INSTITUT SUPÉRIEUR POLYTECHNIQUE DE MADAGASCAR

FILIÈRE : INFORMATIQUE DE GESTION, GÉNIE LOGICIEL ET INTELLIGENCE ARTIFICIELLE (IGGLIA)

RESEAU DE NEURONES

PREDICTION ET MODELISATION DES SERIES TEMPORELLES PAR RESEAUX DE NEURONES MULTICOUCHES

Présenté par : RAKOTOARISOA Solofonirina Franck N°38

Dirigé par : Professeur RABOANARY Roland

Année scolaire : 2021-2022

REMERCIEMENT

Tout d'abord, je tiens à remercier Dieu de nous avoir donné l'opportunité de concevoir et de réaliser ce mini-projet.

Je tiens aussi à remercier de nombreuses personnes pour leurs aides à la mise en œuvre de ce mini-projet :

- Le professeur RABOANARY Julien Amédée, Recteur de l'institut supérieur Polytechnique de Madagascar ;
- Le professeur RABOANARY Rolland, Professeur de Réseaux de Neurones Artificiels ;
- Le corps enseignant de l'institut supérieur Polytechnique de Madagascar, qui n'a pas hésité à partager leur connaissance et à offrir leur aide qui nous sont très précieux ;
- Nos familles et nos amis qui nous ont soutenu par leur aide tant moralement que financièrement ;
- Toutes les personnes ayant contribué de près ou de loin à la réalisation de ce projet.

LISTE DES FIGURES

INTRODUCTION

L’intelligence artificielle, c’est l’une des plus grandes branches de l’informatique car il aide à résoudre plusieurs problèmes.

Les réseaux de neurones artificiels sont des algorithmes pour tâches cognitives simulant le traitement de l’information biophysique. Les applications des réseaux multicouches à propagation de l’information vers l’avant à la prédiction de séries temporelles

RESEAUX DE NEURONES ARTIFICIELS

1. Origine neurobiologique des Réseaux de neurones artificiels :

1.1. Le neurone :

Les réseaux de Neurones Artificiels ont pour origine un modèle de neurone biologique dont il ne garde d'ailleurs qu'une vision simplifiée. Le cerveau humain est composé d'un grand nombre de cellules nerveuses appelées neurones, avec 103 à 104 connexions. Un neurone est formé :

- D'un réseau convergent d'entrée appelé dendrites qui constituent la principale surface de réception du neurone.
- D'un élément de traitement appelé corps cellulaire ou soma qui contient le noyau du neurone et la machinerie biochimique nécessaire à la synthèse des enzymes et des autres molécules essentielles à la vie de la cellule.

D'un réseau divergent de sortie appelé axones.

1.2. Traitement de l'information au niveau du cerveau :

Le traitement de l'information par un réseau de neurone est de nature électrochimique. Chaque neurone est asservi au maintien d'un gradient électrique d'environ −70mV entre l'intérieur et l'extérieur du neurone. Ainsi, le neurone est dit polarisé. Si l'influence des autres neurones sur le potentiel membranaire suffit pour dépolariser le neurone jusqu'à un certain seuil (environ $\theta = -50mV$) alors le corps cellulaire génère une impulsion électrique du type tout ou rien appelé potentiel d'action. Le potentiel ainsi généré se propage sans amortissement le long de l'axone et de ses ramifications. Quand le potentiel d'action atteint une synapse reliée à un autre neurone, il déclenche une émission chimique qui modifie le potentiel membranaire du neurone récepteur soit de façons excitatrice(dépolarisation) soit de façons inhibitrice (hyperpolarisation). Ainsi chaque neurone fait en permanence l'addition des signaux excitateurs et dépolarisants de ceux inhibiteurs et polarisants reçus par sa membrane et

déclenche une impulsion nerveuse lorsque les conditions sont réalisées c'est à dire lorsque le potentiel d'action est atteint.

2. Modélisation ou le Neurone formel :

Le neurone formel est la composante principale d'un réseau de neurones artificiels. Ils sont dotés de caractéristiques inspirées de celles des neurones biologiques que nous avons passées en revue dans la section précédente :

- Le potentiel d'action des cellules nerveuses : il s'agit ici d'une valeur numérique, qui peut être transmise à des neurones en aval. Un neurone formel ne peut transmettre qu'une valeur unique qui correspond à son état d'activation.
- Les dendrites des neurones biologiques leur permettent de recevoir différents signaux de l'extérieur. De la même manière, un neurone formel peut recevoir des signaux x_i de plusieurs neurones. Ces signaux sont combinés en un signal d'entrée unique.
- Les synapses : Les nombres w_{ij} pondèrent les signaux émis par les différents neurones situés en amont où l'on retrouve l'analogue des synapses qui peuvent être inhibitrices ($w_{ij} < 0$), ou excitatrices ($w_{ij} > 0$).

Les x_i sont des variables d'entrées, les w_{ij} sont des paramètres des poids.

Les entrées peuvent être binaires (0, 1) ou bipolaire (−1, 1).

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

- Une combinaison linéaire des entrées

$$U = \sum_{i=1}^{n} w_{ij} + x_i$$

- La sortie du neurone

$$y = f(U) = f\left(\sum_{i=1}^{n} w_{ij} + x_i\right)$$

U est appelé potentiel du neurone.

3. Règle d'apprentissage :

Comme le cerveau humain, les réseaux de neurones artificiels peuvent apprendre par expérience. L'apprentissage d'un réseau de neurone artificiel consiste à déterminer les poids entre les neurones, puis modifier la valeur des poids jusqu'à l'obtention du comportement désiré. On distingue deux grandes classes d'algorithme d'apprentissage :

- L'apprentissage supervisé (back propagation)
- L'apprentissage non supervisé

Apprentissage supervisé :

Cet algorithme d'apprentissage ne peut être utilisé que lorsque les combinaisons d'entrées sorties désirées sont connues à l'avance. L'ajustement des poids se fait directement à partir de l'erreur, soit la différence entre la sortie obtenue par le réseau a_r et la sortie désirée c_r. Le cycle est répété jusqu'à ce que le réseau classe correctement les motifs désirés, c'est-à-dire a_r proche de c_r.

Apprentissage non supervisé :

Il n'y a pas de connaissances à priori des sorties désirées pour des entrées données. En fait, c'est de l'apprentissage par exploration où l'algorithme d'apprentissage ajuste les poids des liens entre les neurones de façon à maximiser la qualité de classification des entrées.

4. Les réseaux multicouches à propagation de l'information vers l'avant :

4.1. 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 :

$$E = \frac{1}{2}\sum_{i\mu}\left(\zeta_i^{\mu} - O_i^{\mu}\right)^2$$

Avec :

- ζ_i^{μ} : sortie désirée
- O_i^{μ} : sortie donnée par le réseau

Où :

$$O_i^{\mu} = g\left(h_i^{\mu}\right) = g\left(\sum_j w_{ij}\, V_j^{\mu}\right) = g\left(\sum_j w_{ij}\, g\left(\sum_k w_{jk}\, \zeta_i^{\mu}\right)\right)$$

Alors :

$$E = \frac{1}{2}\sum_{i\mu}\left(\zeta_i^{\mu} - g\left(\sum_j w_{ij}\, g\left(\sum_k w_{jk}\, \zeta_i^{\mu}\right)\right)\right)^2$$

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.

Connexion couche cachée – couche de sortie :

La méthode de descente de gradient donne :

$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$ Avec η le pas d'apprentissage.

Pour calculer $\frac{\partial E}{\partial w_{ij}}$, nous utilisons la règle de chaine :

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial O_i^{\mu}} \frac{\partial O_i^{\mu}}{\partial w_{ij}}$$

Avec :

$$E = \frac{1}{2}\sum_{i\mu}\left(\zeta_i^{\mu} - O_i^{\mu}\right)^2$$

Et :

$$O_i^{\mu} = g\left(\sum_j w_{ij}\, V_j^{\mu}\right)$$

$$\Delta w_{ij} = \eta \sum_{i\mu} (\zeta_i^\mu - O_i^\mu)\, g'(h_i^\mu) V_j^\mu$$

En posant :

$$\delta_i^\mu = g'(h_i^\mu)(\zeta_i^\mu - O_i^\mu)$$

On a finalement :

$$\Delta w_{ij} = \eta \sum_{i\mu} \delta_i^\mu V_j^\mu$$

<u>Connexion couche d'entrée – couche cachée :</u>

Nous avons : $\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}}$

Utilisons de nouveau la règle de chaine :

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial O_i^\mu} \frac{\partial O_i^\mu}{\partial V_i^\mu} \frac{\partial V_i^\mu}{\partial w_{jk}}$$

Nous obtenons :

$$\Delta w_{jk} = \eta \sum_{i\mu} (\zeta_i^\mu - O_i^\mu)\, g'(h_i^\mu)\, w_{ij}\, g'(h_j^\mu)\, \zeta_j^\mu$$

En remarquant que :

$$\delta_i^\mu = g'(h_i^\mu)(\zeta_i^\mu - O_i^\mu)$$

Nous pouvons écrire :

$$\Delta w_{ij} = \eta \sum_{i\mu} \delta_i^\mu\, w_{ij}\, g'(h_j^\mu)\, \zeta_j^\mu$$

Finalement, en posant : $\delta_j^\mu = \sum_{i\mu} \delta_i^\mu\, w_{ij}\, g'(h_j^\mu)$, nous obtenons :

$$\Delta w_{jk} = \eta \sum_{\mu} \delta_j^\mu\, \zeta_k^\mu$$

4.2. Algorithme d'apprentissage par descente de gradient pour réseau multicouches :

- On considère un réseau à M couches.
- Notations :
  - V_i^m : sortie de la $i^{ème}$ $unité$ dans la $m^{ème}$ $couche$.
    - i : indice d'unité.
    - m : indice de couche. $m = 1,2, \dots, M$
  - w_{ij} : poids connexion de V_j^{m-1} à V_i^m.
  - μ : indice de prototype.
  - S'il y a p prototypes $\mu = 1, \dots, p$.
- On fait entrer un prototype à la fois
- Le procédé de propagation est le suivant :
  - Etape 1 : Initialiser les poids à de petites valeurs aléatoires.
  - Etape 2 : Choisir un prototype ξ_k^μ et l'appliquer à la couche d'entrée $(m = 1)$, c'est-à-dire : $V_k^1 = \xi_k^\mu, \forall \boldsymbol{k}$.
  - Etape 3 : Propager le signal vers l'avant à travers le réseau en utilisant :

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

Pour chaque i et m jusqu'à ce que les sorties finales V_i^M aient été toutes calculées.

  - Etape 4 : Calculer les deltas pour la couche de sortie :

$$\delta_i^M = g'(h_i^M)\left(\zeta_i^\mu - V_i^M\right)$$

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

  - Etape 5 : Calculer les deltas pour les couches précédentes en propageant les erreurs vers l'arrières jusqu'à ce qu'un delta ait été calculé pour chaque unité.

$\delta_i^{m-1} = g'(h_i^{m-1}) \sum_j w_{ji}^m \delta_j^m$ Pour $m = \mu, \mu - 1, \dots, 3$

- Etape 6 : Utiliser $\Delta w_{ij}^{m} = \eta\, \delta_i^m\, V_j^{m-1}$ pour mettre à jour toutes les connexions suivantes :

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}$$

APPLICATION LOGISTIQUE ET PREDICTION DES SERIES TEMPORELLES

1. ***Outils et technologies utilisées :***

- *C# :*

J'ai choisi dans notre travail le langage de programmation C-Sharp (C#).

C# est un langage de programmation orienté objet commercialisé par Microsoft depuis 2002, et destiné à développer sur la plateforme Microsoft.NET. J'ai utilisé ce langage, parce qu'il a beaucoup d'avantages par rapport aux autres langages de programmation.

Dans le domaine de la technologie, il est utilisé pour développer des application web, des services web, des applications de bureau et des commandes. Pour cela, nous devons créer tous les codes de programmation en C-Sharp. En C# une application classes comporte une méthode "Main" et possibilité de l'héritage.

Les développements avec des briques logiciels prêtes emploi sont accélérés, le déploiement devient facile, les conflits de versions lors de l'exécution du code pour minimiser et enfin C# fournit un environnement d'exécution de code sécurisé et performant.

On peut dire que, C# est le moteur qui exécute, contrôle et sécurise toutes les applications.

Figure 1 : Logo C-Sharp

- *Microsoft Visual Studio 2019 :*

Pour la conception et la réalisation du mini-projet, j'ai utilisé Microsoft Visual Studio comme éditeur de code.

Microsoft Visual Studio est un ensemble complet d'outils de développement permettant de générer des applications Web ASP.NET, des Services Web XML, des applications bureautiques et des applications mobiles. Visual Basic, Visual C++, Visual C# et Visual J# utilisent tous le même environnement de développement intégré (IDE, Integrated Development Environment), qui leur permet de partager des outils et facilite la création de solutions faisant appel à plusieurs langages.

Par ailleurs, ces langages permettent de mieux tirer parti des fonctionnalités du Framework .NET, qui fournit un accès à des technologies clés simplifiant le développement d'applications Web ASP et de Services Web XML grâce à Visual Web Developer.

Figure 2 : Logo de Visual studio

2. ***Architecture optimale du réseau :***

Pour un problème de prédiction, il est nécessaire d'avoir un réseau optimal car trop peu de paramètres perdraient les informations de la série et trop de paramètres consommeraient beaucoup de temps. Lorsqu'on utilise le Perceptron Multicouches, if faut choisir pour l'architecture optimale du réseau le nombre d'unités d'entrées, le nombre de couches et d'unités cachées, et le nombre d'unités de sortie.

2.1. Nombre de couche cachées :

On utilise entre la couche d'entrée et la première couche cachée et entre les couches cachées elles-mêmes, la fonction sigmoïde et entre la dernière couche cachée et la couche de sortie la fonction identité.

En fait, le théorème de Cybenko montre qu'une seule couche cachée est suffisante pour approche toute fonction continue.

2.2. ***Nombre d'unités d'entrées :***

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

2.2.1. *Algorithme de Takens :*

- Etape 1 : Soit une suite de données (série temporelle) u (1), u (2), …, u(i), …, u(n)
- Etape 2 : Construire une séquence de vecteurs à partir de cette suite (vecteur à n composantes).

$$\dot{x}(i) = \begin{pmatrix} \mu(i) \\ \mu(i + \tau) \\ \mu(i + 2\tau) \\ . \\ . \\ . \\ \mu(i + n\tau) \end{pmatrix}$$

τ : paramètre de délais.

On prend n assez grand.

- Etape 3 : Définir la matrice de covariance à partir de :

$$\theta = < \dot{x}(i)\, \dot{x}(i)^t >$$

- Etape 4 : Calculer les vecteurs propres de θ ainsi que ses valeurs propres λ qui sont rangées par ordre décroissant.

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

- Etape 5 : L'erreur d'approximation moyenne est :

$$\varepsilon_l = \sqrt{\lambda_l + 1}$$

- Etape 6 : Tracer ε_l en fonction de l.
- Etape 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 aussi égal au nombre d'unités d'entrée du réseau de neurones artificiel.

2.2.2. *Algorithme de Jacobi pour la recherche des valeurs propres :*

Matrice de rotation

On appelle matrice de rotation en dimension 2 toutes matrices de la forme :

$$R = \begin{pmatrix} a & -b \\ b & a \end{pmatrix}$$

Avec : $a^2+b^2 = 1$.

Un exemple particulier qui vérifie cette matrice :

$$R^{-1}(\alpha) = \begin{pmatrix} cos\ \alpha & sin\ \alpha \\ -sin\ \alpha & -cos\ \alpha \end{pmatrix}$$

L'intérêt de la matrice de rotation est que le calcul de l'inverse se fait de manière très rapide selon le calcul ci-dessous :

$$R^{-1}(\alpha) = \begin{pmatrix} cos\ \alpha & -sin\ \alpha \\ sin\ \alpha & -cos\ \alpha \end{pmatrix}$$

En dimension n, la forme générale d'une matrice de rotation est la suivante :

$$\begin{array}{c} \\ i \\ \\ j \\ \\ \end{array} \overset{\begin{array}{ccccc} i & & j & & \end{array}}{\begin{pmatrix} 1 & & 0 & & 0 \\ & cos\alpha & \dots & -sin\alpha & \\ & \vdots & \ddots & \vdots & \\ & sin\alpha & \dots & cos\alpha & \\ 0 & \dots & 0 & \dots & 1 \end{pmatrix}}$$

Or une matrice de rotation est une matrice orthogonale et on sait que si P est une matrice orthogonale, et si A est une matrice symétrique donnée, alors la matrice A1=PAP-1 et A sont des matrices similaires ou semblables, c'est-à-dire A1 et A ont exactement les mêmes valeurs propres.

Pour trouver les valeurs propres, nous allons générer une suite de matrices de la manière suivante :

A_0 : Matrice originale

$A_1 = P_0\ A_0\ P_0^{-1}$

$A_2 = P_1\ A_1\ P_1^{-1}$

…

$A_n = P_{n-1}\ A_{n-1}\ P_{n-1}^{-1}$

Or, nous allons construire les matrices P_i comme étant des matrices de rotation, donc une matrice orthogonale ; par conséquent, Les matrices $A_0, A_1, \dots, A_{i+1}$sont des matrices similaires. Donc, ils ont les mêmes valeurs propres.

La construction des matrices P_i doit se faire tel qu'à chaque itération, on génère des 0 en dehors de la diagonale. Théoriquement, lorsque i tends vers l'infini, on obtient une matrice diagonale et les éléments de la diagonale sont exactement les valeurs propres de A.

Pratiquement, on arrête l'itération lorsque les éléments en dehors de la diagonale sont nuls ou presque nuls.

Construction de la matrice P Supposons qu'à la $i^{ème}$ itération, on a :$A_{i+1} = P_i\, A_i\, P_i^t$. Pour pouvoir éliminer les éléments en dehors de la diagonale :

Si $a_{jj}^{(i)} \neq a_{kk}^{(i)}$

$$(P_i)_{jj}=(P_i)_{kk}=\frac{1}{2}\left(1+\frac{b}{\sqrt{c^2+b^2}}\right)$$

$$(P_i)_{kj}=\frac{c}{2(P_i)_{jj}\sqrt{c^2+b^2}}=-(P_i)_{jk}$$

Avec $c= 2a^{(i)}_{jk}(\text{signe}(a^{(i)}_{jj}- a^{(i)}_{kk}))$ et $b=|a_{jj}-a_{kk}|$

Si $a_{jj}^{(i)} = a_{kk}^{(i)}$

$$(P_i)_{jj}=(P_i)_{kk}=\frac{\sqrt{2}}{2}$$

$$(P_i)_{kj}=-(P_i)_{jk}=\frac{\sqrt{2}}{2}$$

Le choix de j et k doit être de telle manière qu'ils correspondent à l'élément $a^{(i)}{}_{jk}$ de A_i ayant la plus grande valeur en valeur absolue. Par conséquent, à chaque itération, il faut exécuter un programme de recherche de max en valeur absolue parmi les éléments de la matrice A_i.

On arrête le programme lorsque les éléments en dehors de la diagonale sont nuls ou presque. (Ex : < 10-3).

2.3. Nombre d'unités de sortie :

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

2.4. Nombre d'unités cachées :

De façon pratique, le nombre d'unités d'entrée et le nombre d'unités de sortie ayant été déterminés 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. Dans notre travail, l'algorithme de Takens donne deux unités d'entrées et l'erreur d'apprentissage la plus faible correspond à deux unités cachées. Alors l'architecture optimale du réseau est de, 2 unités d'entrées, 2 unités cachées et 1 unité de sortie.

3. Applications et résultats :

3.1. Présentations des données :

La série de Henon est obtenue en générant les relations de récurrence suivantes :

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

$$y_{n+1} = bx_n$$

Prendre : $a = 1{,}4\ et\ b = 0{,}3$

1- Générer les 500 premières valeurs de x_n et de y_n en prenant $x_0 = 0\ et\ y_0 = 0$.
2- Tracer y_n en fonction de x_n.
3- Prédictions :
   Faire les prédictions sur la série formée par les valeurs des x_n.
   a. Déterminer l'architecture optimale du réseau permettant de faire les meilleures prédictions.
   b. Faire l'apprentissage du réseau.
   c. Effectuer des prédictions à un pas. (10 valeurs prédites pour les 10 valeurs existantes). Faire figurer sur un même graphe les valeurs prédites et les valeurs attendues.

d. Effectuer des prédictions à 3 pas en avant, à 10 pas en avant et à 20 pas en avant. Faire figurer sur un même graphe les valeurs prédites et les valeurs attendues.

4- Quelles remarques faites-vous sur l'évolution des résultats dans les deux cas de prédictions ?

3.2. ***Extrait de code :***

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;

namespace Franck38rna
{
    public partial class Form1 : Form
    {
        public int NBUE { get; set; }
        public int NBUC { get; set; }
        public static double[] NMSE = new Double[50];
        public classes.SerieDeHenon srh;
        public void Generation()
        {
            srh = new classes.SerieDeHenon(500, 1.4, 0.3);
            double[] dX = srh.serieX;
            double[] dY = srh.serieY;
            NBUE = srh.CalculNUE();
            NBUC = srh.CalculNUC();
            if (chart1.Series.Count != 1)
            {
                while (chart1.Series.Count != 0)
                {
                    chart1.Series.RemoveAt(0);
                }
                chart1.Series.Add("Yn en fonction de Xn");
            }

            chart1.Series[0].Color = Color.Blue;
            chart1.Series[0].ChartType = SeriesChartType.Point;
```

```
        for (int i = 1; i < srh.n; i++)
        {
            listBox1.Items.Add("x" + i + "= " + Math.Round(dX[i], 8));
            listBox2.Items.Add("y" + i + "= " + Math.Round(dY[i], 8));

            chart1.Series[0].Points.AddXY(Math.Round(dX[i], 2), Math.Round(dY[i], 2));
        }

        label9.Text = "" + NBUE + "";
        label10.Text = "" + NBUC + "";
        label11.Text = "" + 1 + "";

    }
    private void Button2_Click(object sender, EventArgs e)
    {
        Generer.Enabled = false;

        double[,,] W = new double[50, 50, 50];
        double nmse;
        String[] wentree = new String[100];
        String[] wsortie = new String[100];
        int k = 0;
        listBox3.Items.Clear();
        listBox4.Items.Clear();

        while (chart1.Series.Count != 0)
        {
            chart1.Series.RemoveAt(0);
        }
        chart1.Series.Add("NMSE");
        int iN = chart1.Series.IndexOf("NMSE");
        chart1.Series[0].Color = Color.Red;
        chart1.Series[0].ChartType = SeriesChartType.Line;


        for (int i = 1; i < 50; i++)
        {
            classes.Apprentissage dg = new classes.Apprentissage(NBUE, i, srh.serieX);
            nmse = classes.Apprentissage.NMSE();
```

```
            NMSE[i] = nmse;
            chart1.Series[0].Points.AddXY(Math.Round(NMSE[i]), i);
        }
        classes.Apprentissage ddg = new classes.Apprentissage(NBUE, NBUC, srh.serieX);
        W = ddg.ApprentissageR();


        for (int i = 1; i <= classes.Apprentissage.NombreUniteCachee; i++)
        {
            for (int j = 1; j <= classes.Apprentissage.NombreUniteEntree; j++)
            {
                wentree[k] = "W(2," + i + "," + j + ")=" + W[2, i, j];
                listBox3.Items.Add(wentree[k]);
                k++;


            }
        }
        k = 0;
        for (int i = 1; i <= 1; i++)
        {
            for (int j = 1; j <= classes.Apprentissage.NombreUniteCachee; j++)
            {
                wsortie[k] = "W(3," + i + "," + j + ")=" + W[3, i, j];
                listBox4.Items.Add(wsortie[k]);
                k++;


            }
        }
    }
    private void Button3_Click_1(object sender, EventArgs e)
    {
        int nb = int.Parse(comboBox1.Text);
        double[] res = new double[100];
        double[,,] W = new double[50, 50, 50];
        double[] s = srh.serieX;
        classes.Apprentissage ddg = new classes.Apprentissage(NBUE, NBUC, s);
        W = ddg.ApprentissageR();
        if (nb == 1)
```

```
            {
                res = ddg.PredictionAUnpas(W);
            }
            else
            {
                res = ddg.PredictionAPlusieursPas(nb, W);
            }
            List<classes.Resultat> list = new List<classes.Resultat>();
            for (int i = 0; i < res.Count(x => x != 0); i++)
            {
                classes.Resultat resultat = new classes.Resultat(i,
classes.Apprentissage.ValeurAttendue[i], Math.Round(res[i], 8));
                list.Add(resultat);
            }
            dataGridView1.DataSource = list;

            DrawChartLs2(res, classes.Apprentissage.ValeurAttendue, list.Count);
        }
        private void DrawChartLs2(double[] Xpredite, double[] Xattendue, int n)
        {
            while (chart1.Series.Count != 0)
            {
                chart1.Series.RemoveAt(0);
            }

            chart1.Series.Add("Prédiction");
            chart1.Series.Add("Attendue");
            chart1.Series[0].ChartType = SeriesChartType.Line;
            chart1.Series[1].ChartType = SeriesChartType.Line;
            chart1.Series[0].Color = Color.Green;
            chart1.Series[1].Color = Color.Blue;
            int iP = chart1.Series.IndexOf("Prédiction");
            int iA = chart1.Series.IndexOf("Attendue");
            for (int i = 0; i < n; i++)
            {
                chart1.Series[iP].Points.AddXY(i + NBUE, Xpredite[i]);
            }
```

```
        for (int i = 0; i < n + NBUE; i++)
        {
            if (i < NBUE)
            {
                chart1.Series[iA].Points.AddXY(i, srh.serieX[i]);
            }
            else
            {
                chart1.Series[iA].Points.AddXY(i, Xattendue[i - NBUE]);
            }
        }
    }
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {

    }

    private void label1_Click(object sender, EventArgs e)
    {

    }

    private void Generer_Click(object sender, EventArgs e)
    {
        Generation();
        btnApprentissage.Enabled = true;
        comboBox1.Enabled = true;
    }

    private void btnApprentissage_Click(object sender, EventArgs e)
    {
        Generer.Enabled = false;
```

```
double[,,] W = new double[50, 50, 50];
double nmse;
String[] wentree = new String[100];
String[] wsortie = new String[100];
int k = 0;
listBox3.Items.Clear();
listBox4.Items.Clear();

while (chart1.Series.Count != 0)
{
    chart1.Series.RemoveAt(0);
}
chart1.Series.Add("NMSE");
int iN = chart1.Series.IndexOf("NMSE");
chart1.Series[0].Color = Color.Red;
chart1.Series[0].ChartType = SeriesChartType.Line;
for (int i = 1; i < 50; i++)
{
    classes.Apprentissage dg = new classes.Apprentissage(NBUE, i, srh.serieX);
    nmse = classes.Apprentissage.NMSE();
    NMSE[i] = nmse;
    chart1.Series[0].Points.AddXY(Math.Round(NMSE[i]), i);
}
classes.Apprentissage ddg = new classes.Apprentissage(NBUE, NBUC, srh.serieX);
W = ddg.ApprentissageR();


for (int i = 1; i <= classes.Apprentissage.NombreUniteCachee; i++)
{
    for (int j = 1; j <= classes.Apprentissage.NombreUniteEntree; j++)
    {
        wentree[k] = "W(2," + i + "," + j + ")=" + W[2, i, j];
        listBox3.Items.Add(wentree[k]);
        k++;
    }
}
k = 0;
for (int i = 1; i <= 1; i++)
{
```

```
            for (int j = 1; j <= classes.Apprentissage.NombreUniteCachee; j++)
            {
                wsortie[k] = "W(3," + i + "," + j + ")=" + W[3, i, j];
                listBox4.Items.Add(wsortie[k]);
                k++;

            }
        }
    }
    private void btnPredict_Click(object sender, EventArgs e)
    {
        int nb = int.Parse(comboBox1.Text);
        double[] res = new double[100];
        double[,,] W = new double[50, 50, 50];
        double[] s = srh.serieX;
        classes.Apprentissage ddg = new classes.Apprentissage(NBUE, NBUC, s);
        W = ddg.ApprentissageR();
        if (nb == 1)
        {
            res = ddg.PredictionAUnpas(W);
        }
        else
        {
            res = ddg.PredictionAPlusieursPas(nb, W);
        }
        List<classes.Resultat> list = new List<classes.Resultat>();
        for (int i = 0; i < res.Count(x => x != 0); i++)
        {
            classes.Resultat resultat = new classes.Resultat(i,
classes.Apprentissage.ValeurAttendue[i], Math.Round(res[i], 8));
            list.Add(resultat);
        }
        dataGridView1.DataSource = list;

        DrawChartLs2(res, classes.Apprentissage.ValeurAttendue, list.Count);
    }
}
```

3.3. ***Résultat :***

3.3.1. *Génération des 500 valeurs :*

La série de Henon est obtenue en générant les relations de récurrence suivantes :

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

$$y_{n+1} = bx_n$$

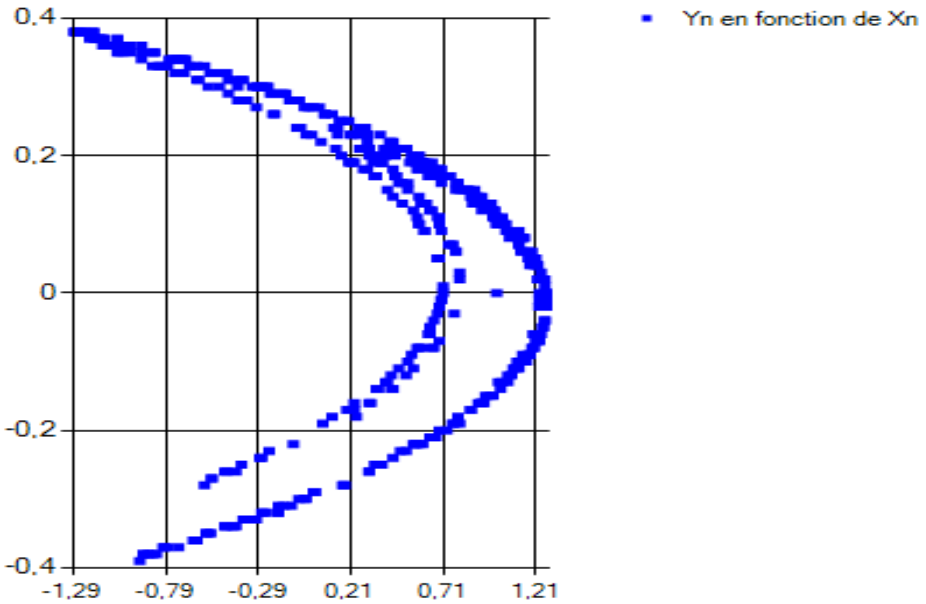
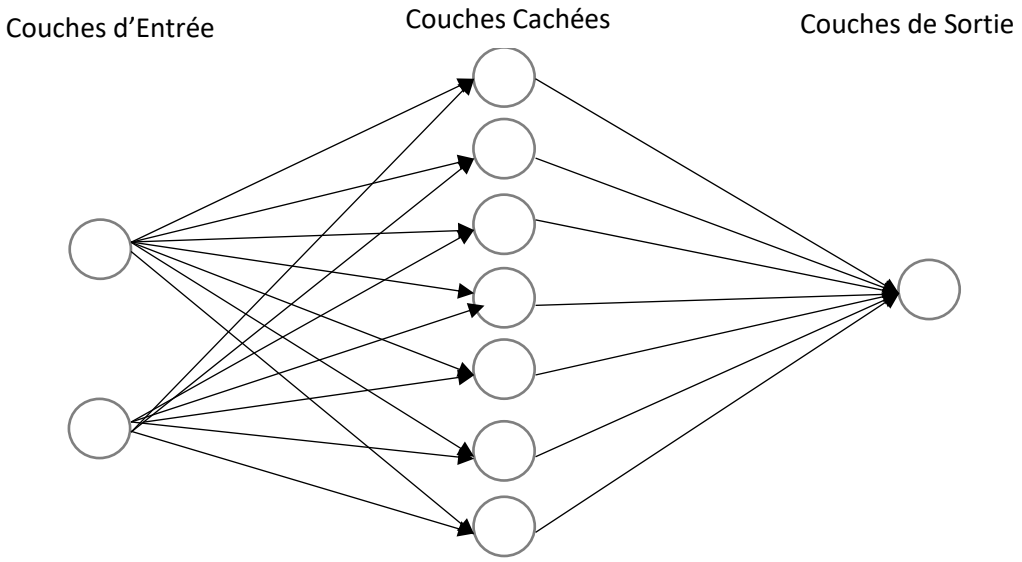


Figure 3 : Yn en fonction de Xn

3.3.2. *Architecture optimale :*

Notre algorithme de Takens donne un nombre d'unité d'entrée égal à deux et d'après les tests que nous avons fait, nous avons l'architecture optimale suivante :



<u>Figure 4 :</u> Architecture optimale (2, 7, 1)

3.3.3. <u>Apprentissage :</u>

Nous avons un réseau optimal (2, 7, 1). On introduit 10 prototypes pour chaque époque. On a pour la première époque :

$$\begin{pmatrix} 1^{\text{ère}}prototype & x(1) & ,x(2) & ,x(3),\dot{x}(3) \\ 2^{\text{ème}}prototype & x(2) & ,x(3) & ,x(4),\dot{x}(4) \\ & \vdots & & \\ 10^{\text{è}}prototype & x(10) & ,x(11) & ,x(12),\dot{x}(12) \end{pmatrix}$$

Avec $x(3) \dots x(12)$: valeurs attendues et $\dot{x}(3) \dots \dot{x}(12)$: valeurs données par le réseau après apprentissage.

Après une époque, on calcule l'erreur quadratique normalisée :

$$NMSE = \frac{1}{N\sigma^2}\sum_{k=1}^{N}\left(x(k) - \dot{x}(k)\right)^2$$

Avec : N : nombre de prototypes,

σ^2 : variance de la série,

$x(k)$: valeur attendue,

$\dot{x}(k)$: valeur de donnée par le réseau.

Les NMSE en fonction de l'époque sont données par la figure suivante :

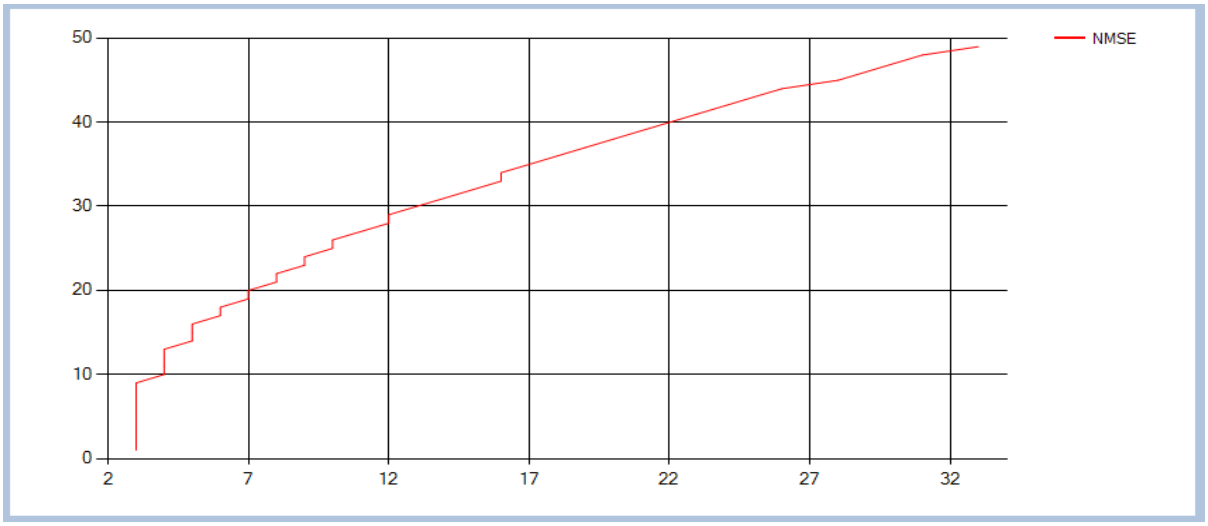


Figure 5 : Erreur quadratique normalisée

3.3.4. Prédictions :

La prédiction d'une série temporelle est composée de 2 phases : Premièrement : l'apprentissage du réseau sur un certain nombre de prototypes. Deuxièmement : la prédiction proprement dite sur les prototypes qui n'ont pas utilisés lors de la phase d'apprentissage.

3.3.4.1. Prédictions à un pas :

Ce type de prédiction consiste à faire la prédiction d'une seule valeur future de la série.

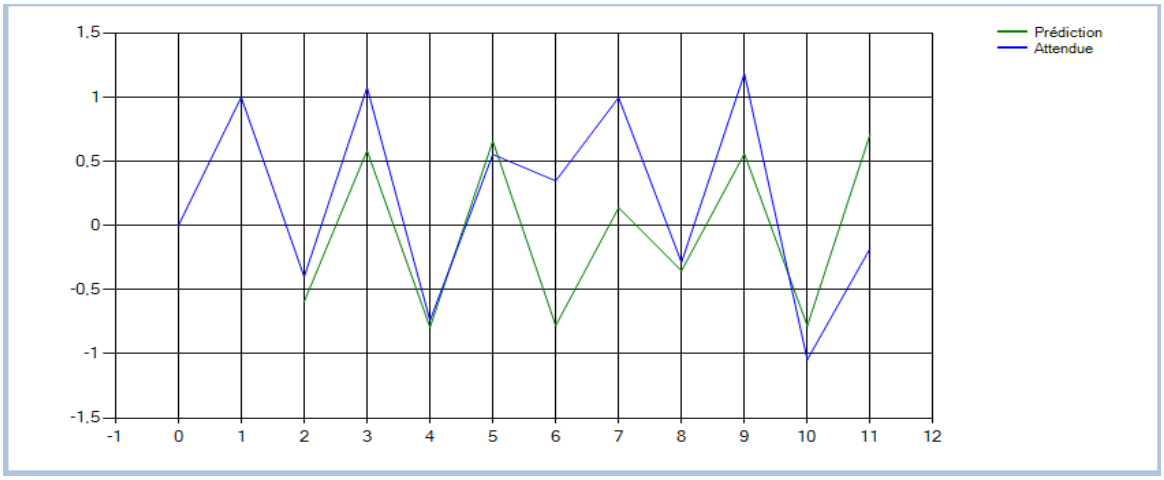


Figure 6 : Prédiction à un pas en avant

Les deux courbes sont presque confondues c'est à dire que, les valeurs attendues et les valeurs prédites coïncident.

3.3.4.2. Pédictions à plusieurs pas en avant :

Le réseau est itéré en bouclage fermé autrement dit la sortie obtenue par le réseau est systématiquement rétro propagée en entrée a` l'itération suivante.

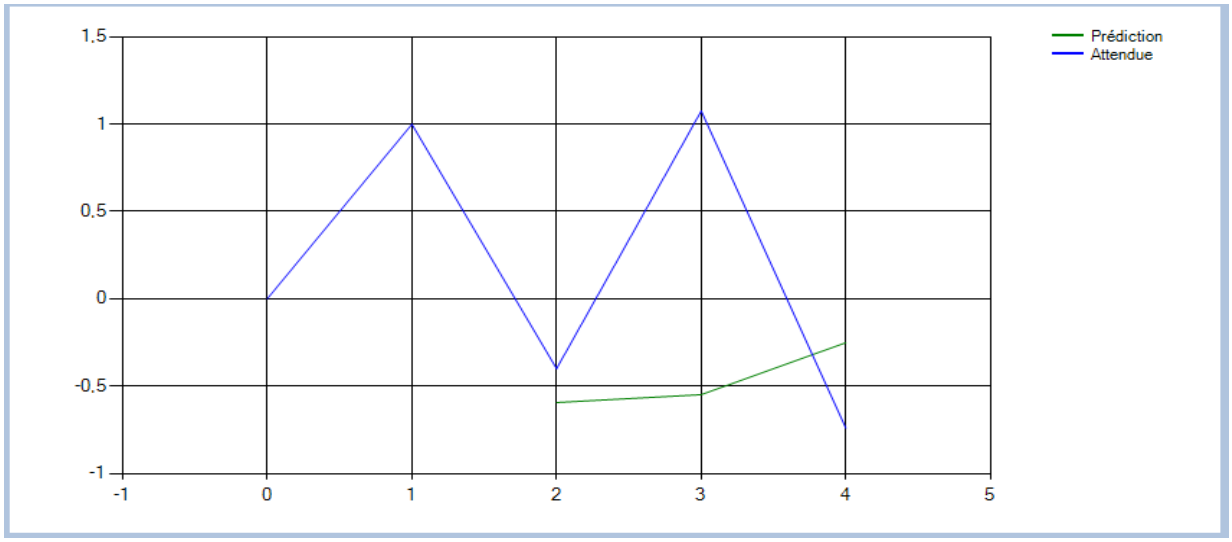


Figure 7 : Prédiction à 3 pas en avant

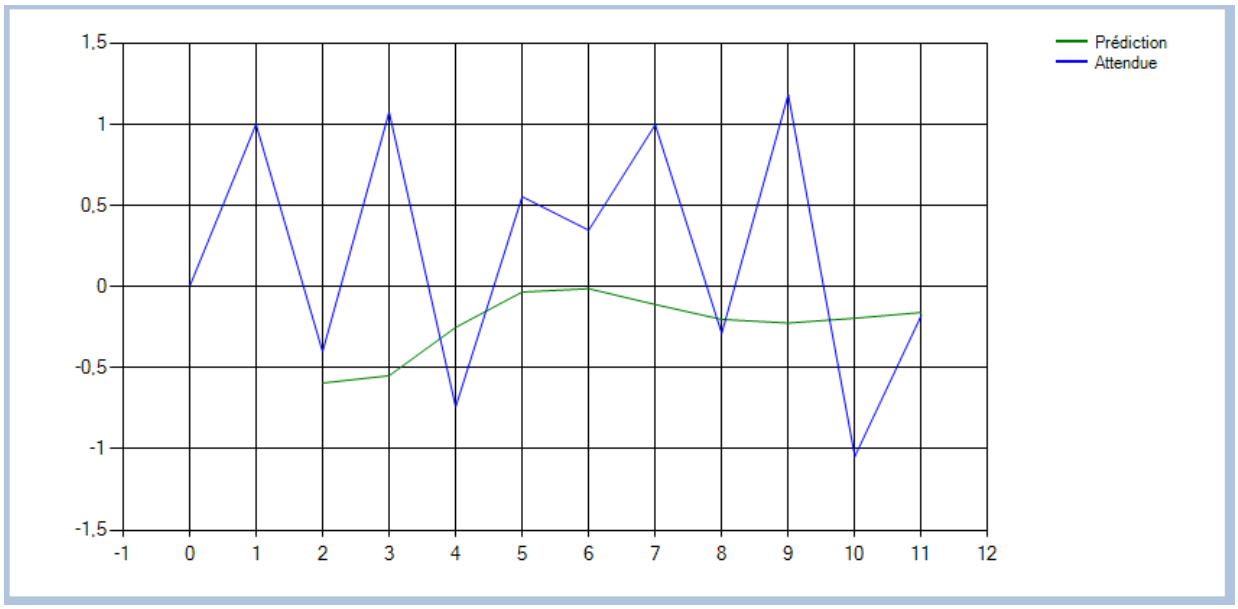
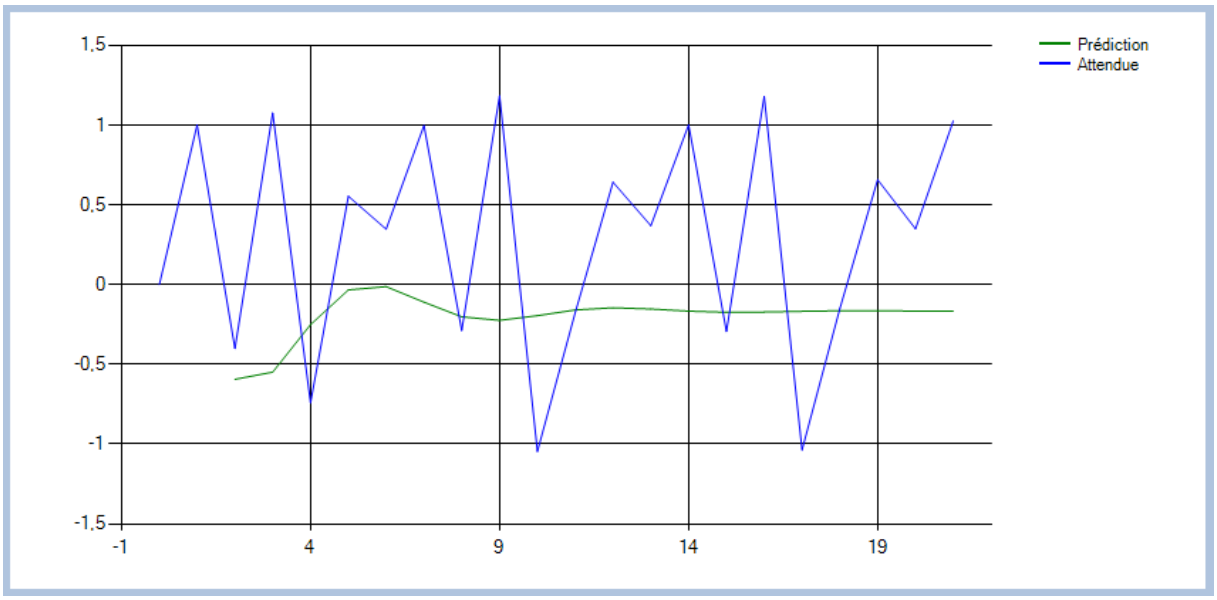


Figure 8 : Prédiction à 10 pas en avant



<u>Figure 9 :</u> Prédiction à 20 pas en avant

Plus on augmente les itérations et, plus les écarts entre les valeurs prédites et valeurs attendues sont élevées.

CONCLUSION

Nous avons pu connaître le déroulement des prédictions à un pas et à plusieurs pas en avant grâce aux réseaux multicouches à propagation de l'information vers l'avant et à rétropropagation de l'erreur.

En regardant les graphes, nous avons pu voir que la prédiction est plus ou moins identiques sur certains points. Plus la prédiction a plusieurs pas en avant, plus la prédiction est précise.

Table des matières

BIBLIOGRAPHIE

1. Cours dispensés :

- Professeur RABOANARY Julien Amédée, Intelligence Artificielle, 4ème année, année universitaire 2021-2022
- Professeur RABOANARY Roland, Réseau de Neurones Artificiels, 4ème année, année universitaire 2021-2022
- Madame Narindra Lova Haingotiana, C#, 3ème année, année universitaire 2020-2021
- Professeur RABOANARY Andry Heriniaina, Algorithmique Avancée, 4ème année 2021-2022

2. Ouvrages :

Manu CARRICANO, Fanny POUJOL. Analyse de données avec SPSS, Paris 2009

3. Webographie :

https://developpez.net/forums/