contraire de sauvegarder un tableau dans un fichier, et manipuler des vecteurs, matrices et polynômes [18].

4.2.1 Historique

Le langage de programmation Python n'a pas été conçu à l'origine pour le calcul numérique. Cependant, il a très tôt attiré l'attention de la communauté scientifique et technique.
En 1995, le groupe d'intérêt spécial (SIG) matrix-sig a été fondé dans le but de définir un paquetage de calcul matriciel. Parmi ses membres, Guido van Rossum, concepteur et développeur de Python, a étendu sa syntaxe, et en particulier, la syntaxe d'indexation, afin de faciliter le calcul des tableaux. Une première implémentation d'un paquetage matriciel a été réalisée par Jim Fulton, puis amélioré par Jim Hugunin et appelée Numeric, également connu sous le nom de "Numerical Python extensions" ou "NumPy".

4.2.2 Fonctionnalités

NumPy cible l'implémentation de référence CPython de Python, interpréteur non optimisé de bytecode. Les algorithmes mathématiques écrits pour cette version de Python sont souvent beaucoup plus lents que leurs équivalents qui ont été compilés. NumPy résout le problème de la lenteur en partie en fournissant des tableaux multidimensionnels avec les fonctions et opérateurs qui travaillent efficacement sur ces tableaux. Leur utilisation nécessite la réécriture d'une partie du code source, et notamment, les boucles internes.
L'utilisation de NumPy dans Python offre des fonctionnalités comparables à celles de MATLAB car

ils sont tous deux interprétés. Ils permettent tous deux à l'utilisateur d'écrire des programmes rapides en réalisant des opérations sur des tableaux ou des matrices à la place de scalaires. MATLAB dispose d'un grand nombre de boîtes à outils supplémentaires telles que Simulink. NumPy, quant à lui, est intrinsèquement intégré à Python, langage de programmation plus récent et complet. De plus, des librairies Python complémentaires sont disponibles.

4.3 Matplotlib

Matplotlib *(Figure 4.3)* est une bibliothèque du langage de programmation Python destinée à tracer et visualiser des données sous forme de graphiques [19]. Elle peut être combinée avec les bibliothèques python de calcul scientifique NumPy et SciPy [20]. Elle fournit également une API orientée objet, permettant d'intégrer des graphiques dans des applications, utilisant des outils d'interface graphique polyvalents tels que Tkinter, wxPython, Qt ou GTK.

Matplotlib est distribuée librement et gratuitement sous une licence de style BSD [21]. Sa version stable actuelle (la 2.0.1 en 2017, la 3.5.0 en novembre 2021) est compatible avec la version 3 de Python.

Plusieurs points rendent cette bibliothèque intéressante :

- Export possible en de nombreux formats matriciels (PNG, JPEG...) et vectoriels (PDF, SVG...)
- Documentation en ligne en quantité, nombreux exemples disponibles sur internet
- Forte communauté très active
- Interface pylab : reproduit fidèlement la syntaxe MATLAB
- Bibliothèque haut niveau : idéale pour le calcul interactif

Chapitre 5

Courbes et 500 premières valeurs des séries de Henon

Maintenant, que nous avons bien défini le système d'équation nécessaire à l'étude de la série de Hénon, on peut commencer l'étude de la série par prédiction de la série générée par la série de Hénon en utilisant les réseaux de neurones artificiels multicouches.

5.1 Les 500 premières valeurs de x_n et de y_n en prenant $x_0 = 0$ et $y_0 = 0$ et la courbe de y_n en fonction de x_n

En utilisant, la définition de la série de Hénon de l'équation 2.1 et l'équation 2.2 et en le simulant à l'aide de python [22], on obtient la fenêtre des 500 premières valeurs *(Figure 5.1)* suivante :

Fenêtre des 500 premières valeurs :

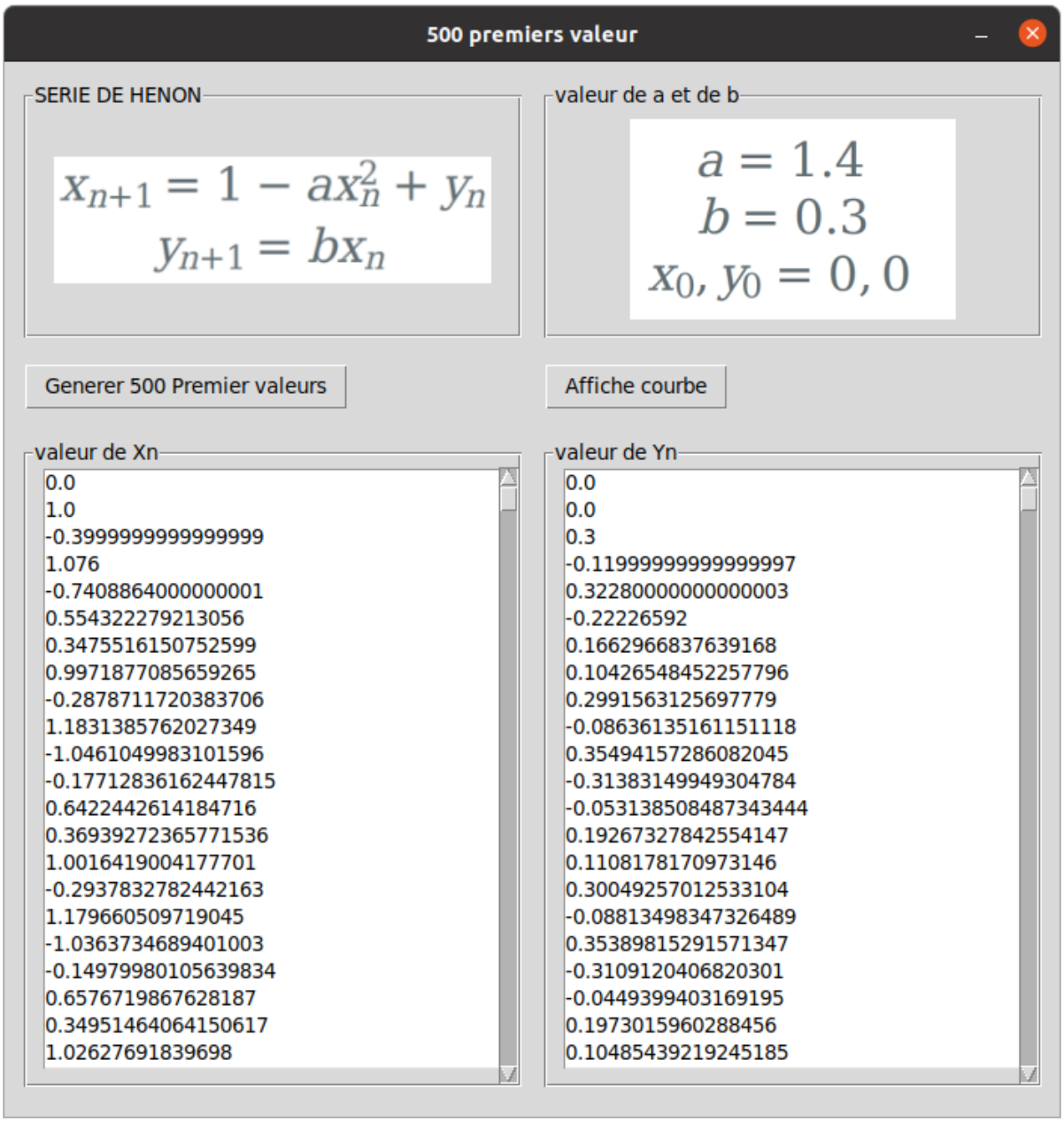

FIGURE 5.1 – 500 premières valeurs

Et avec Matplotlib, on peut utiliser les données de la série pour avoir la courbe *(Figure 5.2)* suivante :

Traçage de y_n en fonction de x_n :

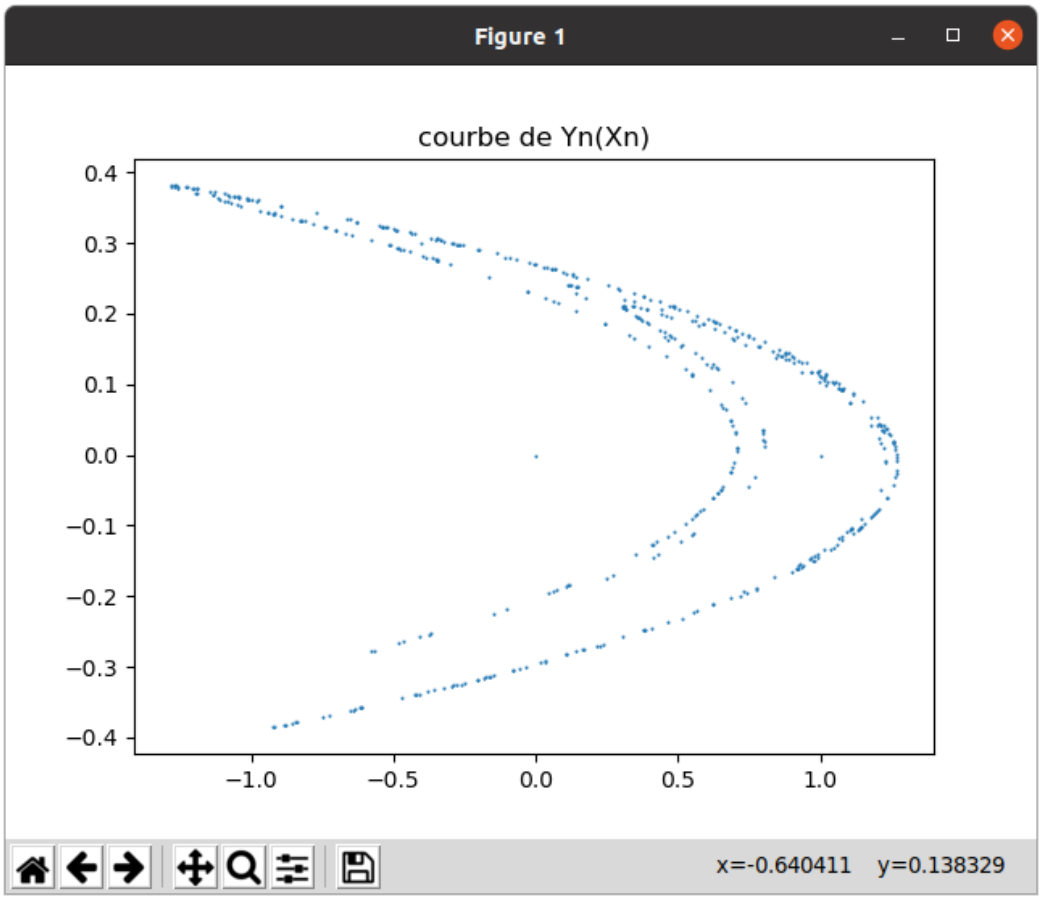

FIGURE 5.2 – courbe de y_n en fonction de x_n

Chapitre 6

Prédictions

6.1 l'architecture optimale du réseau permettant de faire les meilleures prédictions

6.1.1 Nombre de Couche cachée nécessaire

D'Après le théorème de Cybenko, une seule couche cachée est suffisante pour approcher toute fonction continue. Donc notre réseau de neurone est composé d'une couche d'entrée, d'une couche cachée et d'une couche de sortie.

6.1.2 Nombre d'unités d'entrée

Pour trouver le nombre d'unité cachée, on va utiliser **l' algorithme de Takens** *(Figure 6.1)* :

Fenêtre de gestion de la simulation de l'algorithme de Takens :

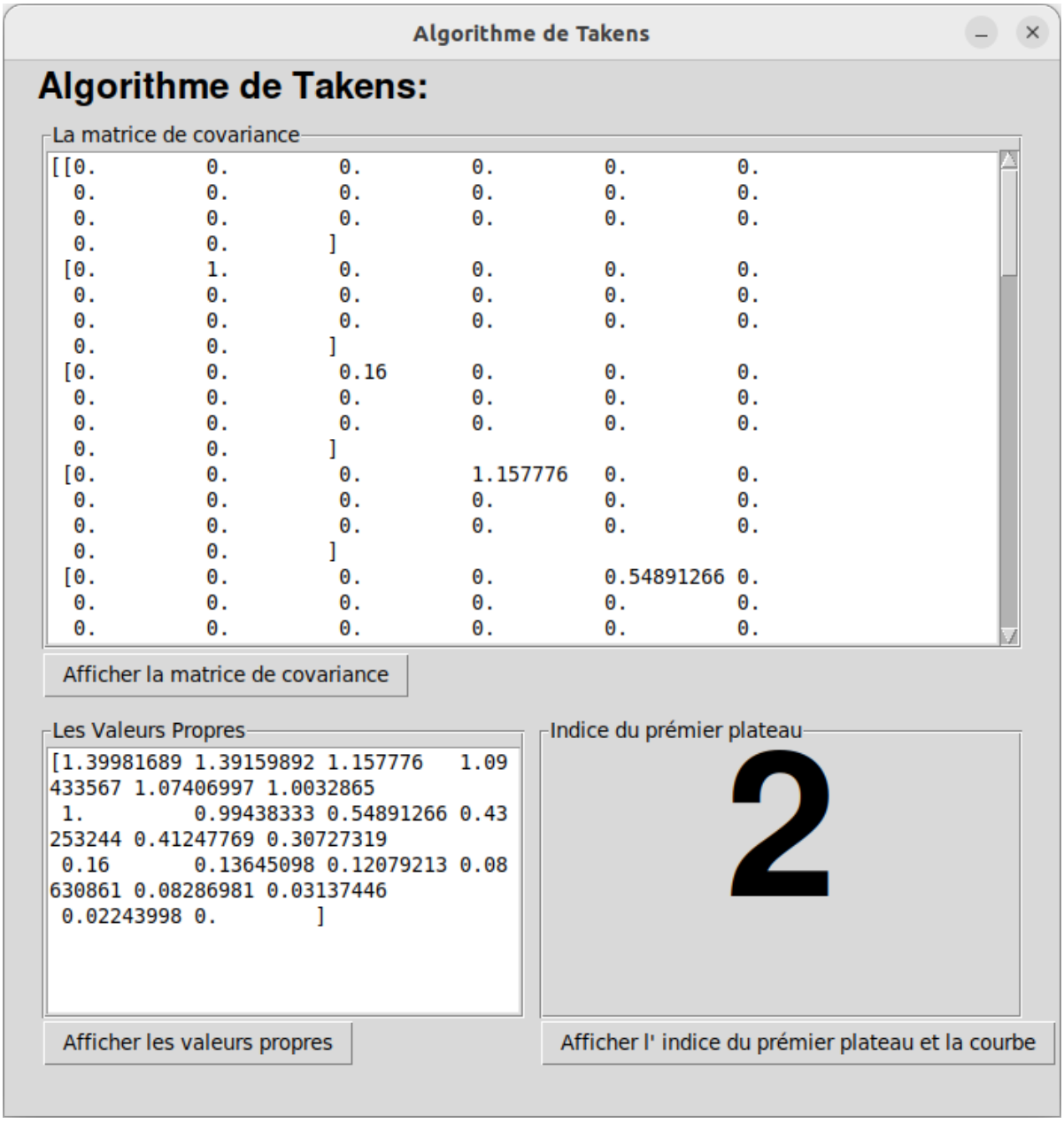

FIGURE 6.1 – Algorithme de Takens

Figure de la courbe indiquant le premier plateau de l'erreur d'approximation moyenne

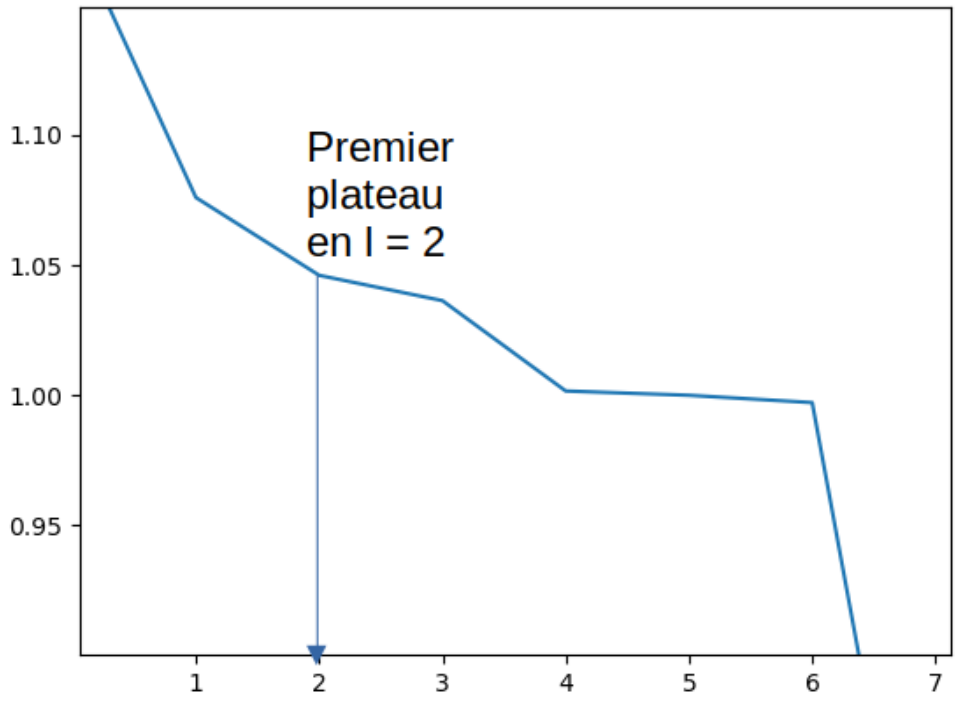

FIGURE 6.2 – Courbe indiquant le premier plateau de l'erreur d'approximation moyenne

Le premier indice de plateau se trouve sur l' indice 2, ce qui implique que le nombre d' unité d' entrée qu' on va adopter est au nombre de deux «2».

6.1.3 Nombre d'unités cachée

Pour trouver le nombre d'unités cachée, on fait varier l'architecture du réseau et on adoptera le nombre d'unités cachées du réseau optimal qui donnera l'erreur d'apprentissage la plus faible.

Représentation de l'architecture optimale du réseau

Les poids utilisés :

W11 = 0.32754594 ; W12 = -1.18991807 ;
W21 = 3.79660811 ; W22 = 1.76143971 ;
W'11 = -0.68517821 ; W'12 = 1.55664188 ;

Figure de l'architecture optimale du réseau :

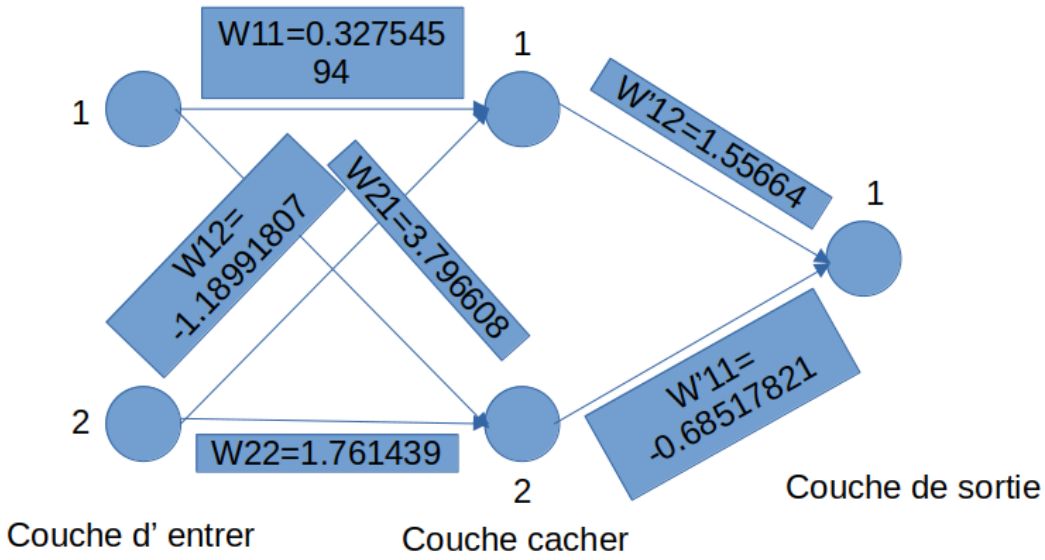

FIGURE 6.3 – Figure de l'architecture optimale du réseau

6.2 Prédiction

6.2.1 Prédiction à un pas en avant

Ici, on va utiliser l'architecture qu'on vient de définir pour prédire à chaque fois la valeur suivante, d'une autre valeur, dont le mode d'introduction des données est identique à celui de l'apprentissage *(Figure 6.7)*.

Tableau de la prédiction à un pas en avant pour 10 valeurs existantes

Tableau de la prédiction à un pas en avant

N°	Prototypes	Valeurs attendues	Valeurs données par le réseau
1	"1.0"; "-0.3999999999999999"	1.076	0.09232017155871358
2	"-0.3999999999999999"; "1.076"	-0.7408864000000001	1.2042620612839081
3	"1.076"; "-0.7408864000000001"	0.554322279213056	-0.15246222738738385
4	"-0.7408864000000001"; "0.554322279213056"	0.3475516150752599	0.897206665345576
5	"0.554322279213056"; "0.3475516150752599"	0.9971877085659265	0.7366480585203753
6	"0.3475516150752599"; "0.9971877085659265"	-0.2878711720383706	1.16721151692979
7	"0.9971877085659265"; "-0.2878711720383706"	1.1831385762027349	0.184414579881868
8	"-0.2878711720383706"; "1.1831385762027349"	-1.0461049983101596	1.2496963311476312
9	"1.1831385762027349"; "-1.0461049983101596"	-0.17712836162447815	-0.3191165894789378
10	"-1.0461049983101596"; "-0.17712836162447815"	0.6422442614184716	0.27931847478108623
11	"-0.17712836162447815"; "0.6422442614184716"	0.36939272365771536	0.9592340245618076
12	"0.6422442614184716"; "0.36939272365771536"	1.0016419004177701	0.7544762268255328
13	"0.36939272365771536"; "1.0016419004177701"	-0.2937832782442163	1.169390444492293
14	"1.0016419004177701"; "-0.2937832782442163"	1.179660509719045	0.17944693621880226
15	"-0.2937832782442163"; "1.179660509719045"	-1.0363734689401003	1.2483071448378558
16	"1.179660509719045"; "-1.0363734689401003"	-0.14979980105639834	-0.31456540200259253
17	"-1.0363734689401003"; "-0.14979980105639834"	0.6576719867628187	0.30318574237570867
18	"-0.14979980105639834"; "0.6576719867628187"	0.34951464064150617	0.9696987179637957
19	"0.6576719867628187"; "0.34951464064150617"	1.02627691839698	0.7382578404584164
20	"0.34951464064150617"; "1.02627691839698"	-0.3696876463357104	1.1812561172324922
21	"1.02627691839698"; "-0.3696876463357104"	1.1165465373245618	0.11674796847931229

FIGURE 6.4 – Tableau de la Prédiction à un pas en avant

Courbe de la Prédiction à un pas en avant

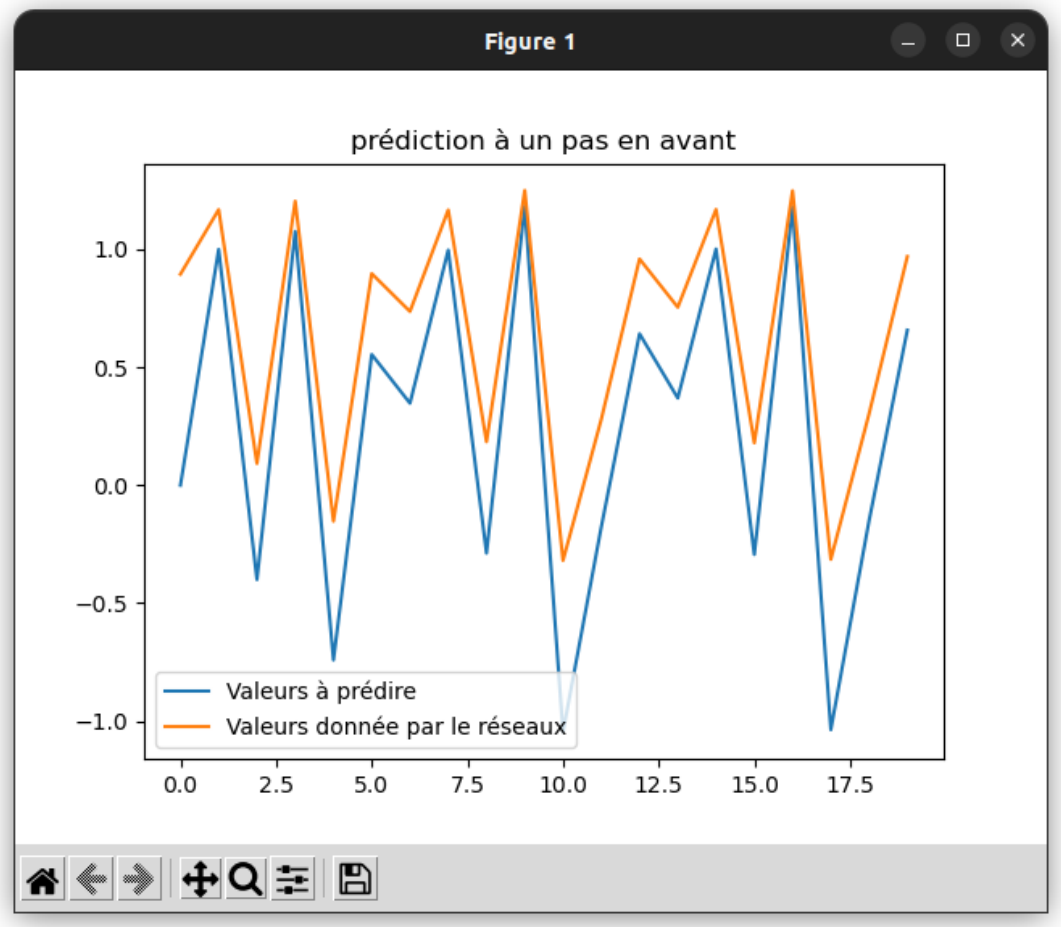

FIGURE 6.5 – Courbe de la Prédiction à un pas en avant

Sur cette courbe, l'axe des ordonnées représente les valeurs générées par le réseau de neurones et les valeurs générées par l'application de Hénon et l'axe des abscisses représente l'indice de chaque valeur générées par le réseau de neurones et l'application de Hénon. On peut voir ici que la courbe générée par le réseau de neurones imite bien le comportement de la courbe générée par l'application de Hénon.

6.2.2 Prédiction à trois pas en avant

Le réseau est itéré en bouclage fermé, ou l'on va prédire 3 valeurs successives *(Figure 6.9)*.

Tableau de la prédiction à 3 pas en avant

Tableau de la prédiction à trois pas en avant

N°	Prototypes	Valeurs attendues	Valeurs données par le réseau
15	"-0.2937832782442163"; "1.179660509719045"	-1.0363734689401003	1.2483071448378558
16	"1.2483071448378558"; "-0.2937832782442163"	-0.14979980105639834	0.17944693621880226
17	"0.17944693621880226"; "1.2483071448378558"	0.6576719867628187	1.2747061045819406

FIGURE 6.6 – Tableau de la Prédiction à trois pas en avant

Courbe de la prédiction à 3 pas en avant

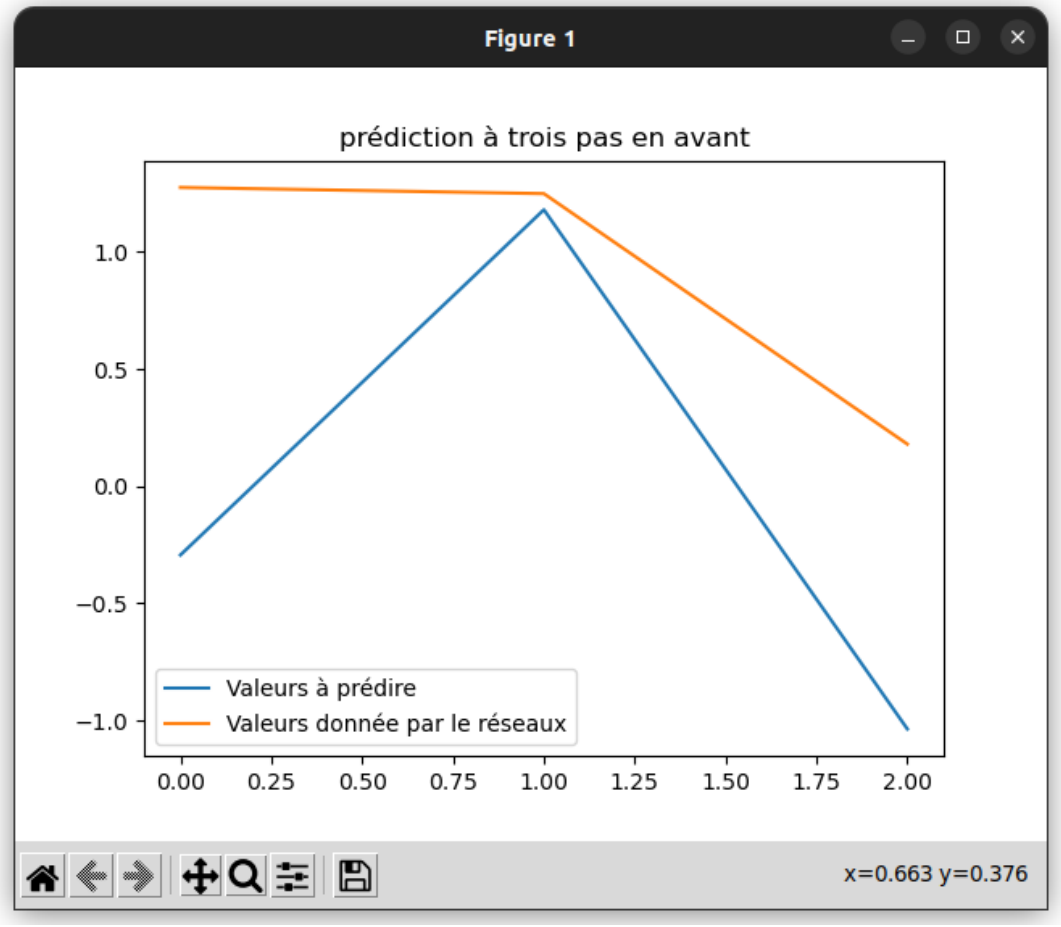

FIGURE 6.7 – Courbe de la Prédiction à trois pas en avant

Ici, on peut voir que les deux premières itérations faite par le réseau de neurones et proche des valeurs de la courbe de l'application de Hénon mais à la troisième itération les deux valeurs sont totalement opposés.

6.2.3 Prédiction à dix pas en avant

Le réseau est itéré en bouclage fermé, ou l'on va prédire 10 valeurs successives *(Figure 6.11)*.

Tableau de la prédiction à dix pas en avant

Tableau de la prédiction à trois pas en avant

N°	Prototypes	Valeurs attendues	Valeurs données par le réseau
1	"1.0"; "-0.3999999999999999"	1.076	0.09232017155871358
2	"0.09232017155871358"; "1.0"	-0.7408864000000001	1.1685884500406507
3	"1.1685884500406507"; "0.09232017155871358"	0.554322279213056	0.5176750217989108
4	"0.5176750217989108"; "1.1685884500406507"	0.3475516150752599	1.2438473435005062
5	"1.2438473435005062"; "0.5176750217989108"	0.9971877085659265	0.8701882776054103
6	"0.8701882776054103"; "1.2438473435005062"	-0.2878711720383706	1.2730552155604857
7	"1.2730552155604857"; "0.8701882776054103"	1.1831385762027349	1.1006668901931658
8	"1.1006668901931658"; "1.2730552155604857"	-1.0461049983101596	1.2837091689446256
9	"1.2837091689446256"; "1.1006668901931658"	-0.17712836162447815	1.2152153673320492
10	"1.2152153673320492"; "1.2837091689446256"	0.6422442614184716	1.2875034672129746

FIGURE 6.8 – Tableau de la Prédiction à dix pas en avant

Courbe de la prédiction à dix pas en avant

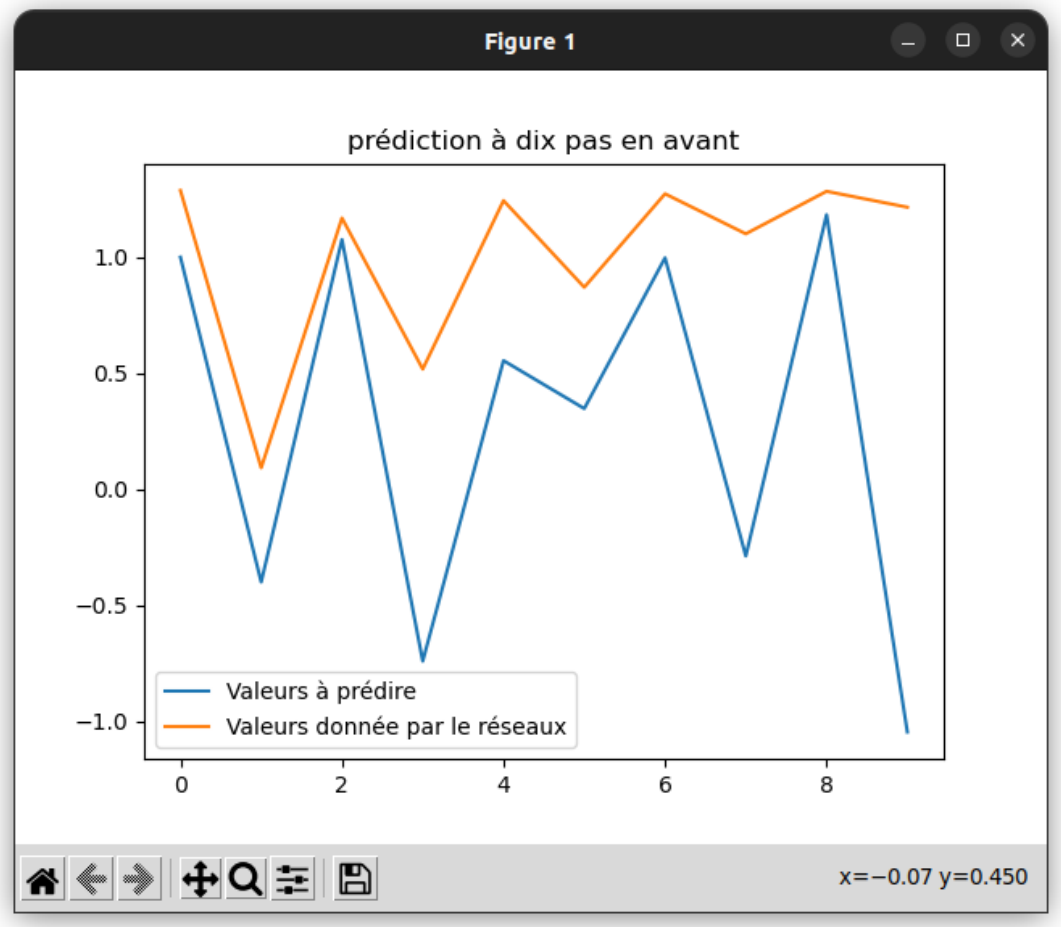

FIGURE 6.9 – Courbe de la Prédiction à dix pas en avant

À dix pas en avant, on constate que les deux courbes sont totalement différentes à partir de la troisième itération.

6.2.4 Prédiction à vingt pas en avant

Le réseau est itéré en bouclage fermé, ou l'on va prédire 20 valeurs successives *(Figure 6.13)*.

Tableau de la prédiction à vingt pas en avant

Tableau de la prédiction à trois pas en avant

N°	Prototypes	Valeurs attendues	Valeurs données par le réseau
1	"1.0"; "-0.3999999999999999"	1.076	0.09232017155871358
2	"0.09232017155871358"; "1.0"	-0.7408864000000001	1.1685884500406507
3	"1.1685884500406507"; "0.09232017155871358"	0.554322279213056	0.5176750217989108
4	"0.5176750217989108"; "1.1685884500406507"	0.3475516150752599	1.2438473435005062
5	"1.2438473435005062"; "0.5176750217989108"	0.9971877085659265	0.8701882776054103
6	"0.8701882776054103"; "1.2438473435005062"	-0.2878711720383706	1.2730552155604857
7	"1.2730552155604857"; "0.8701882776054103"	1.1831385762027349	1.1006668901931658
8	"1.1006668901931658"; "1.2730552155604857"	-1.0461049983101596	1.2837091689446256
9	"1.2837091689446256"; "1.1006668901931658"	-0.17712836162447815	1.2152153673320492
10	"1.2152153673320492"; "1.2837091689446256"	0.6422442614184716	1.2875034672129746
11	"1.2875034672129746"; "1.2152153673320492"	0.36939272365771536	1.2622460820420132
12	"1.2622460820420132"; "1.2875034672129746"	1.0016419004177701	1.2888430579472565
13	"1.2888430579472565"; "1.2622460820420132"	-0.2937832782442163	1.2798096824960532
14	"1.2798096824960532"; "1.2888430579472565"	1.179660509719045	1.2893145430081459
15	"1.2893145430081459"; "1.2798096824960532"	-1.0363734689401003	1.2861203433137935
16	"1.2861203433137935"; "1.2893145430081459"	-0.14979980105639834	1.2894803065814477
17	"1.2894803065814477"; "1.2861203433137935"	0.6576719867628187	1.2883554507149415
18	"1.2883554507149415"; "1.2894803065814477"	0.34951464064150617	1.2895385629260314
19	"1.2895385629260314"; "1.2883554507149415"	1.02627691839698	1.2891430119929614
20	"1.2891430119929614"; "1.2895385629260314"	-0.3696876463357104	1.2895590339062355

FIGURE 6.10 – Tableau de la Prédiction à vingt pas en avant

Courbe de la prédiction à vingt pas en avant

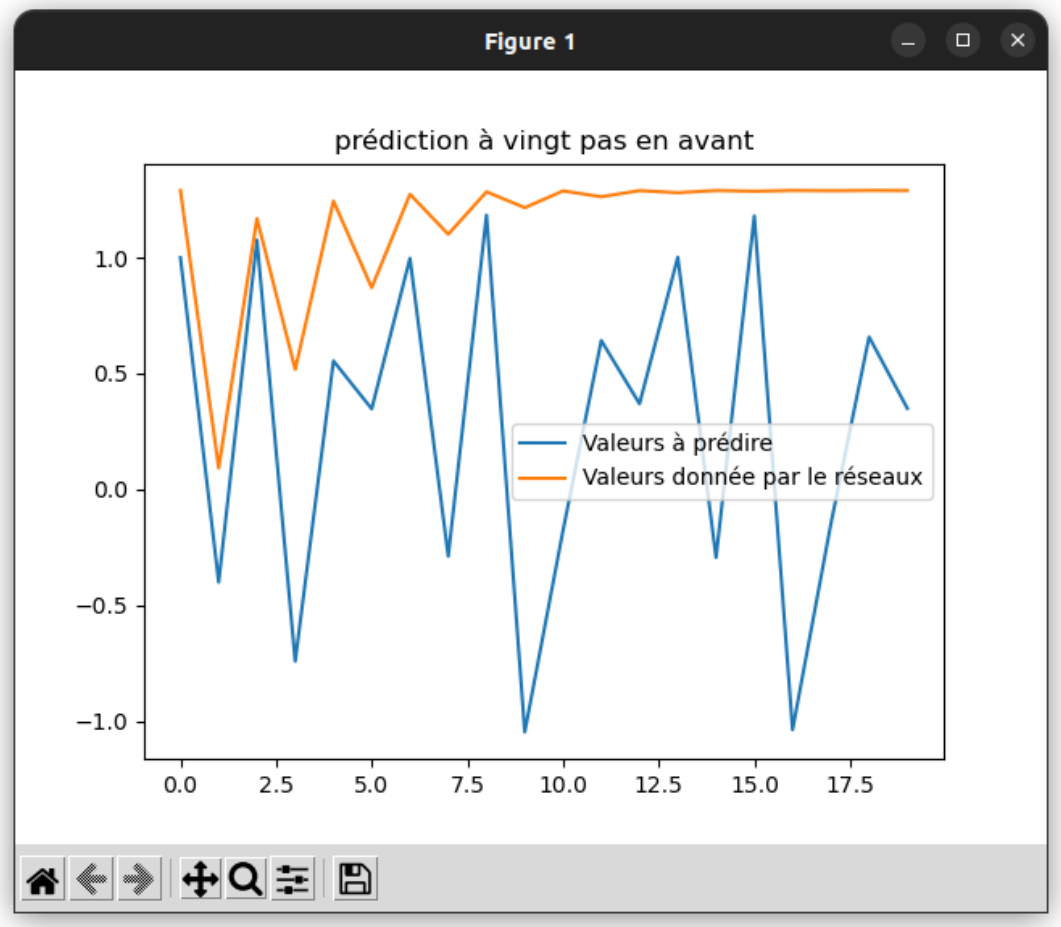

FIGURE 6.11 – Courbe de la Prédiction à vingt pas en avant

La prédiction à vingt pas en avant n'est qu'une suite des itérations de la prédiction à dix pas en avant.

Chapitre 7

Discussion

7.1 Généralité

Le long de ce travail, nous avons utilisé la méthode expérimentale pour l'étude de la série de Hénon. Or cette méthode est une méthode peu orthodoxe pour l'étude d'une si simple équation mathématique pourtant nous avons choisi cette méthode en raison du fait que la série de Hénon se comporte de façon chaotique. Les mathématiciens brésiliens Francisco Dória et Newton da Costa ont prouvé que la théorie du chaos est indécidable(preuve publiée en 1991) et que si elle est correctement axiomatisée au sein de la théorie des ensembles classique, alors elle est incomplète dans la théorie des ensembles classique au sens de Gödel. Ainsi tous cela nous permet de dire qu'une méthode expérimentale est bien adaptée et beaucoup plus concret qu'une méthode analytique.

7.2 Méthode de recherche des unités cachées

Le nombre d'unités d'entrée du réseau optimal est donné par l'algorithme de Takens et celui d'unités de sortie est pris égal à un pour un problème de prédiction. Il n'existe pas de méthode particulière pour trouver le nombre d'unités cachées optimal d'un réseau. La recherche est généralement de nature statistique. Pratiquement, on fait varier le nombre d'unités cachés dans le réseau, le nombre d'unités cachées optimal sera donc celui qui donnera l'erreur d'apprentissage la plus faible.

7.3 Choix du pas d'apprentissage η lors de l'apprentissage

Rappelons tout d'abord que lorsque l'on se rapproche d'un point minimum, le gradient tend vers 0, même si η reste constant. Cependant, il faut choisir η ni trop grand, ni trop petit : η ne doit pas être trop grand car sinon les points vont osciller autour du minimum, mais si η est trop petit alors les points ne s'approcheront du minimum qu'au bout d'un temps très long. Une solution est de faire varier η. Pour les premières itérations, on choisit un η assez grand, puis de plus en plus petit au fil des itérations.

7.4 Le langage utilisé

Nous avons choisi le langage Python pour sa rapidité et pour ses nombreux modules de traitement de donnée à savoir "numpy", "matplotlib"... . Ses nombreuses fonctions intégrées nous ont aussi facilité la tâche dans les calculs et les mises au point du réseau de neurones.

7.5 Le problème des minimums locaux

Le principe d'optimisation par la méthode de descente de gradient est de progresser de solution en solution en minimisant l'erreur à chaque pas. La solution finale est celle qui possède la plus petite erreur localement c'est-à-dire quand le gradient est nul *(Figure 7.1)*.

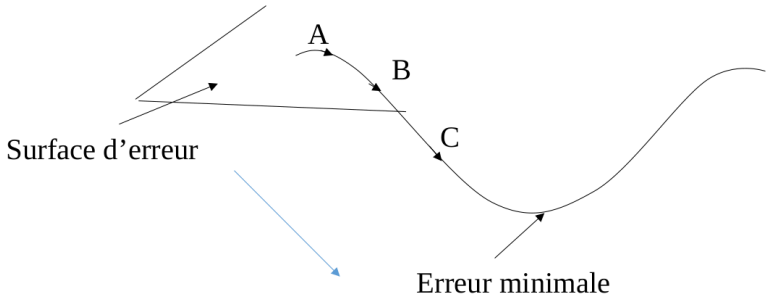

FIGURE 7.1 – Minimum, atteint quand le gradient est nul

Cette technique a l'avantage de ne nécessiter qu'une connaissance très partielle de la forme de la surface d'erreur. Seules deux hypothèses doivent être faites sur la nature de la surface :

1)- Elle doit être continue faute de quoi la recherche peut ne pas aboutir à une solution.

2)- Tous ces minimums doivent être des solutions acceptables sinon une solution trouvée peut être loin de la solution recherchée.

Si la première condition, celle de la continuité pose assez peu de problèmes en pratique, la seconde constitue un problème dans ce type de recherche, c'est le problème des minimas locaux.

Problème des minimas locaux

Etant donnée la solution initiale A, une descente de gradient donnera B comme une solution finale.
Or B n'est qu'un minimum local de la surface, la solution optimale C est inaccessible par cette méthode *(Figure 7.2)*.
Pour contourner ce problème, il est possible de combiner la recherche par descente de gradient avec une technique de recherche stochastique dite de Monte-Carlo :
Quand une solution est obtenue, on explore la surface environnante par une série de sauts aléatoires. Si l'un de ceux-ci tombe sur un point plus bas que la solution courante, la recherche recommence à partir de ce nouveau point.

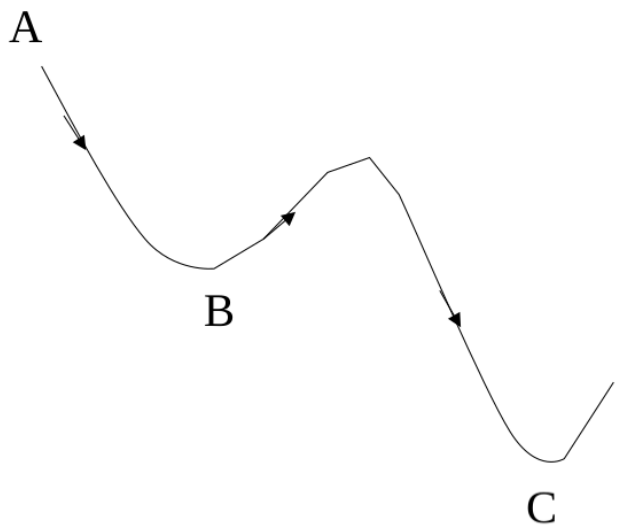

FIGURE 7.2 – B un minimum local et C la solution optimale un minimum global

CONCLUSION

L'utilisation des réseaux de neurones comme moyen d'étudier la série de Hénon(système dynamique d'équation non linéaire) est certes moins conventionnel en mathématique mais il s'agit là de l'un des meilleurs moyens d'étudier Les systèmes dynamiques non linéaires et chaotiques de nos jours.

Le long du travail nous avons pu constater l'efficacité du réseau de neurones à nous fournir les caractéristiques principales de la série de Hénon. Dans le chapitre prédiction, section « prédiction », nous avons vu que la prédiction à un pas en avant imite bien le comportement de la série, mais à chaque fois qu'on augmente le nombre d'itérations on voit que la courbe pour la prédiction s'éloigne de plus en plus de la courbe de la série, tout cela nous permet de déduire que la série de Hénon est bien chaotique en raison de son "imprévisibilité pratique", sa "sensibilité aux conditions initiales" car à chaque fois que le nombre d'itérations augmente il va y avoir des informations manquantes ce qui va changer complètement le comportement de la courbe.

On est en droit de se demander comment un processus d'une simplicité comparable à celle de la série de Hénon contient une structure aussi complexe et riche. La réponse est malheureusement inconnue ou du moins inintelligible.

Le travail que nous avons fait nous ont permit de constater que l'intelligence artificielle combinée avec la capacité de calcul des ordinateurs est un moyen révolutionnaire très pratique pour des études complexes dans la plupart des domaine d'études, parce qu'en tant qu'intelligence artificielle il imite le fonctionnement des neurones c'est-à dire qu'il est capable de reproduire le processus d'apprentissage des hommes ce qui implique que tant qu'il n'y a pas de limite au processus de raisonnement des hommes, l'intelligence artificielle ne cessera pas de s'évoluer et de nous offrir au moins les informations nécessaires à propos des domaines analytiquement difficiles en nous offrant des données numérique structurées.

Bibliographie

[1] Sur larousse. Intelligence artificielle, Août 2020. http ://www.larousse.fr/encyclopedie/divers/intelligence_artificielle/187257.

[2] A. M. Turing. *Mechanical Intelligence*. Collected Works of A. M. Turing, Octobre 1950.

[3] Springer. *Springer Nature*. 2017.

[4] David RUELLE. *Chaos, Imprédictibilité et Hasard*. 2000.

[5] Tien-Yien Li and James A. Yorke. Period three implies chaos, 1975. http ://www.its.caltech.edu/ matilde/LiYorke.pdf.

[6] Pierre ATTEN, Pierre BERGÉ, and Monique DUBOIS. *L' ordre chaotique*. La recherche, Février 1987.

[7] Janine Guespin-Michel. Les bifurcations : exemple de la mayonnaise, décembre 2016. http ://www.revolutionducomplexe.fr/index.php/la-revolution-du-complexe/les-systemes-dynamiques-non-lineaires-sdnl/29-les-bifurcations-exemple-de-la-mayonnaise.

[8] Michel Hénon. A two-dimensionnal mapping with a strange attractor, 1976. http ://projecteuclid.org/Dienst/Repository/1.0/Disseminate/euclid.cmp/1103900150/body/pdf.

[9] Predrag Cvitanoviç, Gemunu Gunaratne, and Procaccia Itamar. *Topological and metric properties of Hénon-type strane attractors*. Physical Review, 1988.

[10] L' attracteur de hénon. https ://experiences.math.cnrs.fr/L-attracteur-de-Henon.html.

[11] Arnaud Bodin and François Recher. Mathématiques des réseaux de neurones, Janvier 2021. exo7.emath.fr.

[12] Roland RABOANARY. *Cours de réseaux de neurones artificiels RNA*. 2021.

[13] George Cybenko. *Approximation by superposition of sigmoidal function, Mathematics of control, Signals Systems 2*. 1989.

[14] 2023. https ://docs.python.org/2/library/htmllib.html.

[15] Faq python 1.2 why was python created in the first place ? https ://www.python.org/doc/faq/general/#why-was-python-created-in-the-first-place.

[16] Mark Lutz. Learning python : Powerful object-oriented programming. *O' Reilly Media*, octobre 2009.

[17] Guido van Rossum. Introduction de la première édition du livre programming python de mark lutz, 1996. https ://www.python.org/doc/essays/foreword/.

[18] Scientific computing tools for python, 2023. http ://docs.scipy.org/doc/numpy/reference/.

[19] Matplotlib for python developers-preface, novembre 2009. https ://books.google.fr/books.

[20] Matplotlib for python developers-about dependencies, novembre 2009. https ://-books.google.fr/books.

[21] License, octobre 2013. http ://matplotlib.org/users/license.html.

[22] Jekyll and Cayman. The henon map|form and formula. https ://blbadger.github.io/.

[23] Description sur le site officiel de python, 2023. https ://wiki.python.org/moin/TkInter.

ANNEXES

Extrait de code

Codes générant les 500 premières valeurs et la courbe de y_n en fonction de x_n(Python(NumPy, Matplotlib))

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  def henon_attractor(x, y, a=1.4, b=0.3):
5
6          x_next = 1 - a * x ** 2 + y
7          y_next = b * x
8          return x_next, y_next
9
10 # nombre d' iteration et initialisation de l' "array"
11 steps = 699
12 X = np.zeros(steps + 1)
13 Y = np.zeros(steps + 1)
14
15 # point de depart
16 X[0], Y[0] = 0, 0
17
18 # ajout des points dans l' "array"
19 for i in range(steps):
20         x_next, y_next = henon_attractor(X[i], Y[i])
21         X[i+1] = x_next
22         Y[i+1] = y_next
23
24 liste_X_n = X
25 liste_Y_n = Y
26
27 # figure
28 def courbe():
29         plt.plot(X, Y, '^', alpha = 0.8, markersize=0.8)
30         plt.title('courbe de Yn(Xn)')
31         plt.show()
32         plt.close()
```

Codes affichant la fenêtre des 500 premières valeurs et la courbe de y_n en fonction de x_n(Python(NumPy, Matplotlib, Tkinter))

```
1 from LesPremieresValeurs import*
2 from tkinter import *
```

```
 3 import tkinter as tk
 4 from PIL import Image, ImageTk
 5
 6 class App_win(Toplevel):
 7     def __init__(self, master = None):
 8         super().__init__(master = master)
 9         self.title("Les 500 premières valeurs")
10         self.geometry("700x700")
11         self.resizable(width=0, height=0)
12
13         label_formule = tk.LabelFrame(self, text = "APPLICATION DE HÉNON")
14         label_valeur_canonique = tk.LabelFrame(self, text = "Valeurs de a et de b")
15         label_valeur_X_n = tk.LabelFrame(self, text = "Valeurs des Xn")
16         label_valeur_Y_n = tk.LabelFrame(self, text = "Valeurs des Yn")
17
18         l = Listbox(label_valeur_X_n)
19         l.place(x=10, width = 305, height = 400)
20
21         scroll=Scrollbar(label_valeur_X_n, command=l.yview)
22         scroll.pack(side="right", fill="y")
23         l.configure(yscrollcommand=scroll.set)
24
25         l_y = Listbox(label_valeur_Y_n)
26         l_y.place(x=10, width = 305, height = 400)
27
28         scroll_y = Scrollbar(label_valeur_Y_n, command = l_y.yview)
29         scroll_y.pack(side = "right", fill = "y")
30         l_y.configure(yscrollcommand = scroll_y.set)
31
32         label_formule.place(x = 13, y = 13, width = 330, height = 170)
33         label_valeur_canonique.place(x = 358, y = 13, width = 330 , height = 170)
34
35         label_valeur_X_n.place(x = 13, y = 250, width = 330, height = 430)
36         label_valeur_Y_n.place(x = 358, y = 250, width = 330 , height = 430)
37
38         btn_courbe = tk.Button(self, text = "Afficher la courbe", command = courbe)
39         btn_courbe.place(x = 358, y = 200)
40
41         def lister():
42             l.insert(0, *liste_X_n)
43             l_y.insert(0, *liste_Y_n)
44
45         btn_500_valeurs = tk.Button(self, text = "Générer les 500 Premières valeurs", command
46         btn_500_valeurs.place(x = 13, y = 200)
47
48         monimage = Image.open("serie.png")
49         photo = ImageTk.PhotoImage(monimage)
50         label = tk.Label(label_formule,image=photo)
51         label.image = photo
52         label.place(x = 8, y = 0, width = 310, height = 146)
53
54         monimage_2 = Image.open("valeurs.png")
55         photo_2 = ImageTk.PhotoImage(monimage_2)
56         label_2 = tk.Label(label_valeur_canonique,image=photo_2)
57         label_2.image = photo_2
58         label_2.place(x = 8, y = 0, width = 310, height = 146)
```

Codes de simulation de l'algorithme de Takens (Python(NumPy, Matplotlib))

```
#Etape 1:
from LesPremieresValeurs import*
import numpy as np
from numpy import linalg as LA
import matplotlib.pyplot as plt

#Etape 2:
def x_bar(i, n, tau = 1):
    return np.array([liste_X_n[i + k * tau] for k in range(n)])

#Etape 3:
def theta(taille):
    Theta = np.zeros((taille, taille))
    for i in range(taille):
        Theta[i, i] = x_bar(0, taille)[i] * np.transpose(x_bar(0, taille)[i])
    return Theta

#Etape 4:
def val_propre(matrice):
    valeur_propre = LA.eigvals(matrice)
    return valeur_propre

def tri_decroissant(liste_val_propre):
    tri = np.sort(liste_val_propre)[::-1]
    return tri

#etape 5:
def erreur_approximation_moyenne(liste):
    lambda_l = tri_decroissant(liste)
    epsilon_lsquare = lambda_l[1 : 20]
    epsilon_l = np.sqrt(epsilon_lsquare)
    return epsilon_l

#etape 6:
def courbe_epsilon():
    X_epsilon = np.arange(0, 19, dtype=int)
    Y_epsilon = erreur_approximation_moyenne(val_propre(theta(20)))

    plt.plot(X_epsilon,Y_epsilon)
    plt.show()

#etape 7:
def indicePremierPlateau():
    for i in range(20):
        a = erreur_approximation_moyenne(val_propre(theta(20)))[i]
        b = erreur_approximation_moyenne(val_propre(theta(20)))[i + 1]
        c = a-b
        if c <= 0.01:
            return i
            break
```

Codes de la fenêtre qui gère la simulation de l'algorithme de Takens

```
1  from tkinter import *
2  import tkinter as tk
3  from AlgorithmeDeTakens import*
4
5  class App_architect(Toplevel):
6      def __init__(self, master = None):
7          super().__init__(master = master)
8          self.title("Algorithme de Takens")
9          self.geometry("700x700")
10         self.resizable(width=0, height=0)
11         self.option_add("*header.font", "helvetica 18 bold")
12         self.option_add("*agrandi.font", "helvetica 100 bold")
13         self.create_label(name="header", text="Algorithme de Takens: ")
14
15
16         label_Theta = tk.LabelFrame(self, text = "La matrice de covariance")
17         label_Val_Propre = tk.LabelFrame(self, text="Les Valeurs Propres")
18         label_indiceDuPremierPlateau = tk.LabelFrame(self, text="Indice du premier plateau")
19
20         label_Theta.place(x=25, y=40, width = 650, height = 350)
21         label_Val_Propre.place(x=25, y=435, width=320, height=200)
22         label_indiceDuPremierPlateau.place(x=355, y=435, width=320, height=200)
23
24         v=Scrollbar(label_Theta, orient='vertical')
25         v.pack(side=RIGHT, fill='y')
26
27         T = Text(label_Theta, height = 320, width = 630, yscrollcommand=v.set)
28         v.config(command=T.yview)
29         T.pack()
30
31         T2 = Text(label_Val_Propre, height=170, width=300)
32         T2.pack()
33
34         def AfficheTheta():
35             Theta = theta(20)
36             fact = str(Theta)
37             T.insert(END, fact)
38
39         def AfficheValPropre():
40             ValPropre = tri_decroissant((val_propre(theta(20))))
41             fact = str(ValPropre)
42             T2.insert(END, fact)
43
44         indice = indicePremierPlateau()
45         def afficheIndice():
46             label = tk.Label(label_indiceDuPremierPlateau, name="agrandi", text=str(indice))
47             label.pack()
48             courbe_epsilon()
49
50         btn_theta = tk.Button(self, text = "Afficher la matrice de covariance", command=Affich
51         btn_theta.place(x = 25, y = 392)
52         btn_valPropre = tk.Button(self, text="Afficher les valeurs propres", command=AfficheV
53         btn_valPropre.place(x=25, y=636)
54         btn_indiceDuPremierPlateau = tk.Button(self, text="Afficher l'indice du premier platea
55         btn_indiceDuPremierPlateau.place(x=355, y=636)
56
```

```
57      def create_label(self, **options):
58          tk.Label(self, **options).pack(padx=20, pady=5, anchor=tk.W)
```

Codes de l'apprentissage du réseau

Codes 1

```
1  import math
2  import random
3  import numpy as np
4  from LesPremieresValeurs import*
5
6  #Etape 1: Initialisation des poids
7  def aleatoire(min, max, entier):
8      if(entier == 1):
9          return math.floor(random.random() * (max - min + 1) + min)
10     if(entier == 0):
11         return random.random() * (max - min + 1) + min
12     else:
13         print("erreur")
14
15 def init_poid(de, ar):
16     w = np.zeros((de, ar))
17     for i in range(de):
18         for j in range(ar):
19             w[i, j] = aleatoire(-2, 2, 0)
20     return w
21
22 def init_poid_2(de, ar):
23     w = np.zeros((de, ar))
24     for i in range(de):
25         for j in range(ar):
26             w[i, j] = aleatoire(-2, 2, 0)
27     return w
28
29 #Etape 2: choix prototype
30 def prototype(d, e):
31     boritra = np.zeros(e)
32     for i in range(e):
33         boritra[i] = liste_X_n[d+i] #le tableau liste_X_n contient les 500 premiere valeurs d
34     return boritra
35
36 def algo_descente_gradient(nbr_unites_entrer, nbr_unites_cacher, nbr_unites_sortie, Prototype
37
38     neta = 0.47
39     #Etape 3: Propager le signal vers l avant a travers le reseau
40     def sigmoide(x):
41         return 1 / (1 + math.exp(-x))
42
43     def identite(x):
44         return x
45
46     def V_entrer(nbr):
47         v = np.zeros(nbr)
48         for a in range(nbr):
49             v[a] = Prototype[a]
50         return v
```

```
51
52      def V_cacher(nbr, h):
53          v = np.zeros(nbr)
54          s = np.zeros(nbr)
55          for i in range(nbr):
56              somme = 0
57              for j in range(nbr_unites_entrer):
58                  somme += (poid_2[i, j] * V_entrer(nbr_unites_entrer)[j])
59              v[i] = sigmoide(somme)
60              s[i] = somme
61          if h == 1:
62              return v
63          if h == 0:
64              return s
65
66      def V_sortie(nbr):
67          v = np.zeros(nbr)
68          for i in range(nbr):
69              somme = 0
70              for j in range(nbr_unites_cacher):
71                  somme += (poid_3[i, j] * V_cacher(nbr_unites_cacher, 1)[j])
72              v[i] = identite(somme)
73          return v
74
75      #Etape 4:  Calculer les deltas pour la couche de sortie
76      def derive_sigmoide(x):
77          return math.exp(-x) / math.pow((1 + math.exp(-x)), 2)
78
79      def delta_sortie(nbr):
80          delta = np.zeros(nbr)
81          for i in range(nbr):
82              somme = 0
83              for j in range(nbr_unites_cacher):
84                  somme += (poid_3[i, j] * V_cacher(nbr_unites_cacher, 1)[j])
85              delta[i] = derive_sigmoide(somme) * (sortie_desire[i] - V_sortie(nbr_unites_sorti
86          return delta
87
88      #Etape 5: Calculer les delta pour les couches precedentes en propageant les erreurs vers
89      def delta_cacher(nbr):
90          delta = np.zeros(nbr)
91          for i in range(nbr):
92              delta[i] = derive_sigmoide(V_cacher(nbr_unites_cacher, 0)[i]) * poid_3[0, i] * de
93          return delta
94
95      #Etape 6: Nouvelles valeurs des poids
96      def new_poid_2():
97          delta_poid_2 = np.zeros((nbr_unites_cacher, nbr_unites_entrer))
98          New_poid_2 = np.zeros((nbr_unites_cacher, nbr_unites_entrer))
99          for i in range(nbr_unites_cacher):
100             for j in range(nbr_unites_entrer):
101                 delta_poid_2[i, j] = neta * delta_cacher(nbr_unites_cacher)[i] * V_entrer(nbr
102                 New_poid_2[i, j] = poid_2[i, j] + delta_poid_2[i, j]
103         return New_poid_2
104
105     def new_poid_3():
106         delta_poid_3 = np.zeros((nbr_unites_sortie, nbr_unites_cacher))
```

```
107         New_poid_3 = np.zeros((nbr_unites_sortie, nbr_unites_cacher))
108         for i in range(nbr_unites_sortie):
109             for j in range(nbr_unites_cacher):
110                 delta_poid_3[i, j] = neta * delta_sortie(nbr_unites_sortie)[i] * V_cacher(nbr
111                 New_poid_3[i, j] = poid_3[i, j] + delta_poid_3[i, j]
112         return New_poid_3
113
114     nouvelle_poid_2 = new_poid_2()
115     nouvelle_poid_3 = new_poid_3()
116
117     return nouvelle_poid_2, nouvelle_poid_3, V_sortie(nbr_unites_sortie)
```

Codes 2

```
1  import math
2  import numpy as np
3  from LesPremieresValeurs import*
4  from Apprentissage import*
5
6  def Apprentissage_avec_epoque(e, c, s):
7
8      def variance():
9          conteneur = np.zeros(50)
10         for i in range(50):
11             conteneur[i] = liste_X_n[i]
12         return np.var(conteneur)
13
14     def epoque_calc(e, c, s, Prototype, poid_2, poid_3, sortie_desire, liste_sortie_reseau):
15
16         somme = 0
17
18         for j in range(50):
19
20             Reseau = algo_descente_gradient(e, c, s, Prototype, poid_2, poid_3, sortie_desire
21             poid_2 = Reseau[0]
22             poid_3 = Reseau[1]
23             Prototype = prototype(j + 1, e)
24             sortie_desire = np.array([liste_X_n[j + 3]])
25             liste_sortie_reseau[j] = Reseau[2]
26             somme += math.pow( liste_X_n[j] - liste_sortie_reseau[j], 2)
27
28         NMSE = (1 / (N * variance())) * somme
29
30         return poid_2, poid_3, NMSE, liste_sortie_reseau
31
32
33     poid_2 = init_poid(c, e)
34     poid_3 = init_poid_2(s, c)
35
36     print("\n",poid_2)
37     print("\n",poid_3)
38
39     Prototype = prototype(0, e)
40     sortie_desire = np.array([liste_X_n[2]])
41     liste_sortie_reseau = np.zeros(50)
42     NMSE = 0
43     N = 50
```

```
44
45      def epoque(e, c, s, Prototype, poid_2, poid_3, sortie_desire, liste_sortie_reseau):
46          NMSE_tab = np.zeros(9)
47          liste = np.zeros((9, 50))
48          liste_poid_2 = np.zeros(((9, c, e)))
49          liste_poid_3 = np.zeros(((9, s, c)))
50          for i in range(9):
51              Epoque_calc = epoque_calc(e, c, s, Prototype, poid_2, poid_3, sortie_desire, liste
52              poid_2 = Epoque_calc[0]
53              poid_3 = Epoque_calc[1]
54              Prototype = prototype( (50 * (i+1)) + 1 , e)
55              sortie_desire = np.array([liste_X_n[(50 * (i+1)) + 3]])
56              NMSE_tab[i] = Epoque_calc[2]
57              for j in range(50):
58                  liste[i, j] = Epoque_calc[3][j]
59
60              for a in range(c):
61                  for b in range(e):
62                      liste_poid_2[i, a, b] = poid_2[a, b]
63
64              for a in range(s):
65                  for b in range(c):
66                      liste_poid_3[i, a, b] = poid_3[a, b]
67
68          return NMSE_tab, liste, liste_poid_2, liste_poid_3
69
70      u_1 = epoque(e, c, s, Prototype, poid_2, poid_3, sortie_desire, liste_sortie_reseau)[0]
71      u_2 = epoque(e, c, s, Prototype, poid_2, poid_3, sortie_desire, liste_sortie_reseau)[2]
72      u_3 = epoque(e, c, s, Prototype, poid_2, poid_3, sortie_desire, liste_sortie_reseau)[3]
73
74      #min NMSE
75
76      return u_1, u_2, u_3
77
78  nombreUniteEntre = 2
79  nombreUniteCache = 2
80
81  a = Apprentissage_avec_epoque(nombreUniteEntre,nombreUniteCache,1)[0]
82  b = Apprentissage_avec_epoque(nombreUniteEntre,nombreUniteCache,1)[1]
83  c = Apprentissage_avec_epoque(nombreUniteEntre,nombreUniteCache,1)[2]
84
85  def courbe_nmse():
86      X_nmse = np.arange(0, 9, dtype=int)
87      Y_nmse = a
88
89      plt.plot(X_nmse,Y_nmse)
90      plt.show()
91
92  def min_loc():
93      for i in range(9):
94          if a[i] < a[i+1]:
95              return i
96              break
97
98  indice_min_loc_nmse = min_loc()
```

Codes de la fenêtre qui affiche l'apprentissage de l'architecture(2,2,1)

```
from tkinter import *
import tkinter as tk
import numpy as np
from Apprentissage_2 import*

class App231(Toplevel):
    def __init__(self, master=None):
        super().__init__(master=master)
        self.title("Apprentissage")
        self.geometry("700x700")
        self.option_add("*header.font", "helvetica 18 bold")
        self.create_label(name="header", text="Apprentissage: ")

        label231 = tk.LabelFrame(self, text="Information sur l'architecture")
        label231.place(x=50, y=50, width=600, height=400)

        labelPoid = tk.LabelFrame(self, text="Les poids pour la NMSE minimale")
        labelPoid.place(x=50, y=460, width=400, height=200)

        v=Scrollbar(label231, orient='vertical')
        v.pack(side=RIGHT, fill='y')

        T = Text(label231, width=590, height=490, yscrollcommand=v.set)
        v.config(command=T.yview)
        T.pack()

        v2=Scrollbar(labelPoid, orient='vertical')
        v2.pack(side=RIGHT, fill='y')

        T2 = Text(labelPoid, width=390, height=190, yscrollcommand=v2.set)
        v2.config(command=T2.yview)
        T2.pack()

        def affichage():
            for i in range(9):
                fact1 = ("""********** """+"""ÉPOQUE"""+str(i+1)+ """ **********""" + """
                """ NMSE de l'époque """ + str(i+1) + """: """+ """\""" + str(a[i])+ """\
                """\n"""
                + """les poids obtenus entre la couche d'entrée et la couche cachée: """ +
                """\n"""
                )

            affiche_poid_2 = np.zeros((nombreUniteCache, nombreUniteEntre))
            affiche_poid_3 = np.zeros((1, nombreUniteCache))
            for x in range(nombreUniteCache):
                for y in range(nombreUniteEntre):
                    affiche_poid_2[x, y] = b[i, x, y]
            for x in range(1):
                for y in range(nombreUniteCache):
                    affiche_poid_3[x, y] = c[i, x, y]

            fact2 = (str(affiche_poid_2) + """\n"""
            + """les poids obtenus entre la couche cachée et la couche de sortie: """ + "
            + str(affiche_poid_3) + """\n""" + """\n"""
```

```
                )

                T.insert(END, fact1 + fact2)

        def affichagePoids():
            fact1 = ("""NMSE minimale: """ + str(np.min(a)) + """\n""")
            fact2 = ("""les poids correspondant sont: """ + """\n""")
            affiche_2 = np.zeros((nombreUniteCache, nombreUniteEntre))
            affiche_3 = np.zeros((1, nombreUniteCache))
            for x in range(nombreUniteCache):
                for y in range(nombreUniteEntre):
                    affiche_2[x, y] = b[np.where(a == np.min(a))[0], x, y]
            for x in range(1):
                for y in range(nombreUniteCache):
                    affiche_3[x, y] = c[np.where(a == np.min(a))[0], x, y]

            fact3 = ("""les poids obtenus entre la couche d' entrée et la couche cachée: """
                     """\n""" + str(affiche_2) + """\n"""
                     + """les poids obtenus entre la couche cachée et la couche de sortie: """ + """\n
                     + str(affiche_3) + """\n"""
                     )

                     T2.insert(END, fact1+fact2+fact3)

            bouton221 = tk.Button(self, text="Information sur l'architecture", command=affichage)
            bouton221.place(x=455, y=510)

            boutonPoid = tk.Button(self, text="POIDS", command=affichagePoids)
            boutonPoid.place(x=455, y=550)

            boutonCourbe = tk.Button(self, text="Courbe", command=courbe_nmse)
            boutonCourbe.place(x=455, y=590)


        def create_label(self, **options):
            tk.Label(self, **options).pack(padx=20, pady=5, anchor=tk.W)
```

Codes de la prédiction à un pas en avant

```
from LesPremieresValeurs import*
import numpy as np
import math

"""poid_1_2 = b[indice_min_loc_nmse]
poid_2_3 = c[indice_min_loc_nmse]"""

W11=0.32754594
W12=-1.18991807
W21=3.79660811
W22=1.76143971

#

Wd11=-0.68517821
Wd12=1.55664188

```

```
18 poid_1_2=np.array([[W11, W12], [W21, W22]])
19 poid_2_3=np.array([[Wd11, Wd12]])
20
21 nombreUniteCache=2
22 nombreUniteEntre=2
23 #g(somme_j(W_ij^m * V_j^m-1))
24
25 def sigmoide(x):
26     return 1 / (1 + math.exp(-x))
27
28 def identite(x):
29     return x
30
31 def predictionUnPasEnAvant():
32     prototype = np.zeros((501, nombreUniteEntre))
33     valeur_predite = np.zeros(500)
34     v_cacher = np.zeros(nombreUniteCache)
35     #creation prototype:
36     for j in range(501):
37         for i in range(nombreUniteEntre):
38             prototype[j, i] = liste_X_n[j + i]
39     #prediction
40     def val_couche_cacher(nbr):
41         val_cacher = np.zeros(nombreUniteCache)
42
43         for j in range(nombreUniteCache):
44             for i in range(nombreUniteEntre):
45                 val_cacher[j] = poid_1_2[j, i] * prototype[nbr, i]
46                 v_cacher[j] = sigmoide(val_cacher[j])
47
48         return v_cacher
49
50     def val_couche_sortie(nbr):
51         v_s_1 = 0
52         for i in range(nombreUniteCache):
53             v_s_1 += poid_2_3[0, i] * val_couche_cacher(nbr)[i]
54         return identite(v_s_1)
55
56     for i in range(500):
57         valeur_predite[i] = val_couche_sortie(i)
58
59     return  valeur_predite, prototype
```

Codes des prédictions à plusieurs pas en avant

```
1 from LesPremieresValeurs import*
2 import numpy as np
3 import math
4
5 W11=0.32754594
6 W12=-1.18991807
7 W21=3.79660811
8 W22=1.76143971
9
10 #
11
```

```
12 Wd11=-0.68517821
13 Wd12=1.55664188
14
15 poid_1_2=np.array([[W11, W12], [W21, W22]])
16 poid_2_3=np.array([[Wd11, Wd12]])
17
18 nombreUniteCache=2
19 nombreUniteEntre=2
20
21 def sigmoide(x):
22     return 1 / (1 + math.exp(-x))
23
24 def identite(x):
25     return x
26
27 def prediction(tab):
28     v_cacher = np.zeros(nombreUniteCache)
29     #prediction
30     def val_couche_cacher():
31         val_cacher = np.zeros(nombreUniteCache)
32
33         for j in range(nombreUniteCache):
34             for i in range(nombreUniteEntre):
35                 val_cacher[j] = poid_1_2[j, i] * tab[i]
36                 v_cacher[j] = sigmoide(val_cacher[j])
37
38         return v_cacher
39
40     def val_couche_sortie():
41         v_s_1 = 0
42         for i in range(2):
43             v_s_1 += poid_2_3[0, i] * val_couche_cacher()[i]
44         return identite(v_s_1)
45
46     predit = val_couche_sortie()
47
48     return predit
49
50 """ la fonction suivante presente 2 paramètre "nbr" et "debut"
51 dont "nbr" represente le nombre de pas d' apprentissage en avant pour
52 la prediction. Et "debut" represente l' indice du premier prototype
53 ou l' on veut commencer. """
54
55 def iteration(nbr, debut):
56     val = np.empty(nbr)
57     prototype = np.empty((nbr, nombreUniteEntre))
58     tabPrototypes = np.zeros(nombreUniteEntre)
59
60     for i in range(nombreUniteEntre):
61         tabPrototypes[i] = liste_X_n[i + debut]
62
63     predit = 0
64     for i in range(nbr):
65         predit = prediction(tabPrototypes)
66         prototype[i] = tabPrototypes
67         #modification de tabPrototypes
```

```
68          tabPrototypes = np.append(predit, tabPrototypes)
69          tabPrototypes = np.delete(tabPrototypes, nombreUniteEntre)
70
71          val[i] = predit
72
73      return val, prototype
```

Titre : Étude dynamique de la série de Hénon par prédiction de la série temporelle générée par la série en utilisant les réseaux de neurones artificiels multicouches.

Résumé :

La série de Hénon est un système dynamique présentant des caractéristiques chaotiques. La prédiction des séries temporelles dérivant de ce système par le biais des réseaux de neurones artificiels permet de contribuer à l'étude dynamique, manifestant le chaos du système. La connaissance de l'architecture du réseau permet de déterminer le modèle optimal permettant la prédiction. Pour cela, l'algorithme de Takens est utilisé pour trouver les nombres d'unités d'entrée, et l'apprentissage du réseau se fait par l'algorithme de descente du gradient. Le langage de programmation utilisé est le "Python". La prédiction à plusieurs pas en avant montre que les erreurs divergent, ce qui montre le caractère chaotique du système.

Mots clés : Réseaux de neurones artificiels, système dynamique, série de Hénon, prédiction, apprentissage

Title : Dynamic study of the Hénon series by prediction of the time series generated by the series using multilayer artificial neural networks

Abstract :

The Hénon series is a dynamical system with chaotic characteristics. The prediction of time series deriving from this system by means of artificial neural networks makes it possible to contribute to the dynamic study, manifesting the choas of the system. The knowledge of the architecture of the network makes it possible to determine the optimal model allowing the prediction. For this, the Takens algorithm is used to find the number of input units, and the learning of the network is done by the gradient descent algorithm. The programming language used is "Python". The multi-step forward prediction shows that the errors diverge, which shows the chaotic nature of the system

Keywords : Artificial neural networks, dynamical system, Hénon series, prediction, learning

Encadreur :
M. RABOANARY Roland
Professeur Titulaire

Impétrant : Ralaikoto Mamitiana Angelo
Téléphone : 034 85 483 69
e-mail : ralaikotoangelo@gmail.com
Addresse : lot IVB 510 Ambohimanala Andoharanofotsy