Projet réseaux de neurones artificiels

Institut des statistiques de Paris (ISUP)

Ousmane LO

Mars 2019

Table des matières

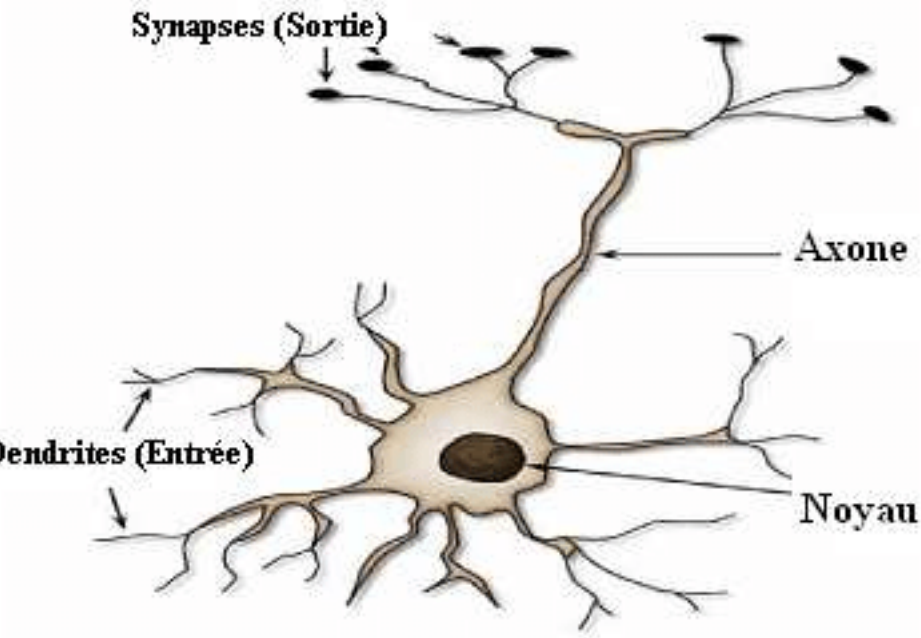


FIGURE 1 – neurone biologique

1 Présentation succincte des réseaux neurones comprenant comment fonctionne l'apprentisage

1.1 Introduction

Les réseaux de neurones artificiels ou réseaux connexionnistes sont fondés sur des modèles qui tentent d'expliquer comment les cellules du cerveau et leurs interconnexions parviennent, d'un point de vue globale, à exécuter des calculs complexes.

Ces systèmes qui stockent et retrouvent l'information de manière "similaire" au cerveau sont particulièrement adaptés aux traitements en parallèle de problèmes complexes comme la reconnaissance automatique de la parole, la reconnaissance de visages ou bien la simulation de fonctions de transfert. Ils offrent donc un nouveau moyen de traitement de l'information utilisé en reconnaissance de formes (vision, image, parole, etc). Les architectures connexionnistes s'inspirent de l'organisation neuronale du cerveau humain.

Dans les réseaux de neurones artificiels de nombreux processeurs appelés cellules ou unités, capables de réaliser des calculs élémentaires, sont structurés en couches successives capables d'échanger des informations au moyen de connexions qui les relient.On dit de ces unités qu'elles miment les neurones biologiques.

1.2 Historique

Le champ des réseaux neuronaux va démarrer par la présentation en 1943 par W. MCCulloch et W. Pitts du neurone formel qui est une abstraction du neurone physiologique. Le retentissement va être énorme. Par cette présentation, ils veulent démontrer que le cerveau est équivalent à une machine de Turing, la pensée devient alors purement des mécanismes matériels et logiques. Il déclara en 1955 "Plus nous apprenons de choses au sujet des organismes, plus nous sommes amenés à conclure qu'ils ne sont pas simplement analogues aux machines, mais qu'ils sont machine." Mysterium Iniquitatis of Sinful Man Aspiring into the Place of God, repris in Embodiments of mind. La démonstration de McCulloch et Pitts sera un des facteurs importants de la création de la cybernétique.

En 1949, D. Hebb présente dans son ouvrage "The Organization of Behavior" une règle d'apprentissage. De

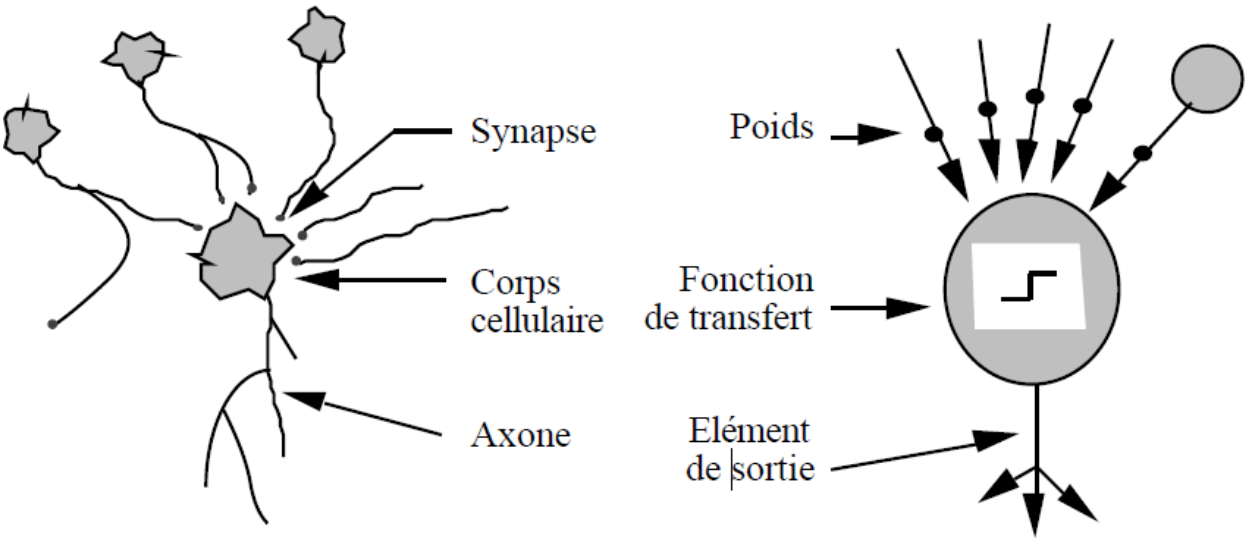


FIGURE 2 –

nombreux modèles de réseaux aujourd'hui s'inpirent encore de la règle de Hebb.

En 1958, F. Rosenblatt développe le modèle du Perceptron. C'est un réseau de neurones inspiré du système visuel. Il posséde deux couches de neurones : une couche de perception et une couche lié à la prise de décision. C'est le premier système artificiel capable d'apprendre par expérience

Dans la même période, Le modèle de L'Adaline (ADAptive LINar Element) a été présenté par B. Widrow, chercheur américain à Stanford. Ce modèe sera par la suite le modèle de base des réseaux multi-couches.

En 1969, M.Minsky et S.Papert publient une critique des propriétés du Perceptron. Cela va avoir une grande incidence sur la recherche dans ce domaine. Elle va fortement diminuer jusqu'en 1972, où T.Kohonen présente ses travaux sur les mémoires associatives. et propose des applications à la reconnaissance de formes.

C'est en 1982 que J. Hopfield présente son étude d'un réseau complétement rebouclé, dont il analyse la dynamique.

1.3 Structure

La figure 2 montre la structure d'un neurone artificiel. Chaque neurone artificiel est un processeur élémentaire. Il reçoit un nombre de variable d'entrées en provenance de neurones amonts. A chacune de ces entrées est associée un poids w abréviation de weight (poids en anglais) représentatif de la force de la connexion. Chaque processeur élémentaire est doté d'une sortie unique, qui se ramifie ensuite pour alimenter un nombre variable de neurones avals. A chaque connexion est associée un poids.

L'architecture du réseau est déterminée par la connectivité (poids w_{ij} nuls ou non nuls), le nombre et le type des neurones. Elle est déterminée par les caractéristiques décrites ci-après.

— Le réseau comporte des boucles s'il existe $i_1, ..., i_p \in [1, N]$ tels que $i_1 = i_p$ et w_{ik} ,$i_k = 0$

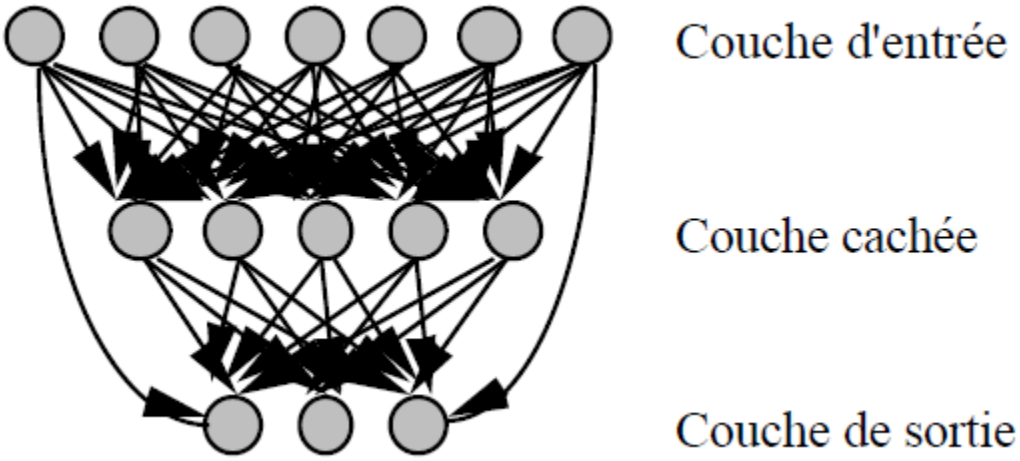
pour $k \in [1, p]$. S'il en est ainsi, le réseau est dit récurrent. Si le réseau est non récurrent alors la matrice des poids W est triangularisable. * Structuration en couches de neurones : connectivité intra-couche.

— Connectivité inter-couches. Supposons que le réseau soit organisé en couches C1, . . .Cp de neurones ; la connectivité inter-couches est complète si pour tout i , $j \in [1, p]$ alors tout neurone de la couche C_i est connecté à chaque neurone de la couche C_j . La connectivité est dite bijective si pour tout i,$j \in [1, p]$ alors tout neurone de la couche Ci est connecté a exactement un unique neurone de la

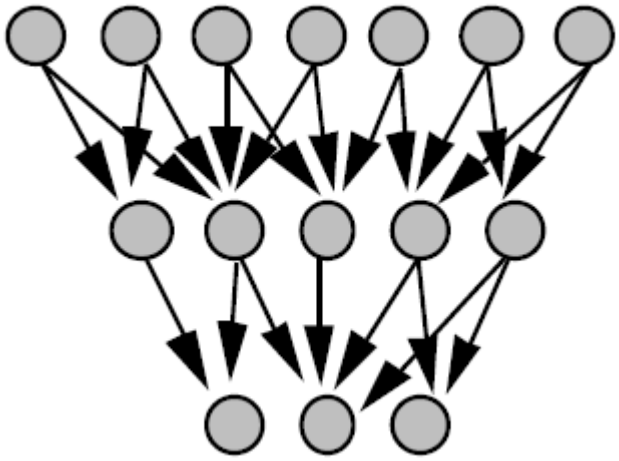
couche C_j . La connectivité est dite probabiliste si elle est distribuée selon une probabilité ou une distribution (gaussienne).

1.3.1 Structure d'interconnexion

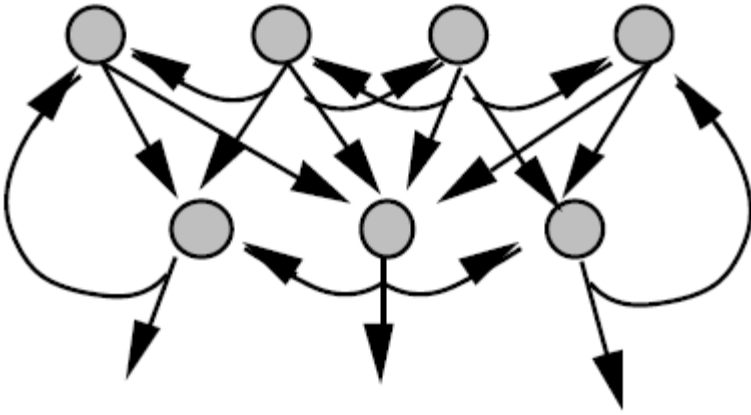
Les connexions entre les neurones qui composent le réseau décrivent la topologie du modèle. Elle peut être quelconque, mais le plus souvent il est possible de distinguer une certaine régularité. * Réseau multicouche (au singulier) : les neurones sont arragés par couche. Il n'y a pas de connexion entre neurones d'une même couche et les connexions ne se font qu'avec les neurones



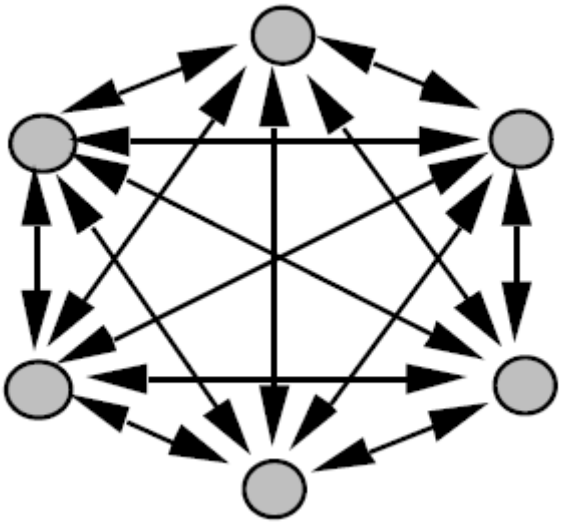
*Réseau à connexions locales : Il s'agit d'une structure multicouche, mais qui à l'image de la rétine, conserve une certaine topologie. Chaque neurone entretien des relations avec un nombre réduit et localisé de neurones de la couche avale (fig. 4). Les connexions sont donc moins nombreuses que dans le cas d'un réseau multicouche classique.



Réseau à connexions récurrentes : les connexions récurrentes ramènent l'information en arrière par rapport au sens de propagation défini dans un réseau multi-



couche. Ces connexions sont le plus souvent locales (fig.5).
Réseau à connexion complète : c'est la structure d'interconnexion la plus générale (fig.6). Chaque neurone est



connecté à tous les neurones du réseau (et à lui-même).
Il existe aussi de nombreuse autres topologies possibles.

1.4 Fonctionement

Le réseau fonctionne selon deux modes :

- en mode de reconnaissance : le réseau est utilisé pour calculer ;
- en mode d'apprentissage : le réseau s'adapte à l'application à l'aide d'exemples ;cette adaptation affecte la matrice des poids W, le nombre de neurones, parfois le "pas d'apprentissage" et très rarement les fonctions d'activation.

1.4.1 Mode de reconnaissance - Propagation de l'activation.

Il s'agit de savoir comment faire fonctionner le réseau. Un cycle de propagation est le calcul de l'activation, en temps discret, de tous les neurones du réseau. La propagation d'un cycle est soit synchrone (i.e. simultanée), soit synchrone par couches (par ex., les perceptrons), soit asynchrone (i.e. séquentielle avec un ordre fixé) soit aléatoire (i.e. séquentielle avec un ordre aléatoire). Dans le cas aléatoire la trajectoire peut être affectée par l'ordre.

- *Exemple. Connexions symétriques (Hopfield, 1982).*

Le réseau symétrique de Hopfield est un réseau symétrique caractérisé ainsi : * il est booléen ; les activations

FIGURE 3 –

des N neurones sont à valeur dans {0,1}, * sa matrice symétrique des poids $W \in \{0,1\}^{N^2}$ est de diagonale nulle : $w_{ii} = 0$,pour tout $i \in [1, N]$, * la fonction d'entrée de chaque neurone $i \in [1, N]$ à l'instant t est définie par : $h(i,t) = \sum_{j=1}^{N} w_{ij} a_j(t)$ * la fonction d'activation commune à tous les neurones du réseau est ainsi définie de N dans {0,1} :

$$f(x) = \begin{cases} 1 & si \quad x > 0 \\ 0 & sinon \end{cases}$$

Pour chaque instant t 2N, nous notons $a_i(t)$ l'activation du neurone i , $e_i(t) = h(i,t)$ son activation pondéré et $a(t)$ le vecteur d'activation du réseau. L'équation du réseau symétrique de Hopfield est :$a_i(t+1) = f(e_i(t))$ où $e_i(t) = \sum_{j=1}^{N} w_{ij} a_j(t)$

1.5 Apprentisage

1.5.1 Définition

L 'apprentissage est une phase du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré. On distingue deux grandes classes d'algorithmes d'apprentissage :

- L'apprentissage supervisé
- L'apprentissage non supervisé
- apprentissage supervisé : les coefficients synaptiques sont évalués en minimisant l 'erreur (entre sortie souhaitée et sortie obtenue) sur une base d'apprentissage.
- apprentissage non-supervisé : on ne dispose pas de base d 'apprentissage. Les coefficients synaptiques sont déterminés par rapport à des critères de conformité : spécifications générales.
- sur-apprentissage : on minimise l'erreur sur la base d 'apprentissage à chaque itération mais on augmente l 'erreur sur la base d 'essai. Le modèle perd sa capacité de généralisation : c 'est l 'apprentissage par cœur.

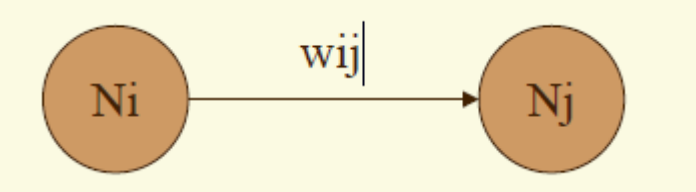
1.5.2 Règles d'apprentissage

L'apprentissage consiste à modifier le poids des connections entre les neurones.

Il existe plusieurs règles de modification :

- Loi de Hebb : $\Delta a w_{ij} = R a_i a_j$
- Règle de Widrow-Hoff (delta rule) : $\Delta a w_{ij} = R(d_i - a_i) a_j$
- Règle de Grossberg : $\Delta a wij = R(a_j - w_{ij}) a_i$
- *Loi de Hebb :*
- n entrées $e_1, \ldots, e_n$
- m neurones $N_1, \ldots, N_m$.
- wij le coefficient synaptique de la liaison entre les neurones Ni et Nj
- une sortie o
- un seuil S
- Fonction de transfert : fonction Signe

$$sign(x) = \begin{cases} 1 & si \quad x > 0 \\ -1 & sinon \end{cases}$$



Principe : Si deux neurones sont activés en même temps, alors la force de connexion augmente.

Base d 'apprentissage : On note S la base d 'apprentissage.

S est composée de couples (e, c) où :

e est le vecteur associé à l 'entrée $(e_1, \ldots, e_n)$

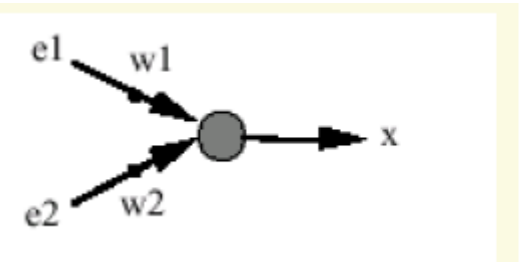
c la sortie correspondante souhaitée

a_i est la valeur d 'activation du neurone Ni.

Algorithme :

- µ est une constante positive.
- Initialiser aléatoirement les coefficients w_i
- Répéter :
- Prendre un exemple (e, c) dans S
- Calculer la sortie o du réseau pour l 'entrée e
- Si $c \neq o$
- Modification des poids w_{ij} :
- $w_{ij} = w_{ij} + \mu \times (a_i \times a_j)$
- Fin Pour
- Fin Si
- Fin Répéter

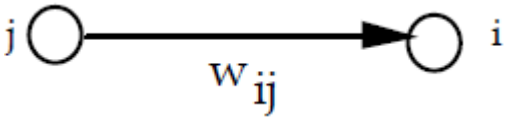
Exemple :



e_1	e_2	x	
1	1	1	(1)
1	-1	1	(2)
-1	1	-1	(3)
-1	-1	-1	(4)

$\mu = 1$ Conditions initiales : coefficients et seuils nuls

- *Loi de Widrow-Hoff (delta rule) :* a_i activation produite par le réseau d_i réponse désirée par l'expert humain Par exemple si la sortie est inférieure à la réponse désirée, il va falloir augmenter le poids de la connexion à condition bien sûr que l'unité j soit excitatrice (égale à 1). On est dans l'hypothèse d'unités



booléennes {0,1}. $a_i = 0$ $a_i = 1$

$d_i = 0$ $\Delta w_{ij} = 0$ $\Delta w_{ij} = -R$

$d_i = 1$ $\Delta w_{ij} = R$ $\Delta w_{ij} = 0$

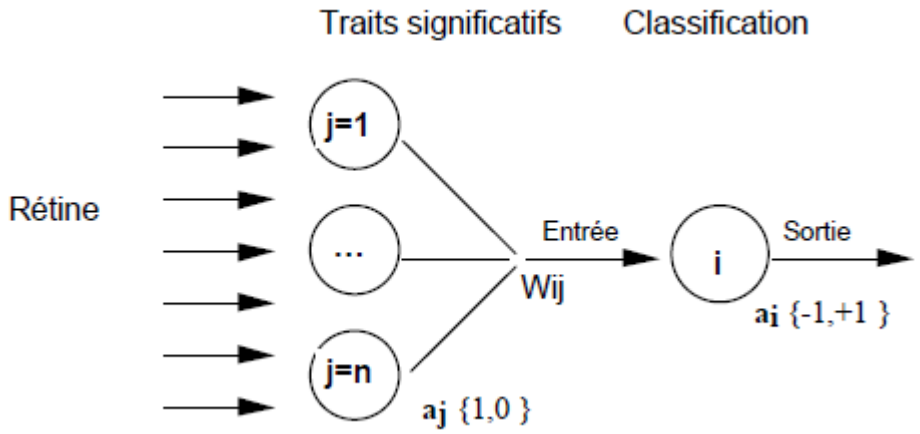
$\Delta w_{ij} = R(d_i - a_i)a_j$

**Loi de Grossberg :* On augmente les poids qui entrent sur l'unité gagnante a_i s'ils sont trop faibles, pour les rapprocher du vecteur d'entrée aj. C'est la règle d'apprentissage utilisée dans les cartes auto-organisatrices

de Kohonen. $\Delta w_{ij} = Ra_i(a_j - w_{ij})$

1.6 Le perceptron

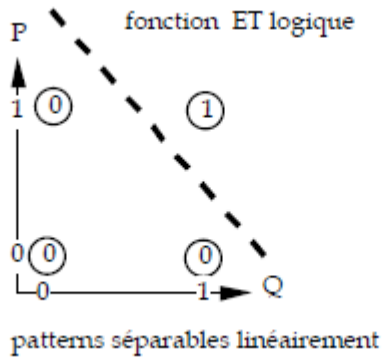
Le perceptron de Rosenblatt (1957) est le premier RNA opérationnel. C'est un réseau à propagation avant avec seulement deux couches (entrée et sortie) entièrement interconnectées.Il est composé de neurones à seuil. L'apprentissage est supervisé et les poids sont modifiés selon la règle delta.



Le perceptron est incapable de distinguer les patterns non séparables linéairement.

	ET				OU				OU Exclusif		
e1	e2		x		e1	e2		x		e1	e2	x
1	1		1		1	1		1		1	1	
1	-1		-1		1	-1		1		-1	-1	
-1	1		-1		-1	1		-1		1	-1	
-1	-1		-1		-1	-1		-1		-1	1	

input P	0	1	0	1
input Q	1	0	0	1
ET	**0**	**0**	**0**	**1**
XOR	**1**	**1**	**0**	**0**



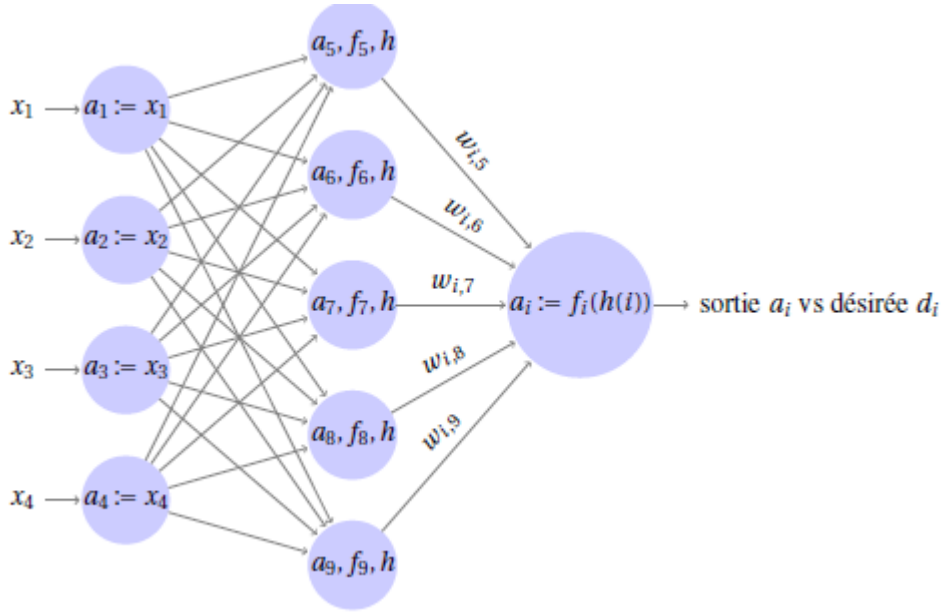


FIGURE 4 –

2 Utilisation de R pour le PMC avec avec comparaison avec une ou des autres méthodes statistiques

2.1 Perceptron multicouche(PMC)

2.1.1 Définition

Les PMC sont des "approximateurs universels de fonctions". Un PMC, comportant une couche cachée avec une fonction de transfert non linéaire et une couche de sortie linéaire, suffit à approximer toute fonction continue à support borné . Ses applications sont nombreuses : reconnaissance des formes, la classification, le contrôle automatique de processus.

2.1.2 Architecture

Le PMC se présente en 3 (ou plus) couches : la couche d'entrée, une couche cachée (ou des couches cachées) et la couche de sortie.

L'activation se propage de l'entrée vers la couche cachée puis de la couche cachée vers la couche de sortie. La fonction d'entrée du neurone i est linéaire pour la couche cachée et la couche de sortie : $h(i) = \sum_{j=1}^{n} w_{ij} a_j$

et la fonction d'activation fi pour la couche cachée est une sigmoïde afin que les couches cachées soient utiles (voir le paragraphe sur le perceptron) et que l'on puisse dériver. La couche de sortie est plutôt linéaire (i.e. $f_i = id$).

2.1.3 Apprentissage

La règle de modification des poids reste $\Delta w_{ij} = -\lambda \frac{\partial E}{\partial w_{ij}}$ où j désigne donc un neurone de la couche précédente à celle du neurone i . La fonction d'erreur E calculée sur les neurones de la couche de sortie S est identique à celle du perceptron, c'est l'erreur quadratique : $E = \frac{1}{2} \sum_{i \in s} (a_i - d_j)^2$

2.1.4 Application

```
library(nnet)
library(class)
library(MASS)
library(mlbench)
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following object is masked from 'package:magrittr':
##
##     extract
```

```
library(magrittr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:MASS':
##
##     select
```

```
## The following object is masked from 'package:kableExtra':
##
##     group_rows
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
data=read.csv("iris1.csv",sep = ",",dec=",", fileEncoding="CP1252")
data=data[,-1]

data[data== "" ] <- NA
data<-data %>% drop_na()
#data<-data%>%
           # mutate_if(sapply(data, is.factor), as.numeric)
appindex = sample(1 :nrow(data), round(2*nrow(data)/3), replace = FALSE)
app = data[appindex, ]
val=data[-appindex, ]
app
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 108          7.3         2.9          6.3         1.8  virginica
## 112          6.4         2.7          5.3         1.9  virginica
## 55           6.5         2.8          4.6         1.5 versicolor
## 103          7.1           3          5.9         2.1  virginica
## 30           4.7         3.2          1.6         0.2     setosa
## 54           5.5         2.3            4         1.3 versicolor
## 143          5.8         2.7          5.1         1.9  virginica
## 139            6           3          4.8         1.8  virginica
```

```
## 150          5.9          3          5.1        1.8  virginica
## 58           4.9        2.4          3.3          1 versicolor
## 45           5.1        3.8          1.9        0.4     setosa
## 149          6.2        3.4          5.4        2.3  virginica
## 23           4.6        3.6            1        0.2     setosa
## 35           4.9        3.1          1.5        0.2     setosa
## 97           5.7        2.9          4.2        1.3 versicolor
## 123          7.7        2.8          6.7          2  virginica
## 90           5.5        2.5            4        1.3 versicolor
## 17           5.4        3.9          1.3        0.4     setosa
## 117          6.5          3          5.5        1.8  virginica
## 147          6.3        2.5            5        1.9  virginica
## 2            4.9          3          1.4        0.2     setosa
## 118          7.7        3.8          6.7        2.2  virginica
## 21           5.4        3.4          1.7        0.2     setosa
## 144          6.8        3.2          5.9        2.3  virginica
## 16           5.7        4.4          1.5        0.4     setosa
## 13           4.8          3          1.4        0.1     setosa
## 79             6        2.9          4.5        1.5 versicolor
## 56           5.7        2.8          4.5        1.3 versicolor
## 38           4.9        3.6          1.4        0.1     setosa
## 86             6        3.4          4.5        1.6 versicolor
## 6            5.4        3.9          1.7        0.4     setosa
## 28           5.2        3.5          1.5        0.2     setosa
## 7            4.6        3.4          1.4        0.3     setosa
## 49           5.3        3.7          1.5        0.2     setosa
## 99           5.1        2.5            3        1.1 versicolor
## 129          6.4        2.8          5.6        2.1  virginica
## 122          5.6        2.8          4.9          2  virginica
## 73           6.3        2.5          4.9        1.5 versicolor
## 130          7.2          3          5.8        1.6  virginica
## 89           5.6          3          4.1        1.3 versicolor
## 39           4.4          3          1.3        0.2     setosa
## 116          6.4        3.2          5.3        2.3  virginica
## 32           5.4        3.4          1.5        0.4     setosa
## 95           5.6        2.7          4.2        1.3 versicolor
## 5              5        3.6          1.4        0.2     setosa
## 111          6.5        3.2          5.1          2  virginica
## 98           6.2        2.9          4.3        1.3 versicolor
## 140          6.9        3.1          5.4        2.1  virginica
## 26             5          3          1.6        0.2     setosa
## 77           6.8        2.8          4.8        1.4 versicolor
## 82           5.5        2.4          3.7          1 versicolor
## 72           6.1        2.8            4        1.3 versicolor
## 67           5.6          3          4.5        1.5 versicolor
## 66           6.7        3.1          4.4        1.4 versicolor
## 46           4.8          3          1.4        0.3     setosa
## 84             6        2.7          5.1        1.6 versicolor
## 106          7.6          3          6.6        2.1  virginica
## 53           6.9        3.1          4.9        1.5 versicolor
## 71           5.9        3.2          4.8        1.8 versicolor
## 124          6.3        2.7          4.9        1.8  virginica
## 120            6        2.2            5        1.5  virginica
## 57           6.3        3.3          4.7        1.6 versicolor
```

```
## 113          6.8           3          5.5         2.1  virginica
## 48           4.6         3.2          1.4         0.2     setosa
## 51             7         3.2          4.7         1.4 versicolor
## 141          6.7         3.1          5.6         2.4  virginica
## 92           6.1           3          4.6         1.4 versicolor
## 1            5.1         3.5          1.4         0.2     setosa
## 110          7.2         3.6          6.1         2.5  virginica
## 40           5.1         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 69           6.2         2.2          4.5         1.5 versicolor
## 59           6.6         2.9          4.6         1.3 versicolor
## 100          5.7         2.8          4.1         1.3 versicolor
## 4            4.6         3.1          1.5         0.2     setosa
## 115          5.8         2.8          5.1         2.4  virginica
## 19           5.7         3.8          1.7         0.3     setosa
## 15           5.8           4          1.2         0.2     setosa
## 94             5         2.3          3.3           1 versicolor
## 36             5         3.2          1.2         0.2     setosa
## 126          7.2         3.2            6         1.8  virginica
## 65           5.6         2.9          3.6         1.3 versicolor
## 145          6.7         3.3          5.7         2.5  virginica
## 128          6.1           3          4.9         1.8  virginica
## 136          7.7           3          6.1         2.3  virginica
## 27             5         3.4          1.6         0.4     setosa
## 50             5         3.3          1.4         0.2     setosa
## 75           6.4         2.9          4.3         1.3 versicolor
## 43           4.4         3.2          1.3         0.2     setosa
## 63             6         2.2            4           1 versicolor
## 74           6.1         2.8          4.7         1.2 versicolor
## 68           5.8         2.7          4.1           1 versicolor
## 83           5.8         2.7          3.9         1.2 versicolor
## 18           5.1         3.5          1.4         0.3     setosa
## 24           5.1         3.3          1.7         0.5     setosa
## 34           5.5         4.2          1.4         0.2     setosa
## 47           5.1         3.8          1.6         0.2     setosa
## 20           5.1         3.8          1.5         0.3     setosa
## 133          6.4         2.8          5.6         2.2  virginica
## 52           6.4         3.2          4.5         1.5 versicolor
```

```
model.dis =nnet(Species~.,data = app,size = 2,decay = 0.1)
```

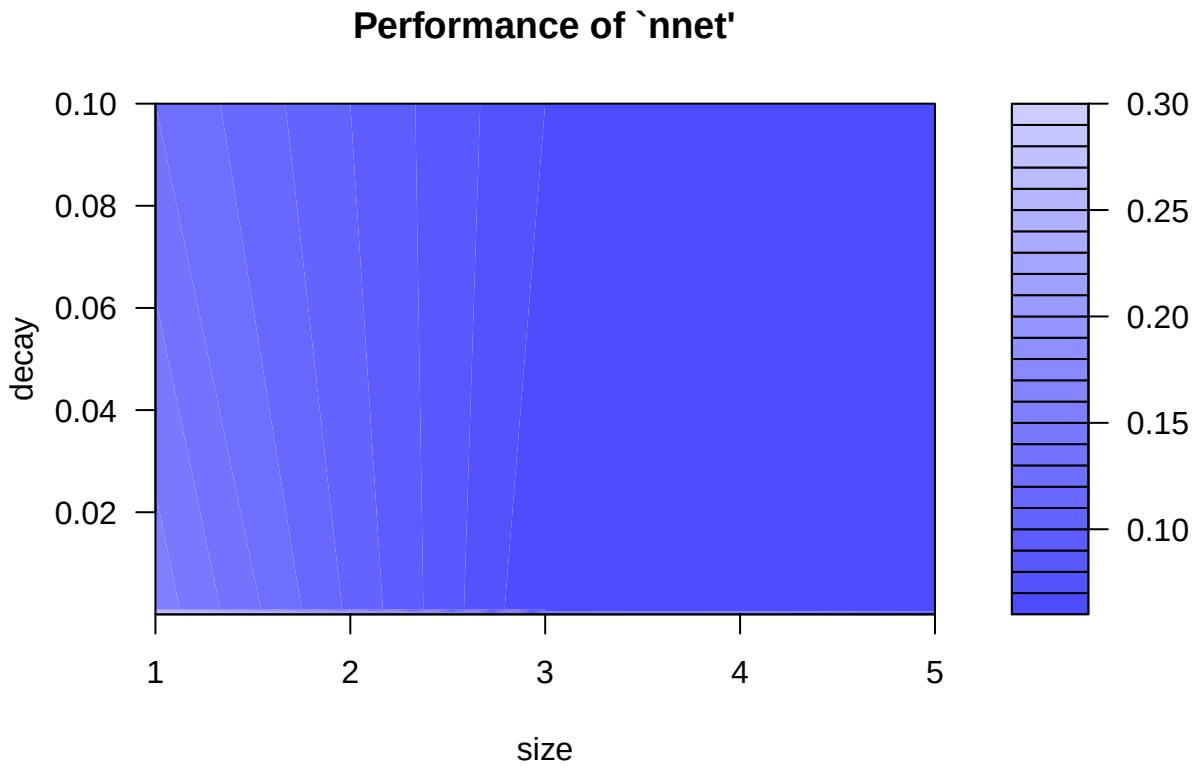
```
## # weights:  249
## initial  value 132.577258
## iter  10 value 55.129175
## iter  20 value 38.350207
## iter  30 value 23.105650
## iter  40 value 21.335173
## iter  50 value 21.205735
## iter  60 value 20.923394
## iter  70 value 20.768836
## iter  80 value 20.734951
## final  value 20.734613
## converged
```

```
summary(model.dis)
```

```
## a 119-2-3 network with 249 weights
## options were - softmax modelling  decay=0.1
##    b->h1   i1->h1   i2->h1   i3->h1   i4->h1   i5->h1   i6->h1   i7->h1
##    -0.26    -0.33     0.00    -0.33    -0.11    -0.66    -0.39    -0.49
##   i8->h1   i9->h1  i10->h1  i11->h1  i12->h1  i13->h1  i14->h1  i15->h1
##    -0.76    -0.07    -0.08    -0.44    -0.12    -0.03    -0.42     0.24
##  i16->h1  i17->h1  i18->h1  i19->h1  i20->h1  i21->h1  i22->h1  i23->h1
##    -0.56     0.24    -0.15     0.13     0.21     0.53     0.27    -0.01
##  i24->h1  i25->h1  i26->h1  i27->h1  i28->h1  i29->h1  i30->h1  i31->h1
##     0.54     0.33     0.00    -0.02     0.14     0.95     0.32     0.00
##  i32->h1  i33->h1  i34->h1  i35->h1  i36->h1  i37->h1  i38->h1  i39->h1
##     0.20     0.60     0.00     0.49    -0.03    -0.09    -0.18     0.00
##  i40->h1  i41->h1  i42->h1  i43->h1  i44->h1  i45->h1  i46->h1  i47->h1
##    -0.03     0.44    -0.04     0.63    -0.02    -0.11    -0.01    -0.16
##  i48->h1  i49->h1  i50->h1  i51->h1  i52->h1  i53->h1  i54->h1  i55->h1
##    -0.15    -0.14    -0.08    -0.13    -0.27    -0.20     0.00    -0.04
##  i56->h1  i57->h1  i58->h1  i59->h1  i60->h1  i61->h1  i62->h1  i63->h1
##    -0.14     0.00    -0.29    -0.45    -1.27    -0.69    -0.48    -0.66
##  i64->h1  i65->h1  i66->h1  i67->h1  i68->h1  i69->h1  i70->h1  i71->h1
##    -0.18    -0.03    -0.05     0.00    -0.05    -0.06     0.00    -0.23
##  i72->h1  i73->h1  i74->h1  i75->h1  i76->h1  i77->h1  i78->h1  i79->h1
##    -0.39    -0.32    -0.06    -0.21    -0.24    -0.51    -0.41    -0.30
##  i80->h1  i81->h1  i82->h1  i83->h1  i84->h1  i85->h1  i86->h1  i87->h1
##    -0.49     0.37     1.17     0.57     0.00     0.52     0.76     0.26
##  i88->h1  i89->h1  i90->h1  i91->h1  i92->h1  i93->h1  i94->h1  i95->h1
##     0.62     0.43     0.53     0.51     0.15     0.35     0.32     0.00
##  i96->h1  i97->h1  i98->h1  i99->h1 i100->h1 i101->h1 i102->h1 i103->h1
##     0.20     0.52     0.00    -1.61    -0.64    -0.88    -0.29     0.00
## i104->h1 i105->h1 i106->h1 i107->h1 i108->h1 i109->h1 i110->h1 i111->h1
##    -0.52    -0.03    -0.38    -0.65    -0.60    -0.50    -0.51     0.00
## i112->h1 i113->h1 i114->h1 i115->h1 i116->h1 i117->h1 i118->h1 i119->h1
##     1.25     0.92     0.96     1.00     0.60     1.03     0.65     0.70
##    b->h2   i1->h2   i2->h2   i3->h2   i4->h2   i5->h2   i6->h2   i7->h2
##    -0.28    -0.27     0.00    -0.33    -0.16    -0.29    -0.18    -0.21
##   i8->h2   i9->h2  i10->h2  i11->h2  i12->h2  i13->h2  i14->h2  i15->h2
##    -0.33    -0.07    -0.08    -0.41     0.23     0.40    -0.01    -0.06
##  i16->h2  i17->h2  i18->h2  i19->h2  i20->h2  i21->h2  i22->h2  i23->h2
##     0.84     0.18     0.54     0.24     0.09    -0.12     0.05     0.07
##  i24->h2  i25->h2  i26->h2  i27->h2  i28->h2  i29->h2  i30->h2  i31->h2
##     0.00    -0.01     0.27     0.15    -0.01    -0.43    -0.20     0.00
##  i32->h2  i33->h2  i34->h2  i35->h2  i36->h2  i37->h2  i38->h2  i39->h2
##    -0.03    -0.13     0.00    -0.25     0.34     0.41     0.79     0.00
##  i40->h2  i41->h2  i42->h2  i43->h2  i44->h2  i45->h2  i46->h2  i47->h2
##     0.48     0.14     0.44    -0.64     0.27     0.21    -0.19    -0.31
##  i48->h2  i49->h2  i50->h2  i51->h2  i52->h2  i53->h2  i54->h2  i55->h2
##    -0.20    -0.43    -0.08    -0.58    -0.27    -0.15     0.00    -0.07
##  i56->h2  i57->h2  i58->h2  i59->h2  i60->h2  i61->h2  i62->h2  i63->h2
##    -0.19     0.00    -0.32    -0.37    -1.07    -0.76    -0.45    -0.69
##  i64->h2  i65->h2  i66->h2  i67->h2  i68->h2  i69->h2  i70->h2  i71->h2
##    -0.17     0.51     0.49     0.00     0.11     0.19     0.00     0.48
##  i72->h2  i73->h2  i74->h2  i75->h2  i76->h2  i77->h2  i78->h2  i79->h2
##     0.59     0.61     0.24     0.31     0.37     1.02     0.70     0.72
```

```
##  i80->h2  i81->h2  i82->h2  i83->h2  i84->h2  i85->h2  i86->h2  i87->h2
##     0.81    -0.08    -0.97    -0.13     0.00    -0.29    -0.39    -0.05
##  i88->h2  i89->h2  i90->h2  i91->h2  i92->h2  i93->h2  i94->h2  i95->h2
##    -0.38    -0.18    -0.33    -0.23    -0.07    -0.04    -0.20     0.00
##  i96->h2  i97->h2  i98->h2  i99->h2 i100->h2 i101->h2 i102->h2 i103->h2
##    -0.03    -0.12     0.00    -1.59    -0.60    -0.92    -0.34     0.00
## i104->h2 i105->h2 i106->h2 i107->h2 i108->h2 i109->h2 i110->h2 i111->h2
##     1.23     0.51     0.68     1.45     1.11     1.08     1.06     0.00
## i112->h2 i113->h2 i114->h2 i115->h2 i116->h2 i117->h2 i118->h2 i119->h2
##    -0.53    -0.60    -0.66    -0.44    -0.18    -0.43    -0.40    -0.21
##  b->o1 h1->o1 h2->o1
##   2.82  -3.57  -3.57
##  b->o2 h1->o2 h2->o2
##  -1.41  -1.51   5.18
##  b->o3 h1->o3 h2->o3
##  -1.40   5.08  -1.61
```

```
library(e1071)
tune.model = tune.nnet (Species~., data = app, size = c (1, 3, 5),decay = c (0.1, 0.001, 0.000001))
plot(tune.model)
```




```
tune.model
```

```
##
## Parameter tuning of 'nnet':
##
## - sampling method: 10-fold cross validation
```

```
##
## - best parameters:
##  size decay
##     3 0.001
##
## - best performance: 0.06
```

```
pred=predict(model.dis,newdata=val[,-5],type="class")
mat = table(pred, val[, 5])
taux1 = sum(diag(mat))/sum(mat)
taux1
```

```
## [1] 0.9
```

2.2 Méthode des k plus proches voisins(knn)

2.2.1 Définition

En intelligence artificielle, la méthode des k plus proches voisins est une méthode d'apprentissage supervisé. En abrégé k-NN ou KNN, de l'anglais k-nearest neighbors.

Dans ce cadre, on dispose d'une base de données d'apprentissage constituée de N couples « entrée-sortie ». Pour estimer la sortie associée à une nouvelle entrée x, la méthode des k plus proches voisins consiste à prendre en compte (de façon identique) les k échantillons d'apprentissage dont l'entrée est la plus proche de la nouvelle entrée x, selon une distance à définir.

Par exemple, dans un problème de classification, on retiendra la classe la plus représentée parmi les k sorties associées aux k entrées les plus proches de la nouvelle entrée x.

En reconnaissance de forme, l'algorithme des k plus proches voisins (k-NN) est une méthode non paramétrique utilisée pour la classification et la régression. Dans les deux cas, il s'agit de classer l'entrée dans la catégorie à laquelle appartient les k plus proches voisins dans l'espace des caractéristiques identifiées par apprentissage. Le résultat dépend si l'algorithme est utilisé à des fins de classification ou de régression :

en classification k-NN, le résultat est une classe d'appartenance. Un objet d'entrée est classifié selon le résultat majoritaire des statistiques de classes d'appartenance de ses k plus proches voisins, (k est un nombre entier positif généralement petit). Si $k = 1$, alors l'objet est assigné à la classe d'appartenance de son proche voisin. en régression k-NN, le résultat est la valeur pour cet objet. Cette valeur est la moyenne des valeurs des k plus proches voisins. La méthode k-NN est basée sur l'apprentissage préalable, ou l'apprentissage faible, ou la fonction est évaluée localement, le calcul définitif étant effectué à l'issue de la classification. L'algorithme k-NN est parmi les plus simples des algorithmes de machines apprenantes.

Que ce soit pour la classification ou la régression, une technique efficace peut être utilisée pour pondérer l'influence contributive des voisinages, ainsi les plus proches voisins contribuent-ils plus à la moyenne que les voisins plus éloignés. Pour exemple, un schéma courant de pondération consiste à donner à chaque voisin une pondération de $\frac{1}{d}$, ou d est la distance de l'élément, à classer ou à pondérer, de ce voisin.

Les voisins sont pris depuis un ensemble d'objets pour lesquels la classe (en classification k-NN) ou la valeur (pour une régression k-NN) est connue. Ceci peut être considéré comme l'ensemble d'entraînement pour l'algorithme, bien qu'un entraînement explicite ne soit pas particulièrement requis.

Une particularité des algorithmes k-NN est d'être particulièrement sensible à la structure locale des données.

2.2.2 Application

```
library(nnet)
library(class)
library(MASS)
library(mlbench)
```

```
library(tidyr)
library(magrittr)
library(dplyr)

data=read.csv("iris1.csv",sep = ",",dec=",", fileEncoding="CP1252")
data=data[,-1]

data[data== "" ] <- NA
data<-data %>% drop_na()
data<-data%>%
            mutate_if(sapply(data, is.factor), as.numeric)
appindex = sample(1 :nrow(data), round(2*nrow(data)/3), replace = FALSE)
app = data[appindex, ]
val=data[-appindex, ]

pred=knn(app,val,app$Species,k=3,prob=FALSE)
taux2=sum(pred==val$Species)/length(pred)
taux2
```

```
## [1] 0.98
```

2.3 comparaison

Le taux d'erreur de prédiction est plus bon pour le knn (2 %) que pour le reseau de neurones (10 %). Cependant, la diffrence est grand et dépend de l'aléa du découpage de notre jeu de donées et du fait qu'aussila méthode des k plus proches voisins(knn) est un algorithme tres puissant. De plus, le critére de qualié de la prédiction (best performance) est meilleur pour la méthode des k plus proches voisins(knn) que pour le réseau de neurones que. Ainsi, on peut affirmer que le Perceptron Multi-Couches est de qualité similaire à la méthode des k plus proches voisins pour faire de la prévision.

3 Démonstration d'un logiel(ou autre que nnet de R) traitant des des réseaux neurones

3.1 Définition

Neuralnet est un package de R permettant de traiter des reseaux neurones et de resoudre des problemes de régression et de classification. ##Résolution des problèmes de régression à l'aide de neuralnet Nous avons déjà vu comment un réseau neuronal peut être utilisé pour résoudre les problèmes de classification en essayant de regrouper des données en fonction de ses attributs.

Visitons maintenant le jeu de données iris1. csv. Dans cet exemple, nous souhaitons analyser l'impact de la capacité des variables explicatives sur la variables Species.

```
#MAX-MIN NORMALIZATION
normalize <- function(x) { return ((x - min(x)) / (max(x) - min(x)))
}
mydata <- read.csv("iris1.csv")
mydata=mydata[,-1]
mydata[mydata== "" ] <- NA
mydata<-mydata %>% drop_na()
mydata<-mydata%>%
            mutate_if(sapply(mydata, is.factor), as.numeric)
maxmindf <- as.data.frame(lapply(mydata, normalize))
#TRAINING AND TEST DATA
```

```
trainset <- maxmindf[1:32, ]
testset <- maxmindf[33:40, ]
```

3.1.1 Sortie réseau neuronal

Nous avons ensuite exécuter notre réseau neuronal et générer nos paramètres :

```
library(neuralnet)
```

```
##
## Attaching package: 'neuralnet'
```

```
## The following object is masked from 'package:dplyr':
##
##     compute
```

```
library(dplyr)
nn <- neuralnet(Species~ Sepal.Length +Sepal.Width +Petal.Length +Petal.Width,data=trainset, hidden=c(2
nn$result.matrix
```

```
##                                   [,1]
## error                     7.862736e-06
## reached.threshold         7.147060e-03
## steps                     2.400000e+01
## Intercept.to.1layhid1    -9.547669e-01
## Sepal.Length.to.1layhid1 -1.256433e+00
## Sepal.Width.to.1layhid1   1.267609e+00
## Petal.Length.to.1layhid1 -2.442811e-01
## Petal.Width.to.1layhid1   9.216830e-01
## Intercept.to.1layhid2     7.928098e-01
## Sepal.Length.to.1layhid2 -4.201989e-01
## Sepal.Width.to.1layhid2  -6.560989e-02
## Petal.Length.to.1layhid2 -2.205553e+00
## Petal.Width.to.1layhid2  -1.689967e-01
## Intercept.to.2layhid1    -7.499256e-01
## 1layhid1.to.2layhid1     -7.778144e-01
## 1layhid2.to.2layhid1     -9.225415e-01
## Intercept.to.Species     -2.575115e-02
## 2layhid1.to.Species       1.556982e-01
```

```
plot(nn)
```

3.1.2 Validation du modèle

Ensuite, nous validerons (ou testons la précision de notre modèle) en comparant les dépenses estimées d'essence produites du réseau neuronal aux dépenses réelles comme indiqué dans la sortie d'essai :

```
temp_test <- subset(testset, select = c("Species","Sepal.Length", "Sepal.Width", "Petal.Length", "Petal
#head(temp_test)
nn.results <- compute(nn, temp_test)
results <- data.frame(actual = testset$Species, prediction = nn.results$net.result)
results
```

```
##    actual    prediction
## 33      0 -9.963329e-04
## 34      0 -9.248406e-04
## 35      0  2.797785e-04
```

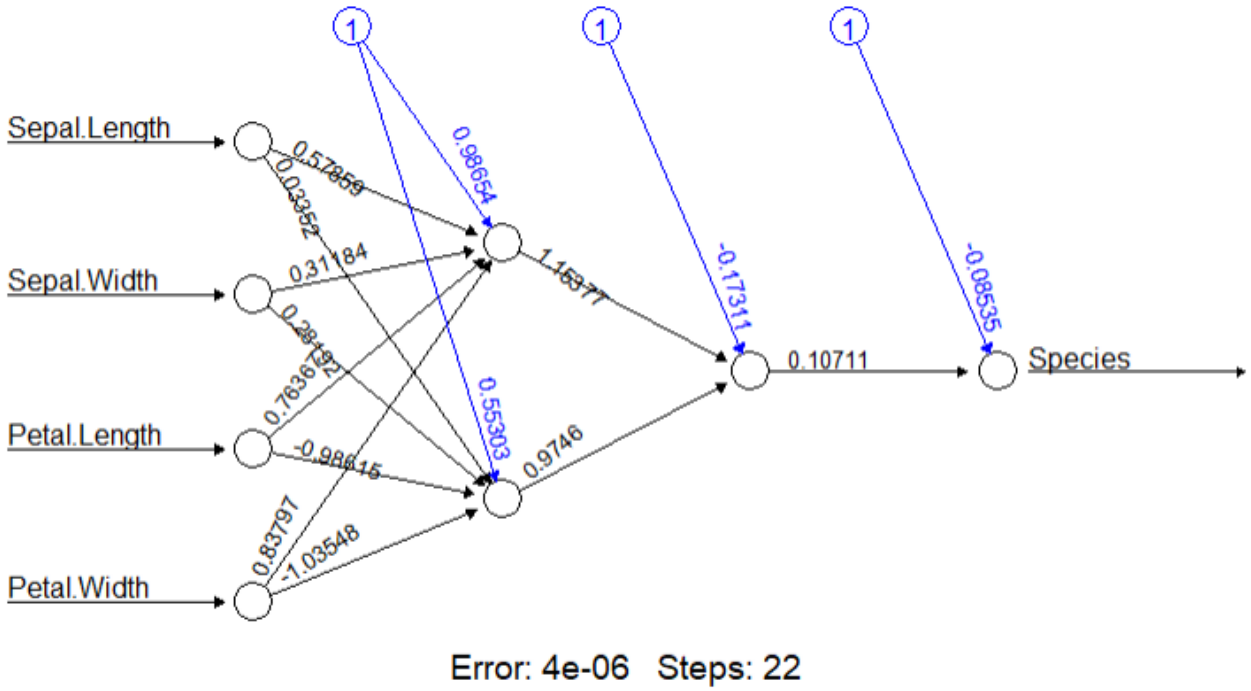


FIGURE 5 –

```
## 36      0 -2.876163e-04
## 37      0  2.693973e-04
## 38      0 -7.673446e-04
## 39      0 -8.424482e-04
## 40      0  7.987721e-05
```

3.1.3 Précision

Dans le code ci-dessous, nous convertissons ensuite les données dans leur format d'origine, et nous avons une précision de 99% sur une base de déviation absolue moyenne . Notez que nous convertissons également nos données en valeurs standard, étant donné qu'elles ont été précédemment mises à l'échelle à l'aide de la technique de normalisation max-min :

```
predicted=results$prediction * abs(diff(range(mydata$Species))) + min(mydata$Species)
actual=results$actual * abs(diff(range(mydata$Species))) + min(mydata$Species)
comparison=data.frame(predicted,actual)
deviation=((actual-predicted)/actual)
comparison=data.frame(predicted,actual,deviation)
accuracy1=1-abs(mean(deviation))
accuracy1
```

```
## [1] 0.9992026
```

Vous pouvez voir que nous obtenons 99.9202618% de précision à l'aide d'une (2, 1) configuration cachée. C'est assez bon, surtout en considérant que notre variable dépendante est en format d'intervalle. Cependant, nous allons voir si nous pouvons l'obtenir plus haut !

Que se passe-t-il si nous utilisons maintenant une configuration cachée (5, 2) dans notre réseau neuronal ? Voici la sortie générée :

```
 nn <- neuralnet(Species~ Sepal.Length +Sepal.Width +Petal.Length +Petal.Width,data=trainset, hidden=c(5
nn$result.matrix
```

```
##                                   [,1]
## error                       0.0036944813
## reached.threshold           0.0088982226
## steps                      25.0000000000
## Intercept.to.1layhid1       0.9979595287
## Sepal.Length.to.1layhid1   -1.3336275049
## Sepal.Width.to.1layhid1    -0.2939779911
## Petal.Length.to.1layhid1    0.5006646416
## Petal.Width.to.1layhid1    -0.4771206315
## Intercept.to.1layhid2      -0.6252898055
## Sepal.Length.to.1layhid2   -0.0321918063
## Sepal.Width.to.1layhid2     0.9421046740
## Petal.Length.to.1layhid2    0.2276798070
## Petal.Width.to.1layhid2     0.8970492462
## Intercept.to.1layhid3      -0.0778208489
## Sepal.Length.to.1layhid3   -0.0182518746
## Sepal.Width.to.1layhid3    -2.0710319850
## Petal.Length.to.1layhid3   -2.1592334576
## Petal.Width.to.1layhid3     0.8359374239
## Intercept.to.1layhid4       0.1963027929
## Sepal.Length.to.1layhid4   -0.0836942984
## Sepal.Width.to.1layhid4    -0.0005285946
## Petal.Length.to.1layhid4    0.6708816890
## Petal.Width.to.1layhid4     1.6399637003
## Intercept.to.1layhid5      -1.2080788479
## Sepal.Length.to.1layhid5   -0.6004141443
## Sepal.Width.to.1layhid5    -0.2476946097
## Petal.Length.to.1layhid5    1.4621823742
## Petal.Width.to.1layhid5     1.2698675848
## Intercept.to.2layhid1      -1.9634195740
## 1layhid1.to.2layhid1        0.9524177271
## 1layhid2.to.2layhid1        1.1316562608
## 1layhid3.to.2layhid1       -0.4927840285
## 1layhid4.to.2layhid1        0.8010143724
## 1layhid5.to.2layhid1       -0.3874784995
## Intercept.to.2layhid2      -1.5296135216
## 1layhid1.to.2layhid2       -1.6826806629
## 1layhid2.to.2layhid2       -0.5033636247
## 1layhid3.to.2layhid2       -1.5590085193
## 1layhid4.to.2layhid2       -1.1536975739
## 1layhid5.to.2layhid2        0.9337212005
## Intercept.to.Species        0.5606597136
## 2layhid1.to.Species        -1.4907286896
## 2layhid2.to.Species         0.0313174829
```

```
plot(nn)
```

```
results <- data.frame(actual = testset$Species, prediction = nn.results$net.result)
results
```

```
##    actual    prediction
## 33      0 -9.963329e-04
```

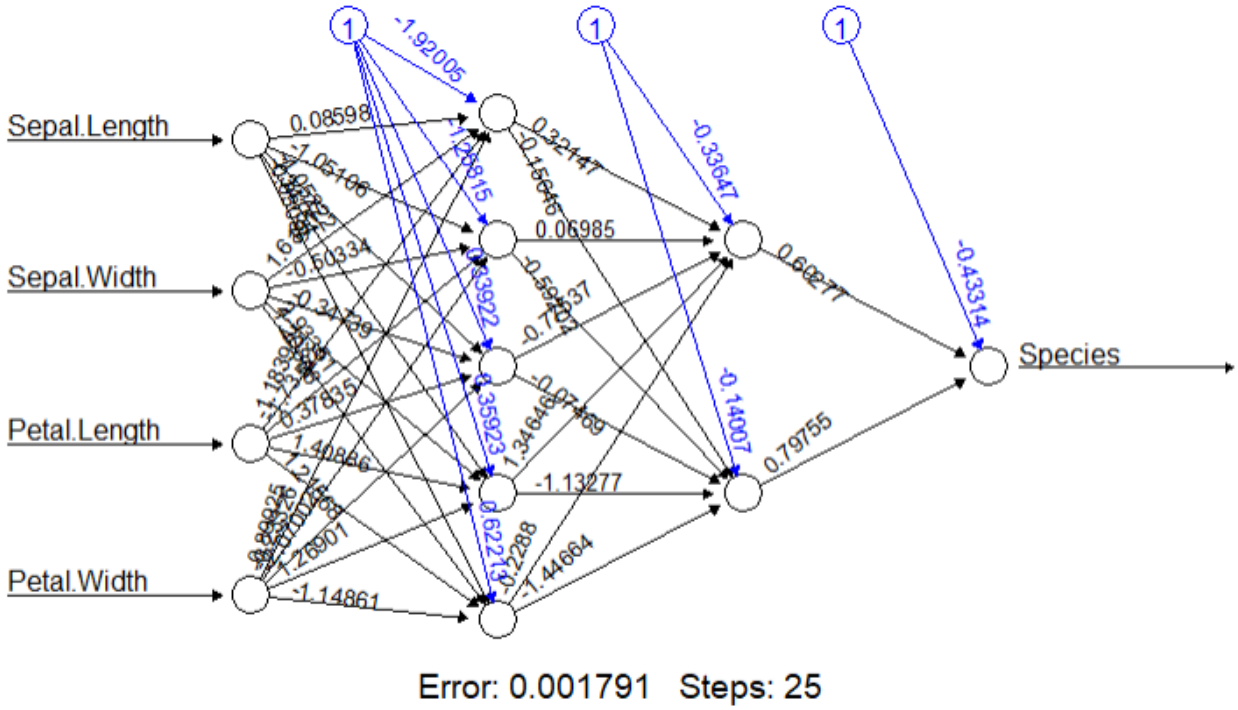


FIGURE 6 –

```
## 34      0 -9.248406e-04
## 35      0  2.797785e-04
## 36      0 -2.876163e-04
## 37      0  2.693973e-04
## 38      0 -7.673446e-04
## 39      0 -8.424482e-04
## 40      0  7.987721e-05
```

```
predicted=results$prediction * abs(diff(range(mydata$Species))) + min(mydata$Species)
actual=results$actual * abs(diff(range(mydata$Species))) + min(mydata$Species)
comparison=data.frame(predicted,actual)
deviation=((actual-predicted)/actual)
comparison=data.frame(predicted,actual,deviation)
accuracy2=1-abs(mean(deviation))
accuracy2
```

```
## [1] 0.9992026
```

Nous voyons que notre taux de précision a maintenant augmenté à près de 99.9202618%, ce qui indique que la modification du nombre de nœuds cachés a amélioré notre modèle !

4 Utilisation d'un logiel pour une démonstrationde carte de kohonen avec discussion

4.1 Définition

Les cartes auto adaptatives, cartes auto-organisatrices ou cartes topologiques forment une classe de réseau de neurones artificiels fondée sur des méthodes d'apprentissage non-supervisées.

Elles sont souvent désignées par le terme anglais self organizing maps (SOM), ou encore cartes de Kohonen du nom du statisticien ayant développé le concept en 1984. La littérature utilise aussi les dénominations : « réseau de Kohonen », « réseau auto-adaptatif » ou « réseau auto-organisé ».

Elles sont utilisées pour cartographier un espace réel, c'est-à-dire pour étudier la répartition de données dans un espace à grande dimension. En pratique, cette cartographie peut servir à réaliser des tâches de discrétisation, quantification vectorielle ou classification.

4.2 Architecture

D'un point de vue architectural, les cartes auto-organisatrices de Kohonen sont constituées d'une grille (le plus souvent uni- ou bidimensionnelle). Dans chaque nœud de la grille se trouve un « neurone ». Chaque neurone est lié à un vecteur référent, responsable d'une zone dans l'espace des données (appelé encore espace d'entrée).

Dans une carte auto-organisatrice, les vecteurs référents fournissent une représentation discrète de l'espace d'entrée. Ils sont positionnés de telle façon qu'ils conservent la forme topologique de l'espace d'entrée. En gardant les relations de voisinage dans la grille, ils permettent une indexation facile (via les coordonnées dans la grille). Ceci s'avère utile dans divers domaines, comme la classification de textures, l'interpolation entre des données, la visualisation des données multidimensionnelles.

Soit A la grille neuronale rectangulaire d'une carte auto-organisatrice. Une carte de neurones affecte à chaque vecteur d'entrée $v \in V$ un neurone $r \in A$ désigné par son vecteur de position $\vec{r}$, tel que le vecteur référent w_r soit le plus proche de v. Mathématiquement, on exprime cette association par une fonction : $\phi_w : V \to A$

$r = \phi_w(v) = \arg\min_{r \in A} \|v - w_r\|$.

Cette fonction permet de définir les applications de la carte.

quantificateur vectoriel : on approxime chaque point dans l'espace d'entrée par le vecteur référent le plus proche par $w_r = \phi_w^{-1}(\phi_w(v))w_r = \phi_w^{-1}(\phi_w(v))$

classificateur en utilisant la fonction $r = \phi_w(v)r = \phi_w(v)$ on affecte à chaque neurone de la grille une étiquette correspondant à une classe ; tous les points de l'espace d'entrée qui se projettent sur un même neurone appartiennent à la même classe. Une même classe peut être associée à plusieurs neurones.

4.3 Algorithme d'apprentissage

```
library(nnet)
library(class)
library(MASS)
library(mlbench)
library(tidyr)
library(magrittr)
library(dplyr)

data=read.csv("features.csv",sep = ",",dec=",", fileEncoding="CP1252")
data=data[,-c(1,2,3,4,5,6,19,42)]
data[data== "" ] <- NA
```

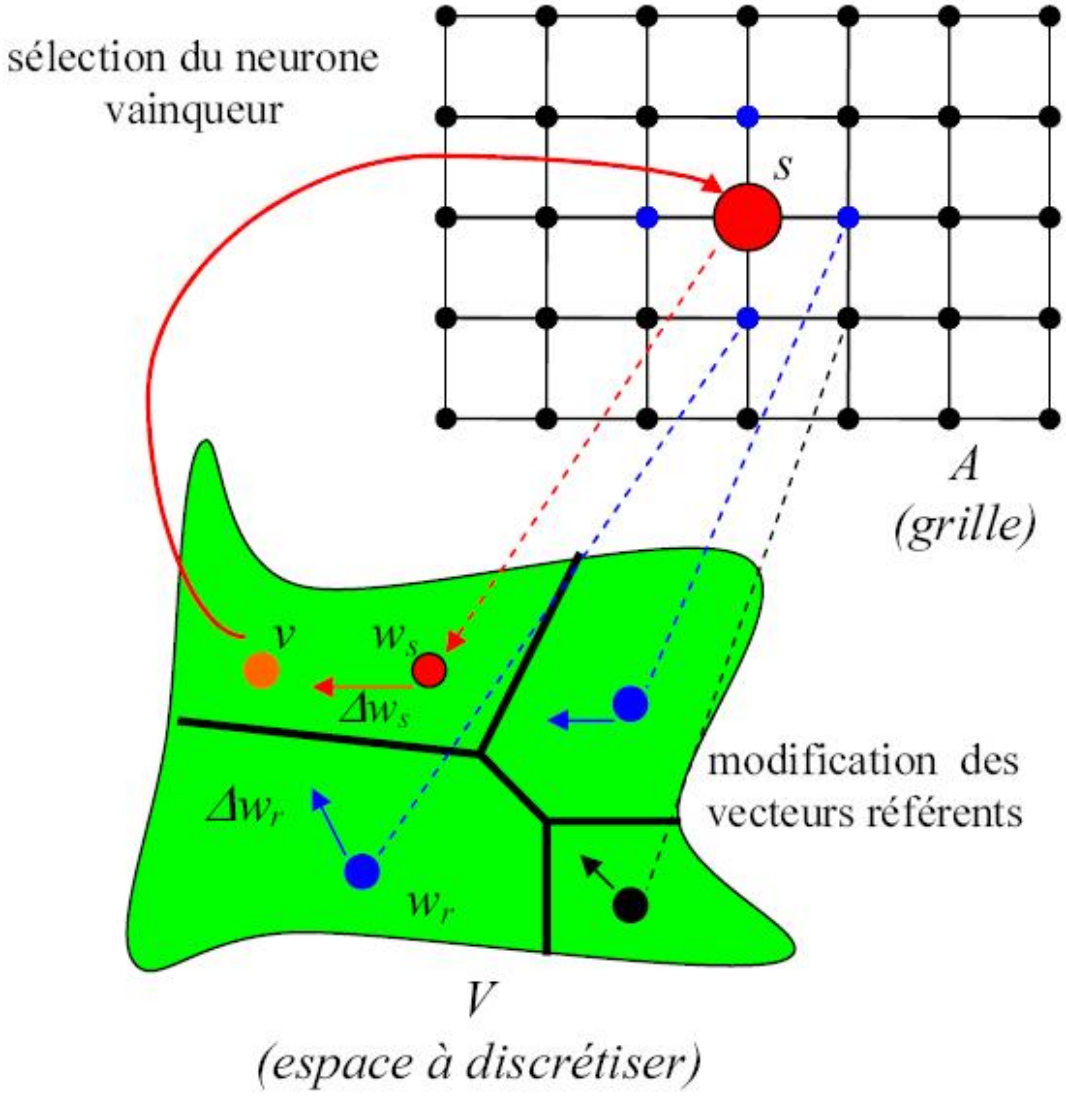

sélection du neurone
vainqueur
s
A
(grille)
v
w_s
Δw_s
Δw_r
w_r
modification des
vecteurs référents
V
(espace à discrétiser)

FIGURE 7 –

```r
data<-data %>% drop_na()
data<-data%>%
            mutate_if(sapply(data, is.factor), as.numeric)

summary(data)
```

```
##        CC               CD               DT               EX
##  Min.   :  0.00   Min.   :  0.00   Min.   : 0.000   Min.   : 0.00
##  1st Qu.:  7.00   1st Qu.: 24.75   1st Qu.: 0.000   1st Qu.: 3.00
##  Median : 14.00   Median : 58.00   Median : 0.000   Median : 8.00
##  Mean   : 20.63   Mean   : 66.45   Mean   : 1.054   Mean   :11.48
##  3rd Qu.: 26.25   3rd Qu.: 93.00   3rd Qu.: 1.000   3rd Qu.:15.00
##  Max.   :225.00   Max.   :471.00   Max.   :12.000   Max.   :96.00
##        FW              INs               JJ              JJR
##  Min.   :  2.00   Min.   :  2.00   Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 31.75   1st Qu.: 19.00   1st Qu.: 0.000   1st Qu.: 0.000
##  Median : 67.00   Median : 40.00   Median : 1.000   Median : 1.000
##  Mean   : 75.15   Mean   : 47.43   Mean   : 2.152   Mean   : 1.901
##  3rd Qu.:103.00   3rd Qu.: 65.00   3rd Qu.: 3.000   3rd Qu.: 3.000
##  Max.   :473.00   Max.   :354.00   Max.   :26.000   Max.   :16.000
##       JJS              LS               MD              NN
##  Min.   :0.000   Min.   : 0.000   Min.   : 17.0   Min.   : 0.000
##  1st Qu.:0.000   1st Qu.: 1.000   1st Qu.: 74.0   1st Qu.: 1.000
##  Median :0.000   Median : 4.000   Median :133.5   Median : 2.000
##  Mean   :0.018   Mean   : 7.139   Mean   :147.1   Mean   : 3.152
##  3rd Qu.:0.000   3rd Qu.:10.000   3rd Qu.:189.2   3rd Qu.: 5.000
##  Max.   :5.000   Max.   :75.000   Max.   :897.0   Max.   :19.000
##       NNPS             NNS              PDT             POS
##  Min.   :  0.00   Min.   :0.000   Min.   : 0.00   Min.   :  0
##  1st Qu.: 14.00   1st Qu.:0.000   1st Qu.: 0.00   1st Qu.:  6
##  Median : 32.50   Median :0.000   Median : 0.00   Median : 17
##  Mean   : 39.11   Mean   :0.387   Mean   : 2.19   Mean   : 25
##  3rd Qu.: 55.00   3rd Qu.:1.000   3rd Qu.: 3.00   3rd Qu.: 39
##  Max.   :305.00   Max.   :7.000   Max.   :58.00   Max.   :175
##       PRP              PRP.             RB              RBR
##  Min.   : 0.00   Min.   :  0.00   Min.   : 0.000   Min.   :0.000
##  1st Qu.: 4.00   1st Qu.:  8.00   1st Qu.: 0.000   1st Qu.:0.000
##  Median : 8.00   Median : 24.00   Median : 0.000   Median :0.000
##  Mean   :10.34   Mean   : 31.32   Mean   : 0.964   Mean   :0.398
##  3rd Qu.:15.00   3rd Qu.: 47.00   3rd Qu.: 2.000   3rd Qu.:1.000
##  Max.   :73.00   Max.   :209.00   Max.   :10.000   Max.   :7.000
##       RBS              RP              SYM             TOs
##  Min.   : 0.000   Min.   : 0.000   Min.   :  0.00   Min.   :0.000
##  1st Qu.: 1.000   1st Qu.: 0.000   1st Qu.:  6.00   1st Qu.:0.000
##  Median : 2.000   Median : 0.000   Median : 14.00   Median :0.000
##  Mean   : 3.351   Mean   : 1.919   Mean   : 16.54   Mean   :0.058
##  3rd Qu.: 5.000   3rd Qu.: 2.000   3rd Qu.: 24.00   3rd Qu.:0.000
##  Max.   :32.000   Max.   :19.000   Max.   :101.00   Max.   :3.000
##        UH              VB              VBD             VBG
##  Min.   : 0.00   Min.   :  0.00   Min.   : 0.00   Min.   : 0.00
##  1st Qu.: 1.00   1st Qu.: 11.00   1st Qu.: 5.00   1st Qu.: 5.00
##  Median : 3.00   Median : 24.00   Median :11.00   Median :10.00
##  Mean   : 4.38   Mean   : 29.95   Mean   :13.58   Mean   :12.79
##  3rd Qu.: 7.00   3rd Qu.: 44.00   3rd Qu.:19.00   3rd Qu.:17.00
```

```
##  Max.   :31.00   Max.   :220.00   Max.   :82.00   Max.   :96.00
##       VBN              VBP              VBZ              WDT
##  Min.   : 0.0   Min.   :  0.00   Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 3.0   1st Qu.:  4.00   1st Qu.: 0.000   1st Qu.: 1.000
##  Median : 8.0   Median : 11.00   Median : 2.000   Median : 2.000
##  Mean   :11.9   Mean   : 17.13   Mean   : 2.547   Mean   : 3.673
##  3rd Qu.:18.0   3rd Qu.: 25.00   3rd Qu.: 4.000   3rd Qu.: 5.000
##  Max.   :97.0   Max.   :142.00   Max.   :27.000   Max.   :26.000
##        WP              WP.             baseform          Quotes
##  Min.   :0.000   Min.   : 0.000   Min.   :  0.00   Min.   : 0.000
##  1st Qu.:0.000   1st Qu.: 1.000   1st Qu.:  6.00   1st Qu.: 0.000
##  Median :0.000   Median : 2.000   Median : 17.00   Median : 1.000
##  Mean   :0.099   Mean   : 3.461   Mean   : 20.59   Mean   : 3.785
##  3rd Qu.:0.000   3rd Qu.: 5.000   3rd Qu.: 30.00   3rd Qu.: 6.000
##  Max.   :3.000   Max.   :37.000   Max.   :175.00   Max.   :28.000
##  questionmarks   exclamationmarks   fullstops          commas
##  Min.   : 0.00   Min.   : 0.00    Min.   :  1.00   Min.   :  0.00
##  1st Qu.: 0.00   1st Qu.: 0.00    1st Qu.: 14.00   1st Qu.:  9.00
##  Median : 0.00   Median : 0.00    Median : 31.00   Median : 23.00
##  Mean   : 1.04   Mean   : 0.21    Mean   : 34.94   Mean   : 28.63
##  3rd Qu.: 1.00   3rd Qu.: 0.00    3rd Qu.: 49.00   3rd Qu.: 40.00
##  Max.   :20.00   Max.   :11.00    Max.   :253.00   Max.   :324.00
##    semicolon          colon           ellipsis      pronouns1st
##  Min.   : 0.000   Min.   : 0.000   Min.   :0.000   Min.   : 0.000
##  1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.:0.000   1st Qu.: 0.000
##  Median : 0.000   Median : 1.000   Median :0.000   Median : 2.000
##  Mean   : 0.302   Mean   : 1.618   Mean   :0.003   Mean   : 4.316
##  3rd Qu.: 0.000   3rd Qu.: 2.000   3rd Qu.:0.000   3rd Qu.: 6.000
##  Max.   :15.000   Max.   :42.000   Max.   :3.000   Max.   :80.000
##   pronouns2nd      pronouns3rd     compsupadjadv         past
##  Min.   : 0.000   Min.   :  0.00   Min.   : 0.000   Min.   :  0.00
##  1st Qu.: 0.000   1st Qu.:  6.00   1st Qu.: 1.000   1st Qu.: 11.00
##  Median : 1.000   Median : 14.00   Median : 4.000   Median : 24.00
##  Mean   : 2.701   Mean   : 19.19   Mean   : 5.419   Mean   : 29.94
##  3rd Qu.: 3.000   3rd Qu.: 29.00   3rd Qu.: 8.000   3rd Qu.: 44.00
##  Max.   :44.000   Max.   :127.00   Max.   :39.000   Max.   :220.00
##    imperative       present3rd     present1st2nd   sentence1st
##  Min.   : 0.000   Min.   :  0.00   Min.   : 0.0   Min.   :0.000
##  1st Qu.: 1.000   1st Qu.:  4.00   1st Qu.: 3.0   1st Qu.:1.000
##  Median : 3.000   Median : 11.00   Median : 8.0   Median :1.000
##  Mean   : 4.389   Mean   : 17.12   Mean   :11.9   Mean   :0.927
##  3rd Qu.: 7.000   3rd Qu.: 25.00   3rd Qu.:18.0   3rd Qu.:1.000
##  Max.   :31.000   Max.   :142.00   Max.   :97.0   Max.   :1.000
##   sentencelast   txtcomplexity
##  Min.   :0.000   Min.   : 5.00
##  1st Qu.:1.000   1st Qu.:15.00
##  Median :1.000   Median :18.00
##  Mean   :0.995   Mean   :19.14
##  3rd Qu.:1.000   3rd Qu.:22.00
##  Max.   :1.000   Max.   :73.00
```

```
d=scale(data,center=T,scale=T)
```

```
set.seed(100)
library(kohonen)
```

```
##
## Attaching package: 'kohonen'
## The following object is masked from 'package:class':
##
##     somgrid
```

```
library(class)
carte=som(d,grid=somgrid(6,7,"hexagonal"))
```

Nous avons demandé une grille hexagonale de taille 6x7.

```
print(summary(carte))
```

```
## SOM of size 6x7 with a hexagonal topology and a bubble neighbourhood function.
## The number of data layers is 1.
## Distance measure(s) used: sumofsquares.
## Training data included: 1000 objects.
## Mean distance to the closest unit in the map: 19.483.
## NULL
```

on peut visualiser l'architecture de la grille

```
#architecture of the grid
print(carte$grid)
```

```
## $pts
##         x         y
##  [1,] 1.5 0.8660254
##  [2,] 2.5 0.8660254
##  [3,] 3.5 0.8660254
##  [4,] 4.5 0.8660254
##  [5,] 5.5 0.8660254
##  [6,] 6.5 0.8660254
##  [7,] 1.0 1.7320508
##  [8,] 2.0 1.7320508
##  [9,] 3.0 1.7320508
## [10,] 4.0 1.7320508
## [11,] 5.0 1.7320508
## [12,] 6.0 1.7320508
## [13,] 1.5 2.5980762
## [14,] 2.5 2.5980762
## [15,] 3.5 2.5980762
## [16,] 4.5 2.5980762
## [17,] 5.5 2.5980762
## [18,] 6.5 2.5980762
## [19,] 1.0 3.4641016
## [20,] 2.0 3.4641016
## [21,] 3.0 3.4641016
## [22,] 4.0 3.4641016
## [23,] 5.0 3.4641016
## [24,] 6.0 3.4641016
## [25,] 1.5 4.3301270
## [26,] 2.5 4.3301270
## [27,] 3.5 4.3301270
## [28,] 4.5 4.3301270
```

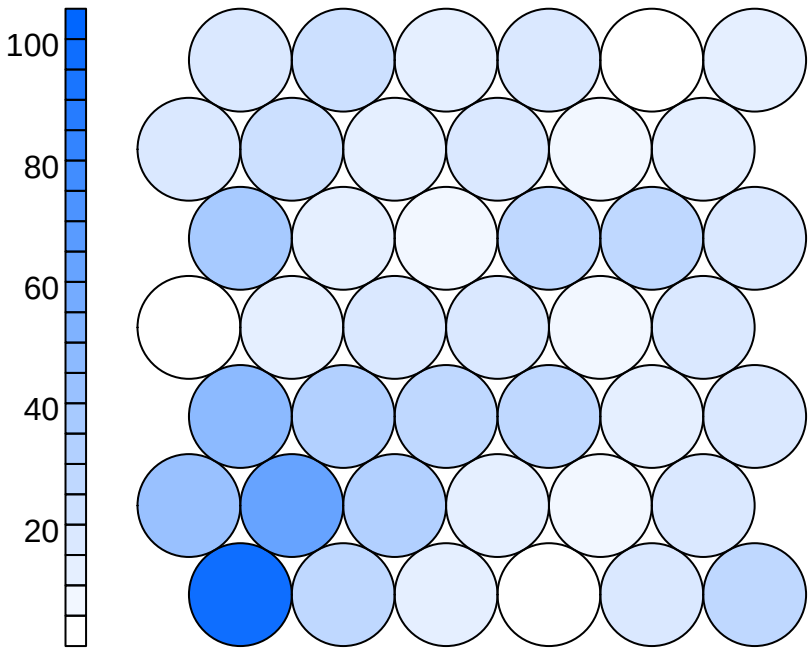
```
## [29,] 5.5 4.3301270
## [30,] 6.5 4.3301270
## [31,] 1.0 5.1961524
## [32,] 2.0 5.1961524
## [33,] 3.0 5.1961524
## [34,] 4.0 5.1961524
## [35,] 5.0 5.1961524
## [36,] 6.0 5.1961524
## [37,] 1.5 6.0621778
## [38,] 2.5 6.0621778
## [39,] 3.5 6.0621778
## [40,] 4.5 6.0621778
## [41,] 5.5 6.0621778
## [42,] 6.5 6.0621778
##
## $xdim
## [1] 6
##
## $ydim
## [1] 7
##
## $topo
## [1] "hexagonal"
##
## $neighbourhood.fct
## [1] bubble
## Levels: bubble gaussian
##
## $toroidal
## [1] FALSE
##
## attr(,"class")
## [1] "somgrid"
```

Les effectifs dans les nœuds permettent d'identifier les zones à forte densité. Le package « kohonen » permet de spécifier le jeu de couleurs à utiliser. Nous définissons la fonction *degrade.bleu ()* [dégradé de bleu]

```
#jeu de couleurs pour les nœuds de la carte
degrade.bleu <- function(n){
 return(rgb(0,0.4,1,alpha=seq(0,1,1/n)))
}
```

La fonction prend en entrée « n » qui correspond au nombre de couleurs à produire. Nous générons un ensemble de couleurs bleues plus ou moins opaques à l'aide de rgb(). Plus la couleur est foncée, plus les effectifs sont élevés. Il reste à afficher la carte avec plot() en spécifiant les options adéquates.

```
#count plot
plot(carte,type="count",palette.name=degrade.bleu)
```



Dans l'idéal, la répartition devrait être assez homogène. La taille de la carte doit être réduite s'il y a de nombreuses cellules vides. A contrario, elle doit être augmentée si des zones de très forte densité apparaissent

Le champ « $unit_classif » indique les numéros des nœuds auxquels sont affectés les individus. Le premier individu est affecté au nœud n° 2, le second au n° 7, . . . , le dernier au n°7.

```
#noeud d'appartenance des observations
print(carte$unit.classif[1:34])
```

```
##  [1]  2  7  1  8 21 13  3 13 32 33 26  1 21  7 21  1  9  9 29 33  1  8  8
## [24] 14 12 25 32  2 22 21 13  1 13  7
```

Nous pouvons ainsi comptabiliser les individus dans chaque nœud avec table(). Il y a 106 individus dans le nœud n°1, 28 dans le n°2, . . . , 13 dans le n°35.

```
#nombre d'observations affectés à chaque noeud
nb <- table(carte$unit.classif)
print(nb)
```

```
##
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
## 106  28  13   6  19  28  48  67  35  13   9  20  51  35  31  32  14  19
##  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##   1  16  19  17  11  22  38  14  11  30  28  20  19  25  16  22   9  13
##  37  38  39  40  41  42
##  19  27  13  19   5  12
```

Nous pouvons aussi vérifier s'il y a des nœuds vides en comptant le nombre de valeurs dans le vecteur engendré par table().

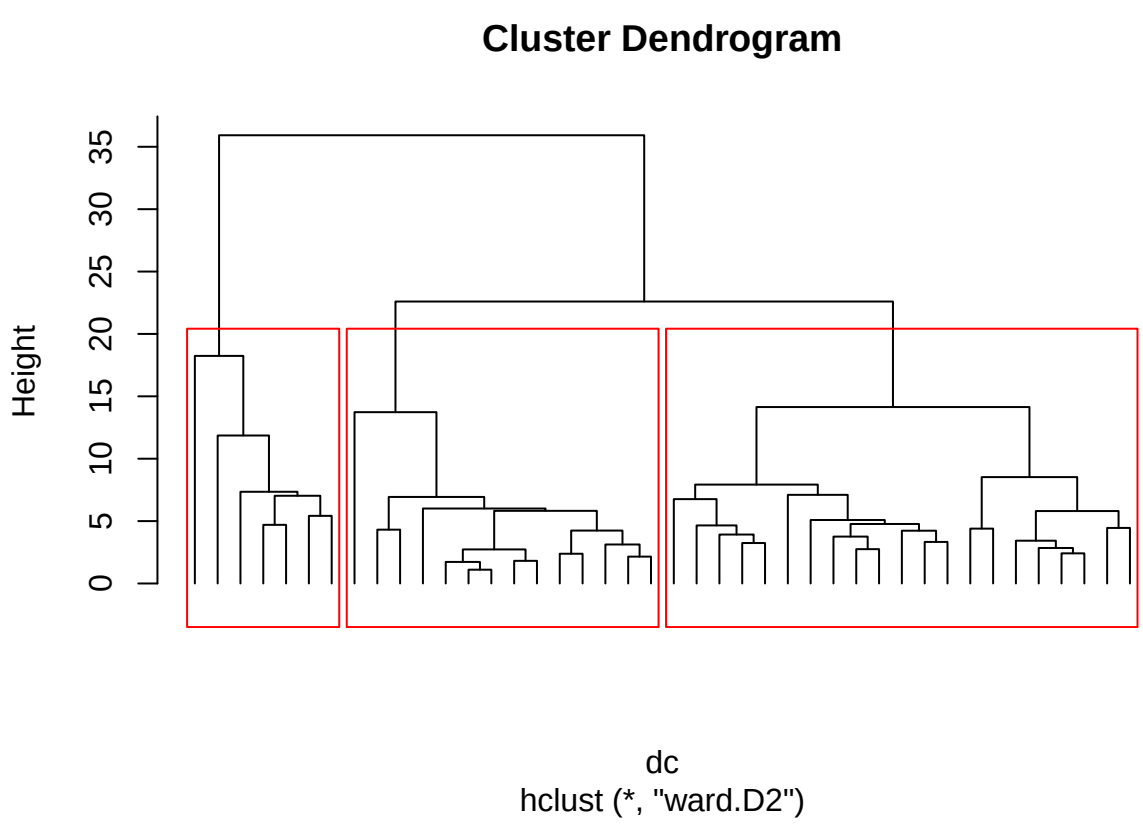
```r
#check if there are empty nodes
print(length(nb))
```

```
## [1] 42
```

4.3.1 Classification automatique à partir de la carte

Pouvoir enchaîner avec une classification automatique est un des intérêts des cartes topologiques de Kohonen. En effet, nous disposons d'une représentation 2D des observations avec des zones et des proximités que l'on sait caractériser avec les variables. Les noeuds constituent un excellent point de départ pour un regroupement itératif en classes. La classification ascendante hiérarchique (CAH) est souvent mise à contribution dans ce contexte.

```r
#matrice de distance entre les noeuds
dc <- dist(carte$codes[[1]])
#cah - attention à l'option members
cah <- hclust(dc,method="ward.D2",members=nb)
plot(cah,hang=-1,labels=F)
#matérialiser les 3 classes dans le dendrogramme
rect.hclust(cah,k=3)
```



Un regroupement en 3 classes paraît légitime à la vue du dendrogramme . Nous créons les groupes sous R (cutree) :

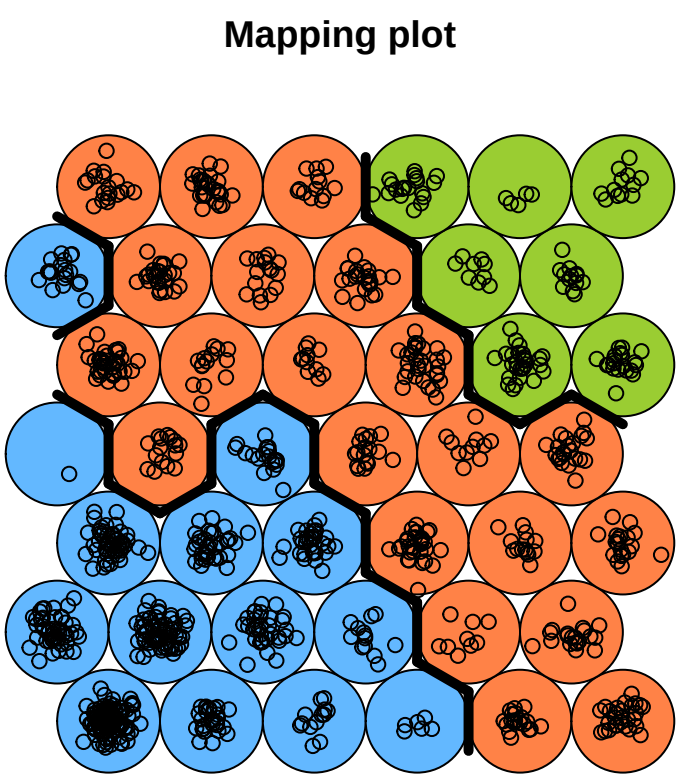
```r
#découpage en 3 classes
groupes <- cutree(cah,k=3)
print(groupes)
```

```
##  V1  V2  V3  V4  V5  V6  V7  V8  V9 V10 V11 V12 V13 V14 V15 V16 V17 V18
##   1   1   1   1   2   2   1   1   1   1   2   2   1   1   1   2   2   2
## V19 V20 V21 V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32 V33 V34 V35 V36
##   1   2   1   2   2   2   2   2   2   2   3   3   1   2   2   2   3   3
## V37 V38 V39 V40 V41 V42
##   2   2   2   3   3   3
```

La variable 'groupes' permet d'associer les nœuds de la carte à leur groupe d'appartenance. Le premier nœud n°1 appartient au groupe 1, le second aussi, ..., le dernier nœud de la carte (n°30) appartient au groupe 3.

Il est possible de visualiser les regroupements dans la carte.

```
#visualisation des classes dans la carte topologique
plot(carte,type="mapping",bgcol=c("steelblue1","sienna1","yellowgreen")[groupes])
add.cluster.boundaries(carte,clustering=groupes)
```



Les 3 zones identifiées dans les différents graphiques précédents sont clairement mises en évidence. Et nous savons les interpréter directement maintenant.

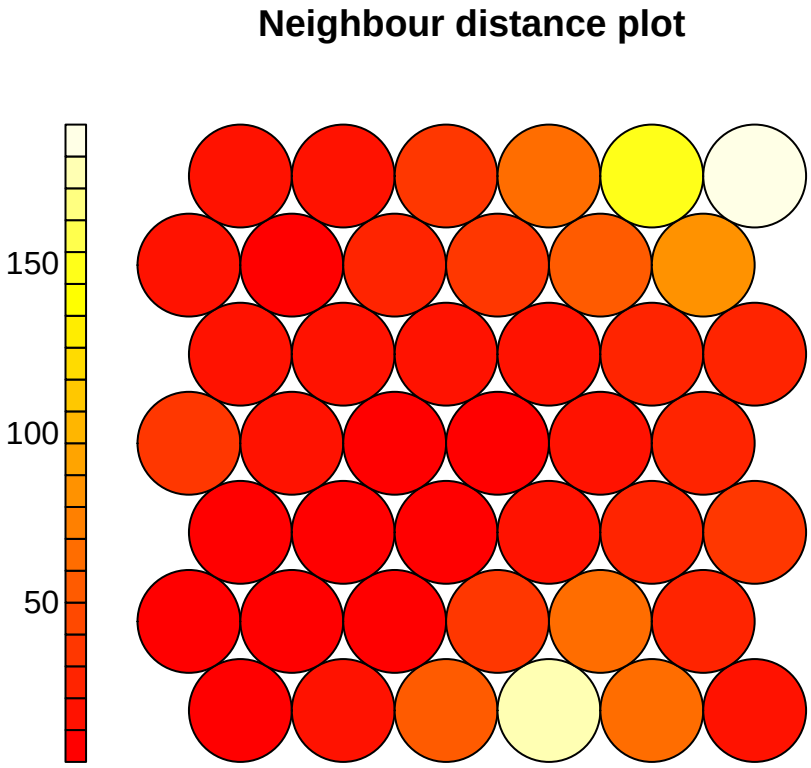
```
#affecter chaque individu à sa classe
ind.groupe <- groupes[carte$unit.classif]
print(ind.groupe)
```

```
##  V2  V7  V1  V8 V21 V13  V3 V13 V32 V33 V26  V1 V21  V7 V21  V1  V9  V9
##   1   1   1   1   1   1   1   1   2   2   2   1   1   1   1   1   1   1
## V29 V33  V1  V8  V8 V14 V12 V25 V32  V2 V22 V21 V13  V1 V13  V7 V31  V7
##   3   2   1   1   1   1   2   2   2   1   2   1   1   1   1   1   1   1
## V13 V15 V13 V13 V27  V7  V8  V1 V13  V1 V26  V7  V7 V21  V1 V26  V7  V7
##   1   1   1   1   2   1   1   1   1   1   2   1   1   1   1   2   1   1
```

```
## V13 V37  V8 V32 V39 V22  V9 V25 V36  V9  V8  V1 V13  V2  V1  V1  V3 V21
##   1   2   1   2   2   2   1   2   3   1   1   1   1   1   1   1   1   1
##  V8  V7 V38  V1  V1  V1  V4  V1  V3 V20 V31  V7 V14 V13  V1 V14  V1 V14
##   1   1   2   1   1   1   1   1   1   2   1   1   1   1   1   1   1   1
##  V8 V23  V1  V8 V38 V22 V10  V8  V8  V9 V32  V1 V25  V1  V8 V25 V37  V2
##   1   2   1   1   2   2   1   1   1   1   2   1   2   1   1   2   2   1
## V32  V9 V15 V14  V7  V1  V1 V16 V25  V7  V1  V1 V21  V9 V13  V1 V32  V1
##   2   1   1   1   1   1   1   2   2   1   1   1   1   1   1   1   2   1
## V32 V20 V39 V14 V20 V32  V5 V25 V25  V8 V16 V35  V1  V7  V7  V1 V37 V33
##   2   2   2   1   2   2   2   2   2   1   2   3   1   1   1   1   2   2
## V13 V14  V1  V1 V14 V14  V8  V1 V13  V1 V29 V16 V16  V1  V1  V1  V1  V2
##   1   1   1   1   1   1   1   1   1   1   3   2   2   1   1   1   1   1
##  V7 V32 V32  V8  V1 V13  V7  V1  V8 V25 V20  V8 V22 V20  V8  V9 V26 V27
##   1   2   2   1   1   1   1   1   1   2   2   1   2   2   1   1   2   2
## V26 V26  V1  V1 V22 V28  V8  V1  V2  V1 V38  V7  V7  V1 V25  V1 V16 V38
##   2   2   1   1   2   2   1   1   1   1   2   1   1   1   2   1   2   2
##  V2 V33 V25  V1 V27 V13 V21 V32  V8  V2 V38  V3  V2  V8  V1  V4 V25  V2
##   1   2   2   1   2   1   1   2   1   1   2   1   1   1   1   1   2   1
## V13  V1 V38 V33  V1 V25 V25  V2  V8  V1  V8 V13  V7 V37 V31  V1 V22  V8
##   1   1   2   2   1   2   2   1   1   1   1   1   1   2   1   1   2   1
##  V1 V13 V29  V3 V29 V26 V13 V16 V21  V8  V7 V28 V29  V1 V30  V9  V8  V7
##   1   1   3   1   3   2   1   2   1   1   1   2   3   1   3   1   1   1
##  V8 V32 V32  V7  V7  V8 V20  V1  V2 V37 V14  V6 V30 V32 V25  V1 V31 V15
##   1   2   2   1   1   1   2   1   1   2   1   2   3   2   2   1   1   1
## V14  V7 V14 V30 V25  V8  V1  V1 V16  V1 V38 V29  V4 V31  V3  V1 V25 V14
##   1   1   1   3   2   1   1   1   2   1   2   3   1   1   1   1   2   1
##  V1  V2 V33  V1 V16 V31 V15  V1  V1 V17  V8  V8 V22  V1 V37 V14 V28 V40
##   1   1   2   1   2   1   1   1   1   2   1   1   2   1   2   1   2   3
## V37 V38 V10  V1 V14 V15 V26 V13 V13  V7 V14 V32 V29 V28 V37  V2 V25  V1
##   2   2   1   1   1   1   2   1   1   1   1   2   3   2   2   1   2   1
## V15 V33 V25 V13  V1 V25 V22 V37  V2 V13  V7  V5 V29 V15 V21  V2  V8  V1
##   1   2   2   1   1   2   2   2   1   1   1   2   3   1   1   1   1   1
## V10  V3  V8  V9 V14 V37 V28  V1  V8  V7 V13 V25  V2  V8 V39 V13 V14 V13
##   1   1   1   1   1   2   2   1   1   1   1   2   1   1   2   1   1   1
## V14  V1 V25 V15 V37 V33 V26 V26  V3  V9 V40 V22  V2 V13  V1 V13  V1  V4
##   1   1   2   1   2   2   2   2   1   1   3   2   1   1   1   1   1   1
##  V9 V32 V15  V7 V31 V42  V1 V14  V9  V9  V1 V29  V8 V10 V13 V38  V3  V1
##   1   2   1   1   1   3   1   1   1   1   1   3   1   1   1   2   1   1
## V16 V12  V1  V8 V15  V1  V2  V7 V37  V9 V26  V9  V9  V1 V13 V15 V32  V8
##   2   2   1   1   1   1   1   1   2   1   2   1   1   1   1   1   2   1
## V25 V38 V30 V14 V14 V34  V1 V23 V34 V20  V7 V27  V9  V3 V31 V38 V15  V7
##   2   2   3   1   1   2   1   2   2   2   1   2   1   1   1   2   1   1
## V38  V1  V8  V8 V13  V1 V22 V15 V21 V27  V9 V13 V33 V30 V17 V26  V8 V38
##   2   1   1   1   1   1   2   1   1   2   1   1   2   3   2   2   1   2
## V21 V13 V33 V38 V15 V14 V13 V14  V8  V9  V1 V25 V36 V20 V14  V1  V1 V25
##   1   1   2   2   1   1   1   1   1   1   1   2   3   2   1   1   1   2
## V31  V1 V21 V37  V9 V13 V34  V8 V23 V13 V39  V1 V21 V23  V1 V25 V25  V9
##   1   1   1   2   1   1   2   1   2   1   2   1   1   2   1   2   2   1
## V39 V38 V13 V12 V34 V13 V31 V27 V39  V1  V8  V8 V36 V15 V25  V8 V32 V11
##   2   2   1   2   2   1   1   2   2   1   1   1   3   1   2   1   2   2
##  V1  V8 V14  V1 V29 V10 V25 V28 V32 V13  V1  V8  V9  V7  V9 V37 V36  V8
##   1   1   1   1   3   1   2   2   2   1   1   1   1   1   1   2   3   1
## V15  V9 V10 V13 V38 V13 V40 V13  V1 V29 V41  V8  V7 V29 V21  V9 V21 V32
##   1   1   1   1   2   1   3   1   1   3   3   1   1   3   1   1   1   2
```

```
## V23  V1  V8 V39 V12  V7  V3  V8 V14  V7 V25  V1  V8  V7 V14  V7  V8  V2
##   2   1   1   2   2   1   1   1   1   1   2   1   1   1   1   1   1   1
## V10  V8  V1 V13  V8 V26 V14 V10 V34 V25 V15  V9  V2  V9 V18 V15 V32  V7
##   1   1   1   1   1   2   1   1   2   2   1   1   1   1   2   1   2   1
## V38 V37 V38 V29  V1 V18 V30 V38 V28  V8 V14 V14 V13  V7 V38 V13 V29 V36
##   2   2   2   3   1   2   3   2   2   1   1   1   1   1   2   1   3   3
##  V3 V13  V1 V25  V1 V14 V40 V38 V41 V20  V1 V37 V14 V32  V9 V27 V25  V8
##   1   1   1   2   1   1   3   2   3   2   1   2   1   2   1   2   2   1
## V31  V7 V21 V25  V2 V25  V7 V16  V9 V31  V2  V1  V1 V38 V37  V1  V2 V29
##   1   1   1   2   1   2   1   2   1   1   1   1   1   2   2   1   1   3
## V20 V29 V33 V13 V39 V40 V27  V6 V15  V5 V12 V16 V10 V11  V6 V35 V42 V28
##   2   3   2   1   2   3   2   2   1   2   2   2   1   2   2   3   3   2
##  V5 V12 V40 V29 V29  V6 V12 V36 V12  V8  V6 V35 V10 V23 V24 V16 V37 V40
##   2   2   3   3   3   2   2   3   2   1   2   3   1   2   2   2   2   3
##  V4 V40 V16 V10 V36 V18 V40 V30 V28 V40 V16 V30 V35 V28 V24 V16  V8 V34
##   1   3   2   1   3   2   3   3   2   3   2   3   3   2   2   2   1   2
## V18 V34  V6  V5  V6 V34 V29 V16 V35 V17  V6 V40 V31 V28 V22  V6 V28 V40
##   2   2   2   2   2   2   3   2   3   2   2   3   1   2   2   2   2   3
## V15 V18 V34 V16 V42  V5 V42 V40 V15 V42 V30 V34 V12 V12  V7 V28 V24 V28
##   1   2   2   2   3   2   3   3   1   3   3   2   2   2   1   2   2   2
## V17 V28  V1 V18  V6  V6  V8 V30 V20 V24 V29 V25 V18 V24  V8 V24 V29 V16
##   2   2   1   2   2   2   1   3   2   2   3   2   2   2   1   2   3   2
## V29 V18 V15 V34 V12 V12  V7  V6 V28 V39 V24 V15  V9  V5  V7 V22 V30 V11
##   3   2   1   2   2   2   1   2   2   2   2   1   1   2   1   2   3   2
## V22 V17 V16  V2  V6  V3 V18 V28 V36  V8 V22 V34  V8 V36 V17 V29 V16 V40
##   2   2   2   1   2   1   2   2   3   1   2   2   1   3   2   3   2   3
## V22 V15 V24 V37 V39  V6  V6 V31 V20 V29 V18 V28 V13  V4 V41 V23 V30 V28
##   2   1   2   2   2   2   2   1   2   3   2   2   1   1   3   2   3   2
## V39  V6 V35 V24 V18 V27  V6 V11 V17 V28 V24 V24  V8  V9 V16 V33 V18 V20
##   2   2   3   2   2   2   2   2   2   2   2   2   1   1   2   2   2   2
## V16 V23 V34 V24 V28 V22 V20 V31 V41  V9 V30 V38 V10  V6 V29  V6 V42 V25
##   2   2   2   2   2   2   2   1   3   1   3   2   1   2   3   2   3   2
## V18 V29  V2 V24 V23 V28 V23  V6 V33 V36 V16 V24 V21 V29  V1 V28 V15  V5
##   2   3   1   2   2   2   2   2   2   3   2   2   1   3   1   2   1   2
## V18 V15 V30 V36 V29 V34 V24 V34  V6 V30 V35 V40  V6  V5 V18  V1 V11 V11
##   2   1   3   3   3   2   2   2   2   3   3   3   2   2   2   1   2   2
## V18 V34 V32 V41 V18  V5  V5 V28 V17 V31 V40 V42  V6 V42 V13 V15  V9 V40
##   2   2   2   3   2   2   2   2   2   1   3   3   2   3   1   1   1   3
## V16 V16 V26 V24 V34 V11 V12 V24  V5 V39 V12 V25 V36  V5 V24 V31 V25 V38
##   2   2   2   2   2   2   2   2   2   2   2   2   3   2   2   1   2   2
## V18 V31 V28 V23 V14 V30 V27 V16  V5 V17  V9 V34 V15 V30 V35 V12 V22 V42
##   2   1   2   2   1   3   2   2   2   2   1   2   1   3   3   2   2   3
## V31 V27 V38 V32 V40 V28 V17 V24 V20 V21 V40 V28 V34 V12 V34 V33 V15  V7
##   1   2   2   2   3   2   2   2   2   1   3   2   2   2   2   2   1   1
## V28  V5  V1 V16  V6 V17 V11 V16 V12 V38 V32 V20  V7  V2 V12 V16  V2 V17
##   2   2   1   2   2   2   2   2   2   2   2   2   1   1   2   2   1   2
## V12 V39 V42 V16  V5 V15  V8 V34  V1  V6 V28 V38 V30 V36 V24 V33  V8 V24
##   2   2   3   2   2   1   1   2   1   2   2   2   3   3   2   2   1   2
## V30 V24 V19 V18  V7 V35 V33  V5 V11 V16 V12  V5 V42 V34 V42  V6 V10 V16
##   3   2   1   2   1   3   2   2   2   2   2   2   3   2   3   2   1   2
## V17 V17 V15 V14 V28  V6  V6 V30 V13  V5
##   2   2   1   1   2   2   2   3   1   2
```

```r
#plot distance to neighbours
plot(carte,type="dist.neighbours")
```



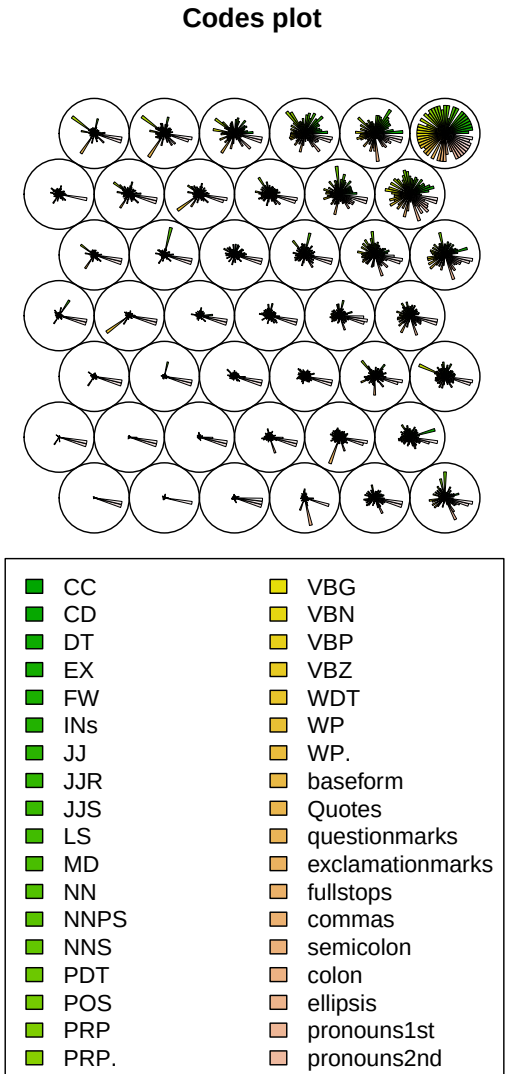


FIGURE 8 –

5 Explication de la machine de Botzmann au moins restreinte ou de l'apprentissage par renforcement

5.1 Définition

En apprentissage automatique, la machine de Boltzmann restreinte ou RBM ("Restricted Boltzmann Machine") est un type de réseau de neurones artificiels pour l'apprentissage non supervisé.Une RBM ("Restricted Boltzmann Machine") est un réseau neuronal stochastique génératif destiné à apprendre une distribution de probabilité sur ses entrées.Elle a initialement été inventée sous le nom de Harmonium en 1986 par Paul Smolenski.

5.2 Architecture

Dans sa forme la plus simple, une machine de Boltzmann est composée d'une couche de neurones qui reçoit l'entrée, ainsi que d'une couche de neurones cachée. Si on suppose que les neurones d'une même couche sont

indépendants entre eux, on appelle cette configuration une machine de Boltzmann restreinte (RBM).

C'est un réseau composé de couches "visibles‘" et "cachées". Il est symétrique (la propagation est dirigée dans un sens ou dans l'autre avec le même poids), entièrement connecté sous la restriction suivante : nous ne permettons pas à toutes les couches d'être ainsi inter-connectées (sinon ce serait une Machine de Bolzmann non restreinte).

5.3 Description

Fonctions d'entrée h et d'activation f : Nous considérons le RBM à unités binaires sous une distribution de Bernouilli : $f(x) := Prob(X = x) = p^x(1-p)^(1-x)$, pour x $\in 0, 1$. La fonction d'entrée possédant un seuil : au-dessus du seuil $h(i) = 1$, en dessous, $h(i) = 0$.

On définit une énergie d'activation pour une Machine de Boltzmann Restreinte de la manière suivante :

$$E = -\left(\sum_{i,j} w_{ij}\, x_i\, h_j + \sum_i b_i\, x_i + \sum_j c_j h_j\right)$$

Avec :

w_{ij} la matrice de poids entre le neurone j et le neurone i ; x_i est l'état, $x_i \in \{0, 1\}$, du neurone visible i ; h_j est l'état du neurone caché j ; b_i et c_j sont respectivement les biais des neurones x_i et h_j.

La probabilité conjointe d'avoir une configuration (x_i, h_j) est alors donnée par

$P(x_i, h_j) = \exp(-E(x_i, h_j))/Z$

Avec :

E E la fonction d'énergie définie ci-dessus ; Z une fonction de normalisation, qui fait en sorte que la somme de toutes les probabilités fasse 1.

5.4 Apprentissage

La machine de Boltzmann s'entraîne à l'aide d'un apprentissage non supervisé. On cherche à minimiser la log-vraisemblance.La dérivée de la log-vraisemblance donne l'expression suivante : $\frac{\partial\left[-\log(p(x^{(t)})\right]}{\partial\theta} = \mathbb{E}_h\left[\frac{\partial E(x^{(t)}, h)}{\partial\theta}|x^{(t)}\right] - \mathbb{E}_{x,y}\left[\frac{\partial E(x,h)}{\partial\theta}\right]$

Avec :

θ les variables du systèmes (les poids ou le biais) ; $\mathbb{E}_{x,y}$ l'espérance mathématiques sur les variables aléatoires x et y ;

$x^{(t)}$ une valeur du jeu de données ; $E(x, h)$ l'énergie définie ci-dessus. On remarque la présence de deux termes dans cette expression, appelés phase positive et phase négative. La phase positive se calcule aisément pour le biais et pour la matrice des poids.

On obtient alors3 :

$\mathbb{E}_h\left[\frac{\partial E(x^{(t)}, h)}{\partial W_{ij}}|x^{(t)}\right] = -h(x^{(t)}) * {x^{(t)}}^\intercal$

Avec $h(x)$ l'état de la couche cachée sachant x donnée par la formule

$h(x) = sigm(W * x + b)h(x) = sigm(W * x + b)$

La partie la plus compliquée est de calculer ce qu'on appelle la phase négative. On ne peut pas la calculer directement car on ne connaît pas la fonction de normalisation du système. Pour pouvoir effectuer une descente de gradient, on calcule ce que l'on appelle la reconstruction de l'entrée $x^{(t)}$. En effet, les propriétés de symétrie du système permettent de calculer l'entrée estimée par le modèle, il suffit d'appliquer la formule :

$x_{rec} = W^{\top} * h(x) + c$

avec c le biais de la couche cachée de neurones H .

De la même manière, on peut recalculer l'état de la couche cachée en réitérant le procédé. Au final, on peut résumer l'algorithme de descente du gradient ainsi4 (on parle de l'algorithme CD-k)

5.5 Application

```
# x <= x(t)
# h <= W*x + b
# phasePositive <= -h*Transpose(x)
# Pour i allant de 1 à k:
#     x = Transpose(W) * h(x) + c
#     h = W*x + b
# phaseNegative <= -h*transpose(x)
# gradient <= phasePositive-phaseNegative
# W <= W + alpha*gradient
# c <= c + alpha*(x(t)-x)
# b <= b + alpha*(h(x(t)) - h)
```

5.5.1 Commencer avec RBM

```
library(deepnet)
a<-matrix(c(1,0,0,0,1,0,0,0,1,1,1,1),nrow=4,ncol=3,byrow=T)
RBM_trn<-rbm.train(a, 2, numepochs = 30, batchsize = 100, learningrate=0.8,
momentum =0.5 ,visible_type = "bin",hidden_type = "bin" , cd = 1)
#RBM_trn
a
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
## [4,]    1    1    1
```

Commençons par installer mon paquet de GitHub, ceci peut être fait avec l'extrait de code suivant :

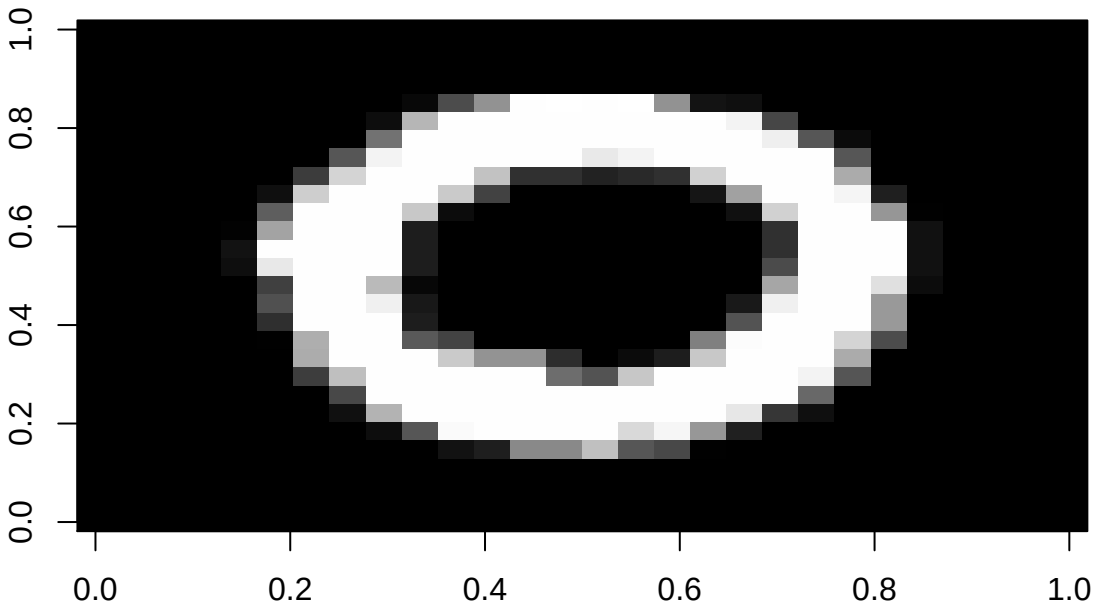
```
# First install devtools
#install.packages("devtools")
# Load devtools
library(devtools)
# Install RBM
#install_github("TimoMatzen/RBM")
# load RBM
library(RBM)
```

Le jeu de données MNIST sera utilisé dans ce document.Les données sont dans le format d'une liste avec un jeu de train, les étiquettes de train, le jeu de test et les étiquettes de test. Permet de charger les données :

```
# Load the MNIST data
data(MNIST)
```

Il Permet également de vérifier un moment si les données semblent comme il est censé regarder :

```
# Lets plot a data example of the train set
image(matrix(MNIST$trainX[2, ], nrow = 28), col = grey(seq(0, 1, length = 256)))
```



5.5.2 Utilisation de RBM ()

Si les données sont chargées correctement, il devrait ressembler à ceci :

```
# First get the train data from MNIST
train <- MNIST$trainX
# Then fit the model
modelRBM <- RBM(x = train, n.iter = 1000, n.hidden = 100, size.minibatch = 10)
```

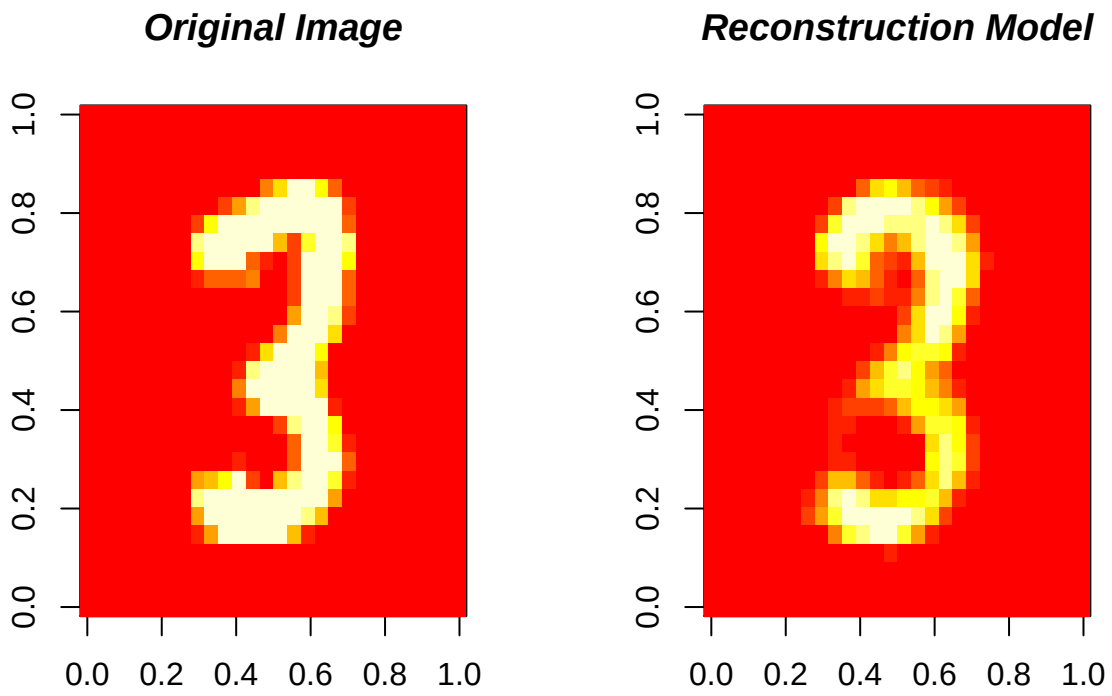
Nous pouvons commencer à utiliser la fonction RBM. Je vais fournir un petit exemple de la façon d'utiliser la fonction RBM.

```
# Turn plot on
#RBM(x = train, plot = TRUE, n.iter = 10, n.hidden = 3, size.minibatch = 3)
```

On a fait un gif des parcelles afin qu'on puisse voir à quoi il devrait ressembler :

Après la formation du modèle RBM, nous pouvons vérifier la façon dont il reconstruit les données avec la fonction ReconstructRBM () :

```
# Get the test data from MNIST
test <- MNIST$testX
# Reconstruct the image with modelRBM
ReconstructRBM(test = test[6, ], model = modelRBM)
```



La fonction va ensuite sortir l'image d'origine avec l'image reconstruite à côté. Si le modèle est tout bon l'image reconstruite devrait ressembler à semblable ou même mieux que l'original :

5.5.3 RBM() pour classification

IlPermet d'essayer de classer les étiquettes du jeu de données MNIST en formant une classification RBM. Nous pouvons utiliser la fonction RBM () à nouveau, la seule différence est que nous fournissons maintenant également les étiquettes comme argument y :

```
# First get the train labels of MNIST
TrainY <- MNIST$trainY
# This time we add the labels as the y argument
modelClassRBM <- RBM(x = train, y = TrainY, n.iter = 1000, n.hidden = 100, size.minibatch = 10)
```

Maintenant que nous avons formé notre classification RBM nous pouvons l'utiliser pour prédire les étiquettes sur certaines données de test invisibles avec la fonction PredictRBM () :

```
# First get the test labels of MNIST
TestY <- MNIST$testY
# Give our ClassRBM model as input
PredictRBM(test = test, labels = TestY, model = modelClassRBM)
```

```
## $ConfusionMatrix
##      truth
## pred   0   1   2   3   4   5   6   7   8   9
##    0 188   0   3   7   2   5   3   1   5   5
##    1   0 200   1   1   0   2   0   0   1   0
##    2   0   1 153   2   3   1   4   2   2   2
```

```
##    3   0   4  12 154   3   9   2   0   7   3
##    4   0   0   0   0 186   4   4   1   4  10
##    5   2   2   4  24   6 123   8   2   7   6
##    6   1   1   5   3   4   4 195   0   2   0
##    7   0   4   4   2   1   1   0 187   2  17
##    8   5  10   6   5   6  11   0   1 148   5
##    9   1   3   2   0  15   2   0   8   1 157
##
## $Accuracy
## [1] 0.8455
```

Ce qui devrait produire une matrice de confusion et le score de précision sur le jeu de test :

Pas mal pour un premier essai avec le RBM ! Une précision de 85%. Nous pourrions encore améliorer notre classification en effectuant un réglage hyper-paramètresur la (1) régularisation, (2) Momentum, (3) nombre d'époques et la taille des minilots (pour plus d'informations sur ces termes ? RBM). Toutefois, pour la taille du mini-lot, il est recommandé de le faire de la même taille que le nombre de classes dans les données que la fonction RBM prend toujours un échantillon équilibré, pour l'exemple actuel, cela signifie que la fonction RBM prend un échantillon aléatoire de chaque chiffre (0 :9). #Démonstration d'un logiel pour model de réseaux profonds(sauf PMC)

5.6 Présentation des réseaux RBF

L'idée générale des réseaux RBF dérive de la théorie d'approximation des fonctions, ces réseaux sont une architecture Feedforward puissante. Ce type de réseaux a été introduit pour la première fois par Hardy, et la théorie correspondante a été développée par Powell, ensuite, ces réseaux ont pris le terme de réseaux de neurones grâce a Broomhead et Lowe. La raison de son application vient du fait que le réseau utilise des fonctions gaussiennes standard qui sont à symétrie radiale. Son apprentissage est basé sur l'algorithme K-means et l'algorithme des moindres carrées. Les réseaux de neurones RBFs, sont principalement utilisés pour résoudre des problèmes d'approximation de fonctions dans des espaces de grandes dimensions.

Ils sont lus adaptés, en raison d'apprentissage local. Ce type d'apprentissage peut rendre le processus d'entrainement bien plus rapide que dans le cas d'un MLP, qui apprend de façon globale.

5.7 Architecture générale d'un réseau RBF

Pour des raisons de simplicité, on a décidé de faire une petite dualité entre le réseau RBF et le PMC, en précisant les ressemblances et les différences entre les deux types des réseaux. Ce choix est justifié par la popularité des PMCs et leur vaste utilisation dans les applications industrielles. Un réseau de neurone de type RBF est un PMC spéciale, son architecture est identique a celle d'un PMC a une seule couche cachée donc on peut dire qu'il prend toutes les caractéristiques d'un PMC simple sauf qu'il différent en quelques points nous citons quelques uns :

— Le nombre des couches cachées :

Un réseau RBF ne peut contenir qu'une seule couche cachée, son architecture est fixée pour tous les problèmes à étudier.

— La fonction d'activation :

Le réseau RBF utilise toujours une fonction dite a base radiale centrée d'un point et munie d'un rayon.

— Les poids synaptiques :

Les poids entre la couche d'entrée et la couche cachée dans les modèles neuronaux de type RBF sont toujours d'une valeur d'unité, c'est-à-dire que l'information inscrite sur la couche d'entrée sera retransmise sans distorsion vers les neurones de la couche cachée.

En ce qui concerne les ressemblances entre un réseau RBF et un PMC, on peut mentionner quelques points :

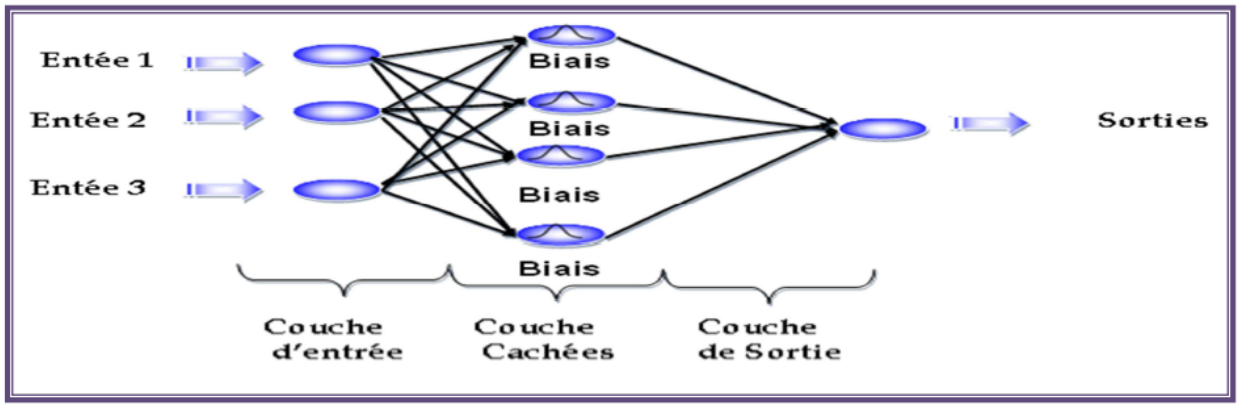


FIGURE 9 – Architecture d'un Réseau de Neurone RBF

— La fonction de sortie :

Généralement une simple fonction linéaire qui renvoie une sommation pondérée des valeurs calculées par les neurones de la couche cachée. Bien sur, ce n'est pas toujours le cas, parfois l'utilisation d'autres fonctions pourrait être plus adéquate dans un problème donné.

— Le sens des connexions :

Les connexions entre les couches suivent le même sens, on peut dire qu'elles ne sont pas récurrentes, et chaque neurone est entièrement connecté vers les neurones de la couche suivante.

— L'apprentissage :

Pour calculer les poids de la couche de sortie, on utilise un apprentissage supervisé pour les deux types de réseaux.

5.8 Application

5.9 approximation de fonction

Maintenant que nous avons une meilleure compréhension de la façon dont nous pouvons utiliser les réseaux neuronaux pour l'approximation de fonction, écrivons un peu de code !

Premièrement, nous devons définir nos données de «formation» et RBF. Nous allons coder notre RBF gaussien.

ici le code est commenter pour pouvoir l'afficher sous rmardown

```
#from scipy import *
#from scipy.linalg import norm, pinv

#from matplotlib import pyplot as plt

#import numpy as np

#def rbf(x, c, s):
    #return np.exp(-1 / (2 * s**2) * (x-c)**2)
```

Maintenant, nous aurons besoin d'utiliser l'algorithme de clustering k-means pour déterminer les centres de cluster. on a déjà codé une fonction qui nous donne les centres de cluster et les écarts types des clusters.

```
#def kmeans(X, k):
    #"""Performs k-means clustering for 1D input

    #Arguments:
        #X {ndarray} -- A Mx1 array of inputs
        #k {int} -- Number of clusters

    #Returns:
       # ndarray -- A kx1 array of final cluster centers
    #"""

    # randomly select initial clusters from input data
    #clusters = np.random.choice(np.squeeze(X), size=k)
    #prevClusters = clusters.copy()
    #stds = np.zeros(k)
    #converged = False

    #while not converged:
        #"""
        #compute distances for each cluster center to each point
       # where (distances[i, j] represents the distance between the ith point and jth cluster)
        #"""
        #distances = np.squeeze(np.abs(X[:, np.newaxis] - clusters[np.newaxis, :]))

        # find the cluster that's closest to each point
        #closestCluster = np.argmin(distances, axis=1)

        # update clusters by taking the mean of all of the points assigned to that cluster
        #for i in range(k):
            #pointsForCluster = X[closestCluster == i]
            #if len(pointsForCluster) > 0:
                #clusters[i] = np.mean(pointsForCluster, axis=0)

        # converge if clusters haven't moved
        #converged = np.linalg.norm(clusters - prevClusters) < 1e-6
        #prevClusters = clusters.copy()

    #distances = np.squeeze(np.abs(X[:, np.newaxis] - clusters[np.newaxis, :]))
    #closestCluster = np.argmin(distances, axis=1)

    #clustersWithNoPoints = []
    #for i in range(k):
        #pointsForCluster = X[closestCluster == i]
        #if len(pointsForCluster) < 2:
            # keep track of clusters with no points or 1 point
            #clustersWithNoPoints.append(i)
            #continue
        #else:
            #stds[i] = np.std(X[closestCluster == i])

    # if there are clusters with 0 or 1 points, take the mean std of the other clusters
    #if len(clustersWithNoPoints) > 0:
        #pointsToAverage = []
```

```
        #for i in range(k):
            #if i not in clustersWithNoPoints:
                #pointsToAverage.append(X[closestCluster == i])
        #pointsToAverage = np.concatenate(pointsToAverage).ravel()
        #stds[clustersWithNoPoints] = np.mean(np.std(pointsToAverage))

    #return clusters, stds
```

Ce code implémente simplement l'algorithme de clustering k-means et calcule les écarts types. S'il y a un cluster avec aucun ou un des points attribués à lui, nous avons simplement la moyenne de l'écart type des autres clusters. (Nous ne pouvons pas calculer l'écart type sans points de données, et l'écart type d'un point de données unique est 0).

Maintenant, nous pouvons arriver au cœur réel du réseau RBF en créant une classe.

```
#class RBFNet(object):
    #"""Implementation of a Radial Basis Function Network"""
   # def __init__(self, k=2, lr=0.01, epochs=100, rbf=rbf, inferStds=True):
        #self.k = k
        #self.lr = lr
        #self.epochs = epochs
        #self.rbf = rbf
        #self.inferStds = inferStds

        #self.w = np.random.randn(k)
        #self.b = np.random.randn(1)
```

Ensuite, nous devons écrire notre fonction d'ajustement pour calculer nos poids et biais. Dans les premières lignes, nous utilisons soit les écarts types de l'algorithme k-means modifié, soit nous forons toutes les bases à utiliser le même écart-type calculé à partir de la formule. Le reste est similaire à la rétropropagation où nous propagons notre entrée aller de l'avant et mettre à jour nos poids en arrière.

```
#def fit(self, X, y):
    #if self.inferStds:
        # compute stds from data
        #self.centers, self.stds = kmeans(X, self.k)
    #else:
        # use a fixed std
        #self.centers, _ = kmeans(X, self.k)
        #dMax = max([np.abs(c1 - c2) for c1 in self.centers for c2 in self.centers])
        #self.stds = np.repeat(dMax / np.sqrt(2*self.k), self.k)

    # training
    #for epoch in range(self.epochs):
        #for i in range(X.shape[0]):
            # forward pass
            #a = np.array([self.rbf(X[i], c, s) for c, s, in zip(self.centers, self.stds)])
            #F = a.T.dot(self.w) + self.b

            #loss = (y[i] - F).flatten() ** 2
            #print('Loss: {0:.2f}'.format(loss[0]))

            # backward pass
            #error = -(y[i] - F).flatten()
```

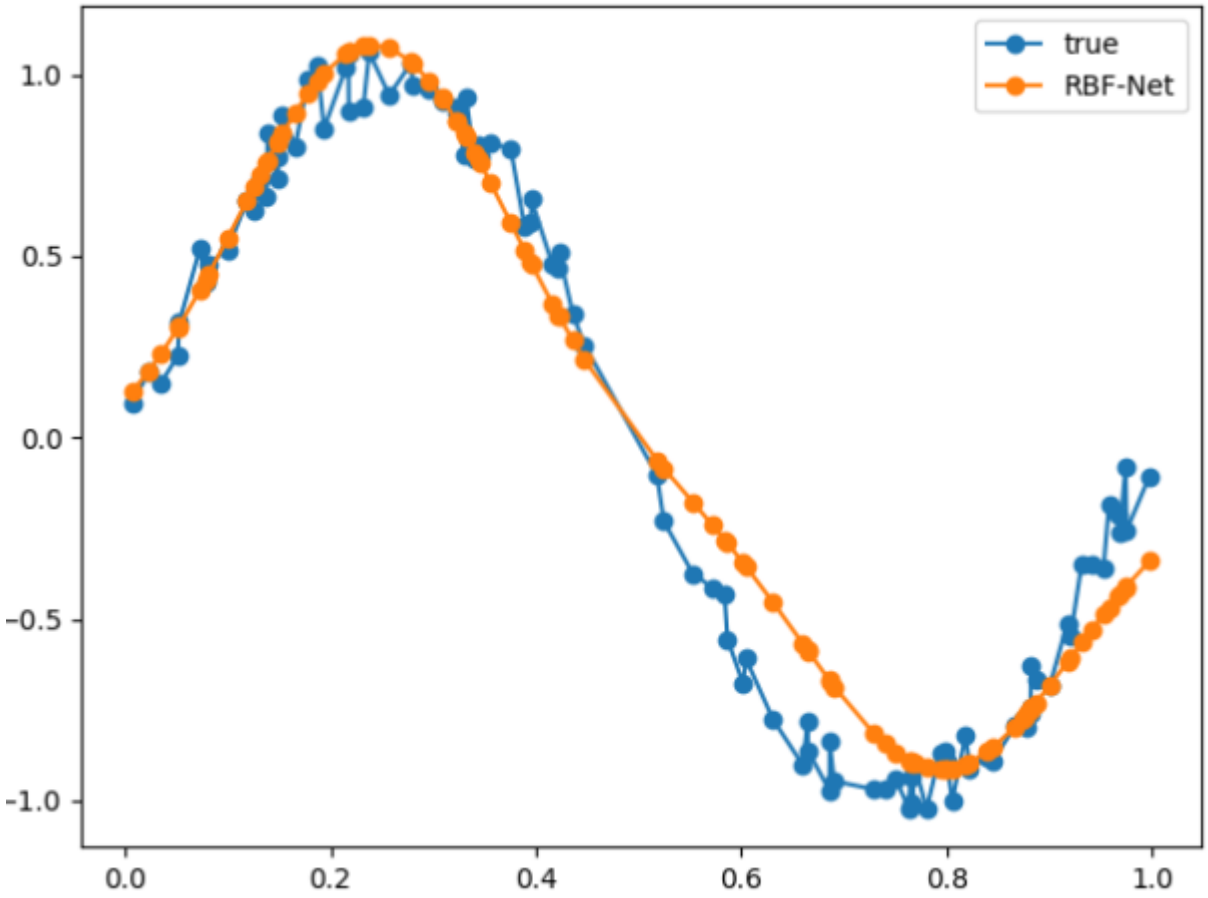


FIGURE 10 –

```
            # online update
            #self.w = self.w - self.lr * a * error
            #self.b = self.b - self.lr * error
```

Pour la verbosité, nous imprimons la perte à chaque étape. Remarquez que nous exécutes également une mise à jour en ligne, ce qui signifie que nous mettons à jour nos pondérations et les biais chaque entrée.

Faire une prédiction est aussi simple que de propager notre entrée en avant.

```
#def predict(self, X):
   # y_pred = []
    #for i in range(X.shape[0]):
       # a = np.array([self.rbf(X[i], c, s) for c, s, in zip(self.centers, self.stds)])
        #F = a.T.dot(self.w) + self.b
        #y_pred.append(F)
    #return np.array(y_pred)
```

Enfin, nous pouvons écrire du code pour utiliser notre nouvelle classe. Pour nos données de formation, nous allons générer 100 échantillons à partir de la fonction sinus. Ensuite, nous ajouterons un certain bruit uniforme à nos données.

Nous pouvons tracer notre fonction approximative contre notre fonction réelle pour voir comment est bien effectué notre filet de RBF effectué.

```
# sample inputs and add noise
#NUM_SAMPLES = 100
#X = np.random.uniform(0., 1., NUM_SAMPLES)
#X = np.sort(X, axis=0)
#noise = np.random.uniform(-0.1, 0.1, NUM_SAMPLES)
#y = np.sin(2 * np.pi * X)  + noise

#rbfnet = RBFNet(lr=1e-2, k=2)
#rbfnet.fit(X, y)

#y_pred = rbfnet.predict(X)

#plt.plot(X, y, '-o', label='true')
#plt.plot(X, y_pred, '-o', label='RBF-Net')
#plt.legend()

#plt.tight_layout()
#plt.show()
```

De nos résultats, notre filet de RBF s'est très bien produit ! Si nous voulions évaluer notre réseau de RBF plus rigoureusement, nous pourrions échantillonner plus de points de la même fonction, passer à travers notre filet de RBF et utiliser la distance euclidienne additionnée en tant que métrique.

6 Conclusion

Contrairement aux méthodes classiques qui ont montré leurs limites, les réseaux de neurones ont montré leurs tendances à s'adapter à des problèmes complexes grâce à leur grande capacité de calcul et d'apprentissage. Ils sont l'objet d'utilisation dans les différents domaines tels que : La reconnaissance des formes et le traitement des images.

Le grand avantage caractérisé dans le PMC : * Accepte les données bruitées et la classification non-linéaire * Représentation globale de l'espace * Architecture simple

Le grand avantage caractérisé dans les réseaux de neurones de Kohonen est que ces derniers sont léger en cout et en calcule et sont portable dans différents domaines, ce qui les rend les plus simples à utiliser et les plus rapides.

Le grand avantage caractérisé dans les réseaux de neurones à base radiale est que ces derniers n'ont qu'une seule couche cachée, ce qui les rend les plus simples à utiliser et les plus rapides.

Les machines Boltzmann restreints sont simples et complexes en même temps. Le mécanisme d'entrée-sortie RBM est déterministe et PROBABILISTE (parfois appelé stochastique), mais il est relativement facile à comprendre et à implémenter.

7 Réferences

— 1.http://www.di.fc.ul.pt/~jpn/r/neuralnets/neuralnets.html
— 2.http://univ.ency-education.com/uploads/1/3/1/0/13102001/mi-les_rseaux_de_neurones_artificiels.pdf
— 3.http://www.r-tutor.com/deep-learning/introduction
— 4.http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/fr_Tanagra_Kohonen_SOM_R.pdf
— 5.coursM2ISUP_RN.pdf
— 6.cours_ReseauxNeurones.pdf
— 7.les-rc3a9seaux-de-neurone-rbf.pdf