TIPE : Les Réseaux de Neurones Formels

Joël Felderhoff

Introduction

Les Réseaux de Neurones Formels (en abrégé RNF) sont une catégorie d'objets mathématiques utilisés dans de nombreux domaines de l'approximation de fonction, dans la classification et la séparation.
L'intéret de ces réseaux tient dans le fait qu'ils permettent un apprentissage : ils sont capable de généraliser un nombre fini d'exemples afin d'en tirer une fonction, qui donnera une image pour toute entrée
Ces réseaux sont utilisés dans des situations où un très grand nombre de paramètres doivent être gérés : reconnaissance faciale, détection de réseaux piratés sur internet, recherche de "patterns" dans les comportements d'utilisateurs à l'aide du Big Data...
Nous nous contenterons ici d'étudier un cas particulier de RNF : les perceptrons multicouches.

1 Présentation des RNF

Les notions étudiée étant assez graphiques, des schémas sont disponibles en annexe.

1.1 Le Neurone Formel

Un neurone formel a e entrées est un triplet $N = ((u_i)_{1\leq i\leq e}, f, b)$
Les u_i sont les coefficients synaptiques (ou poids) du neurone.
f est la fonction d'activation du neurone.
b est appellé coeficient de biais.
Le neurone à e entrée défini donc une fonction de sortie :

$$\begin{aligned} s\colon \mathbb{R}^e &\to \mathbb{R} \\ (x_i)_{1\leq i\leq e} &\mapsto f\left(\sum_{k=1}^{e} u_i x_i + b\right) \end{aligned}$$

1.2 La Couche

Une couche de n neurones à e entrées est un n-uplets de neurones à e entrées. Usuellement, ces neurones ont tous une même fonction d'activation f.
L'intéret d'une couche est d'étendre la sortie des neurones à $\mathbb{R}^n$: les poids des neurones peuvent alors être regroupés dans une matrice $W = (w_j^i)_{1\leq i\leq e, 1\leq j\leq n}$ où w_j^i est le jème coeficient synaptique du ième neurone.
De même, les coeficients de biais peuvent être regroupés dans un vecteur de $\mathbb{R}^n$, que l'on note B.
On étend de manière triviale la fonction d'activation f de la couche à $\mathbb{R}^n$, la sortie d'une couche de n neurones à e entrée est alors définie par :

$$\begin{aligned} S\colon \mathbb{R}^e &\to \mathbb{R}^n \\ X &\mapsto f\left(W \cdot X + B\right) \end{aligned}$$

On peut donc représenter une couche à n neurones à e entrées par un triplet

$$(W, f, B) \text{ avec } W \in M_{n,e}(\mathbb{R}),\ f : \mathbb{R}^n \to \mathbb{R}^n,\ B \in \mathbb{R}^n$$

1.3 Le Réseau

Finalement, un réseau de neurones à c couches, à e entrées à s sorties est un ensemble de couches "reliées" les unes aux autres.
Plus formellement : on note $t_0 = e, t_c = s$ et on fixe $t_i\ \forall\ 1 \leq i \leq c-1$ la taille de chacunes des couches. Alors la ième couche st une couche à t_{i-1} entrées et t_i sorties.
La ième couche est : $(W^{(i)}, f_i, B^{(i)})$.
La sortie d'un réseau est alors la composée des sorties de ses différentes couches. Avec l'entrée $E \in \mathbb{R}^e$

$$\begin{aligned} &Y_0 = E \\ &\forall\ 1 \leq i \leq c,\quad Y_i = f_i(W^{(i)} \cdot Y_{i-1} + B^{(i)}) \end{aligned}$$

Alors la sortie du réseau est $S = Y_n$.

Note

Les réseaux avec un nombre de couches, une entrée, une sortie, des tailles de couches et des fonctions d'activations fixées ne sont plus alors que des multiplets de matrices et de vecteurs. On peut donc leur donner une structure canonique d'espace vectoriels, ce qui sera utile pour la suite.

1.4 Notre étude

Nous étudierons ici un type précis de RNF : les perceptrons à deux couches. Ce sont des réseaux à deux couches : une couche dite cachée, et une couche dite de sortie.
Nous choisirons pour la couche cachée une fonction d'activation sigimoide (ce terme sera défini dans la partie suivante), et pour la couche de sortie, la fonction identité ($id : x \mapsto x$).
Si ces réseaux peuvent sembler assez simplistes, ils sont en fait largement suffisant pour approximer n'importe quelle fonction continue (c'est ce que nous verrons dans la section 2).
En notant σ une fonction sigimoide, e la taille de l'entrée, l'espace des fonctions représentable par un réseau à deux couches et une sortie est :

$$E = \left\{ \begin{array}{ll} g\colon \mathbb{R}^e \to \mathbb{R} & n \in \mathbb{N} \\ & (b_i) \in \mathbb{R}^n \\ x \mapsto \sum_{i=1}^{n} \alpha_i \sigma\left({}^t y_i \cdot x + b_i\right) & , \ (y_i) \in (\mathbb{R}^e)^n \\ & (\alpha_i) \in \mathbb{R}^n \end{array} \right\} = vect < \left(x \mapsto \sigma({}^t y \cdot x + b)\right)_{b\in\mathbb{R}, y\in\mathbb{R}^e} >$$

On peut identifier tous les termes à des valeurs caractéristiques de notre réseau : n est la taille de la couche cachée, les b_i sont les coeficients de biais de la couche cachée, les y_i sont les vecteurs des coeficients synaptiques des neurones de la couche cachée, et les α_i sont les coeficients synaptiques de la couche de sortie (qui est prise dans notre cas avec un coeficient de biais nul).

Tous les résultats pourrons s'étendre trivialement aux fonctions à valeur dans $\mathbb{R}^s$.

1.5 Ma production informatique

Lors de mon étude des Réseaux de Neurones formels, j'ai été amené à écrire un programme permettant de gérer des réseaux de neurones formels.
Mon programme permet de sauvegarder des réseaux de neurones formels (pour une utilisation ultérieure ou pour un entrainement supplémentaire). J'ai aussi codé une fonction permettant de renvoyer la sortie du réseau, et une fonction pour les entrainer.
Si toutes ces fonctionalités ont été codée en c++, qui est un langage compilé donc présentant une vitesse d'exécution moyenne plus grande que le python, j'ai fais en sorte de pouvoir utiliser mon code c++ avec python, afin de concevoir des routines d'apprentissages et de tracé de graphe plus rapide à coder (le c++ étant un langage assez complexe).
La représentation en mémoire de mon programme a été faite selon l'étude effectuée precedemment : un réseau est représenté en mémoire par un tableau de vecteurs, de matrices et de fonctions, qui sont sauvegardés dans des fichiers.
J'ai utilisé pour coder ces structures de données (non présentes dans le langage à la base) la bibliothèque Armadillo, spécialisée dans l'algèbre linéaire.
J'ai effectué l'entrainement de mes réseaux à l'aide de la méthode de descente de gradient étudiée dans la section 3 de mon exposé, cela m'a permis d'obtenir des résultats graphiques présentés en annexe, j'ai donc validé l'étude effectuée.

2 Le théorème d'approximation universelle

Le théorème d'approximation universelle, prouvé par G. Cybenko en 1989, justifie que les perceptrons à deux couches sont un choix judicieux dans l'approximation de fonction continue : l'espace des fonctions représentable est dense pour la norme infinie dans l'espace des fonctions continues.

2.1 Les mesures

Soit E un ensemble, T une tribu sur cet ensemble. Une mesure signée sur T est une application $\mu : T \mapsto \mathbb{R} \cup \{+/-\infty\}$ vérifiant les axiomes suivants :

$$\mu(\varnothing) = 0$$

$$\text{Pour toute suite d'ensemble 2 à 2 disjoint de } T, (A_i)_{i\in\mathbb{N}} \ : \ \mu\left(\bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} \mu(A_n)$$

Une mesure finie est une mesure qui ne prend jamais une valeur infinie.
La tribu borélienne sur $\mathbb{R}^n$ est la tribu engendrée par les pavés fermés de $\mathbb{R}^n$, une mesure borélienne est une mesure définie sur cette tribu.

On défini l'intégration par rapport à une mesure, que l'on note : $\int_{\mathbb{R}^n} f(x)d\mu(x)$ elle est définie sur les fonctions à valeurs dans $\mathbb{R}$ ou $\mathbb{C}$ mesurable par la mesure μ.
Une fonction f est dite intégrable pour μ lorsque l'intégrale par rapport à la mesure μ de $|f|$ est finie
Cette nouvelle intégration "prolonge" l'intégration de Riemann étudiée en cours et a des propriétés similaires (notament, elle est linéaire, croissante...).
Elle possède de nouvelles propriétés, mais seules certaines nous serons utiles pour prouver notre théorème :

2.1.1 Mesure de fonctions indicatrices

Soit $A \in T$, en notant L la fonction indicatrice de cet ensemble, on a $\int_E L d\mu = \mu(A)$.

2.1.2 Fonctions étagée

L'espace vectoriel des combinaisons linéaires de fonctions indicatrices (les fonctions étagées) sont dense dans l'espace vectoriel des fonctions mesurables.

2.1.3 Convergence dominée

Soit $(f_\lambda)_{\lambda\in\mathbb{R}}$ de fonction mesurables sur $(\mathbb{R}^n, T, \mu)$ convergeant simplement (en l'infini) vers une fonction f.
Si $\exists g$ intégrable tel que $\forall\lambda \in \mathbb{R}\forall x \in \mathbb{R}^n |f_\lambda(x)| \leq g(x)$
Alors $\lim\limits_{\lambda\to\infty}(\int_{\mathbb{R}^n} f_\lambda(x)d\mu(x)) = \int_{\mathbb{R}^n} f(x)d\mu(x)$
C'est en fait le même théorème que dans le cas de l'intégration étudiée en cours.

2.1.4 Transformée de Fourier d'une mesure

On défini la transformée de Fourier d'une mesure μ de $\mathbb{R}^n$ par :

$$\forall y \in \mathbb{R}^n \quad \widehat{\mu}(y) = \int_{\mathbb{R}^n} exp(i\ {}^t yx)d\mu(x)$$

Théorème : (admis) La transformée de Fourier est injective sur les mesure boreliennes signées finies.
En particulier, la seule mesure borelienne finie de transformée de Fourier nulle est la mesure nulle.

2.2 Quelques théorèmes

2.2.1 Théorème de Hahn Banach et corollaire

Soit E un $\mathbb{R}$ Espace vectoriel de dimension quelquonque
Soit $p : E \mapsto \mathbb{R}_+$. Cette fonction est une sous norme si :

$$\forall t \geq 0 \quad \forall x \in E \quad p(tx) = tp(x)$$
$$\forall (x, y) \in E^2 \ p(x + y) \leq p(x) + p(y)$$

Théorème d'Hahn Banach (preuve en annexe) :
Soit p une semi norme sur E, G un sous espace vectoriel de E, et $\varphi \in G^*$ vérifiant
$\forall x \in G\ \varphi(x) \leq p(x)$
Alors il existe $\varphi^* \in E^*$ prolongeant φ tel que
$\forall x \in E \quad \varphi^*(x) \leq p(x)$

Corollaire (preuve en annexe) :
Pour tout sous espace G fermé de E il existe une forme linéaire φ sur E continue non nulle telle que $G \subseteq ker(\varphi)$

2.2.2 Théorème de Riesz-Markov (admis)

Posons $E = C^0([0, 1]^n, \mathbb{R})$, soit $\varphi \in E^*$ alorsil existe une unique mesure borelliene finie μ telle que

$$\forall f \in E\ ,\ \varphi(f) = \int_{[0,1]^n} f d\mu$$

2.3 Théorème d'approximation universelle de Cybenko

Dans tout la suite, on note $I_n = [0, 1]^n$ ainsi que $M(I_n)$ l'ensemble des mesures boreliennes finies sur I_n

2.3.1 Fonction discriminante

Soit $\sigma : \mathbb{R} \to \mathbb{R}$ continue σ est dite discriminante si

$$\forall \mu \in M(I_n),$$
$$(\forall y \in \mathbb{R}^n \ , \ \forall t \in \mathbb{R} \ , \ \int_{I_n} \sigma({}^t yx + t) d\mu(x) = 0) \implies \mu = 0$$

2.3.2 Théorème de Cybenko

Soit σ une fonction discrimante, on note $E = C^0(I_n, \mathbb{R})$, on note

$$G = \left\{ \begin{array}{ll} g \colon \mathbb{R}^n \to \mathbb{R} & u \in \mathbb{N} \\ & (b_i) \in \mathbb{R}^u \\ x \mapsto \sum_{i=1}^{u} \alpha_i \sigma\left({}^t y_i \cdot x + b_i\right) & (y_i) \in (\mathbb{R}^n)^u \\ & (\alpha_i) \in \mathbb{R}^u \end{array} \right\} = vect < \left(x \mapsto \sigma({}^t y \cdot x + b)\right)_{b \in \mathbb{R}, y \in \mathbb{R}^n} >$$

Alors G est dense pour la norme infinie dans E.

2.3.3 Preuve du théorème

Nous allons raisonner par l'absurde : supposons que G ne soit pas dense, posons $F = \overline{G}$ F est alors un sous espace strict fermé de E. Par le corollaire au théorème de Hahn Banach, il existe $L \in E^*$ continue non nulle sur E nulle sur F.
Par le théorème de Riesz-Markov, il existe $\mu \in M(I_n)$ non nulle telle que

$$\forall f \in E, \ L(f) \int_{[0,1]^n} f d\mu \qquad \text{Or, comme L est nulle sur G, } \forall y, t \in \mathbb{R}^n \times \mathbb{R}, \ \int_{I_n} \sigma({}^t yx + t) d\mu(x) = 0$$

Donc $\mu = 0$, absurde. Donc $E = \overline{G}$

2.4 Fonction sigimoide

Soi $\sigma : \mathbb{R} \to \mathbb{R}$ continue. Elle est dite sigimoide si

$$\left\{ \begin{array}{l} \lim\limits_{x \to +\infty} \sigma(x) = 1 \\ \lim\limits_{x \to -\infty} \sigma(x) = 0 \end{array} \right.$$

2.4.1 Théorème

Toute fonction sigimoide est discriminante
Cela justifie donc le choix d'une fonction sigimoide en fonction d'activation de couche cachée de notre perceptron à deux couches

2.4.2 Demonstration

On fixe σ sigimoide, $\mu \in M(I_n)$ telle que $\forall y \in \mathbb{R}^n \ , \ \forall t \in \mathbb{R} \ , \ \int_{I_n} \sigma({}^t yx + t) d\mu(x) = 0$, montrons que μ est nulle.

$\forall y \in \mathbb{R}^n, t \in \mathbb{R}, \psi \in \mathbb{R}$
$\forall \lambda \in \mathbb{R}, \forall x \in \mathbb{R}^n \sigma_\lambda(x) = \sigma(\lambda({}^t yx + t) + \psi)$
On a $\forall \lambda \in \mathbb{R} \ , \ \int_{I_n} \sigma_\lambda d\mu = 0$

$$\sigma_\lambda \underset{\lambda \to +\infty}{\longrightarrow} \gamma \text{ avec } \forall x \in \mathbb{R}^n \ \gamma(x) = \left\{ \begin{array}{ll} 1 \text{ si } {}^t yx + t > 0 & \\ 0 \text{ si } {}^t yx + t < 0 & \text{(Convergence simple)} \\ \sigma(\psi) \text{ si } {}^t yx + t = 0 & \end{array} \right.$$

σ est bornée, donc $\forall \lambda \in \mathbb{R} \forall x \in \mathbb{R}^n, \ |\sigma_\lambda(x)| \leq \|\sigma\|$ Qui est une constante, intégrable donc car μ est finie
On pose : $\Pi_{t,y} = \{x \ , \ {}^t yx + t = 0\}$ et $H_{y,t} = \{x \ , \ {}^t yx + t > 0\}$
Par théorème de convergence dominée, $0 = \int_{I_n} \gamma(x) d\mu(x) = \sigma(\psi)\mu(\Pi_{t,y}) + \mu(H_{y,t})$
En faisant tendre ψ vers l'infini, on a $\mu(H_{y,t}) = 0$ et $\mu(\Pi_{t,y}) + \mu(H_{y,t}) = 0 \ , \forall y, t$

Montrons que cela implique que $\mu = 0$

On fixe $y \in \mathbb{R}^n$ Pour toute fonction h bornée mesurable, on pose $F(h) = \int_{I_n} h({}^t x) d\mu(x)$

Note : F est une fonction continue car μ est finie

Posons h la fonction indicatrice de $[t, \infty[$Alors par ce que nous avons démontré, $F(h) = 0$

De même, avec h fonction indicatrice de $]t, \infty[$ $F(h) = 0$

Alors par linéarité, l'image d'une fonction étagée par F est nulle

Or, ces fonctions sont denses dans l'espace des fonctions continues mesurables sur $\mathbb{R}$, donc $F = 0$

Donc : $F(cos) + iF(sin) = 0 = \int_{\mathbb{R}^n} exp(i\ {}^t yx) d\mu(x)\ \forall y \in \mathbb{R}^n$

Donc $\mu = 0$ CQFD

3 L'approximation en elle même

Nous avons pu voir que chaque fonction peut être approchée par un réseau de neurones. Néanmoins, cela ne nous donne aucune information sur les différents facteurs rentrant en compte. Cette section va détailler leur calcul.
L'idée est de modifier de manière itérative les coefficients synaptiques et de biais des neurones de nos réseaux, par une méthode de descente.
Nous aborderons ici l'apprentissage supervisé d'un réseau de neurone quelquonque. Cela consiste à la présentation au réseau d'un nombre fini de couples (entrée, sortie attendue), appellé ensemble d'entrainement.
On va présenter cet ensemble un certain nombre de fois à notre réseau (en fait jusqu'à ce qu'une condition d'arrêt soit atteinte), en cherchant à chaque fois à minimiser la fonction d'erreur suivante :

$$F_E(R) = \|S - g(R, E)\|^2$$

avec R le réseau, (E, S) le couple entrée/sortie et $g(R, E)$ la sortie du réseau R pour l'entrée E.
Nous supposerons que toutes les fonctions utilisées sont de classe C^1.
En se rappellant que l'ensemble des réseaux de neurones de grandeurs caractéristiques fixées est un espace vectoriel, on se rammenne finalement à un "simple" problème de minimisation d'une fonction $E \to \mathbb{R}$ qui est différentiable.

3.1 Algorithme de descente

Les algorithmes de descente sont une classe d'algorithmes ayant pour but la minimisation d'une fonction $f : E \to \mathbb{R}$ différentiable.
Descente : (on a fixé x_0)

Tant que une condition d'arrêt n'est pas vérifiée

- on choisi une direction de descente $d_k \in E$
- on choisi $t_k \in \mathbb{R}$ telle que $f(x_k + td_k)$ soit minimale
- $x_{k+1} = x_k + t_k d_k$

La différence entre les différents algorithmes se trouve dans le choix de la direction de descente.
Dans notre cas, nous utiliserons l'algorithme de descente de gradient, où la direction de descente choisie est $-\nabla f(x_k)$.

3.2 Justification

Une direction d est une direction de descente en x lorsque la fonction $t \mapsto f(x + td)$ est décroissante sur un voisinage de zéro, c'est à dire que ${}^t\nabla f(x) \cdot h \leq 0$. En effet,

$$\forall t \in \mathbb{R}, f(x + th) = f(x) + t({}^t\nabla f(x) \cdot h) + o(t)$$
$$\text{Donc } \lim_{t \to 0} \frac{f(x + th) - f(x)}{t} = {}^t\nabla f(x) \cdot h$$

Et en fait, la direction $-\nabla f(x)$ est donc la direction de pente la plus forte (à normes égales). (preuve en annexe)

4 Annexe

4.1 Preuve du théorème d'Hanh Banach

4.1.1 Lemme de Zorn

On admet ici le lemme de Zorn, qui dit que tout ensemble inductif pour un ordre possède un maximum pour cet ordre.

4.1.2 La démonstration

On pose

$$B = \{\psi : D_\psi \to \mathbb{R}, G \subseteq D_\psi, D_\psi \ SeV \ de \ E, \psi \text{ forme linéaire}, \psi/_G = \varphi, \forall x \in D_\psi, \psi(x) \leq p(x)\}$$

On munit cet ensemble de la relation d'ordre $\prec$:
$\forall \psi_1, \psi_2 \in B^2, \psi_1 \prec \psi_2 \Leftrightarrow D_{\psi_2} \subseteq D_{\psi_1} \ et \ \psi_2$ prolonge ψ_1

Montrons que B est inductif pour cet ordre.
$\varphi \in B \Rightarrow B \neq \emptyset$
Soit A une partie totalement ordonnée de B. On pose : $D = \bigcup_{f \in A} D_f$ et $\psi : D \to \mathbb{R}$ telle que
$\forall x \in D, \psi(x) = f(x) \ si \ x \in D_f$
Cette contruction est valide car A est totalement ordonnée. Alors de manière évidente, ψ est un majorant de A.

B est bien inductif, donc il possède un élément maximal (un élément qui n'est prolongeable par aucun autre élément de B)
Notons cet élément $\varphi^* : D \to \mathbb{R}$
Nous allons donc montrer que $D = E$.
Raisonnons par l'absurde : supposons $D \neq E$
Soit $x_0 \in E \setminus D$
Le but va être de trouver α tel que la fonction ψ définie sur $H = D + vect < x_0 >$ par
$\forall (x,t) \in D \times \mathbb{R}, \psi(x+tx_0) = \varphi^*(x) + t\alpha$ prolonge φ^* (ie $\psi \in B$). On aura alors une contradiction, le théorème sera alors prouvé.

$$\begin{aligned}
&\psi \in B \\
&\Leftrightarrow \forall (x,t) \in D \times \mathbb{R}, \psi(x+tx_0) \leq p(x+tx_0) \\
&\Leftrightarrow \forall (x,t) \in D \times \mathbb{R}, \varphi^*(x) + t\alpha \leq p(x+tx_0) \qquad (1)
\end{aligned}$$

On a :

$$(1) \implies \forall x \in D, \begin{cases} \varphi^*(x) + \alpha \leq p(x+x_0) \\ \varphi^*(x) - \alpha \leq p(x-x_0) \end{cases} \qquad (2)$$

Réciproquement, supposons (2), montrons (1).

$$\begin{aligned}
&\text{Soient } x, t \in D \times \mathbb{R} \\
&si \ t > 0 \ : \ t(\varphi^*(\frac{x}{t}) + \alpha) \leq tp(\frac{x}{t} + x_0) \ \Leftrightarrow \ \psi(x+tx_0) \leq p(x+tx_0) \\
&si \ t < 0 \ : \ -t(\varphi^*(\frac{x}{-t}) - \alpha) \leq -tp(\frac{x}{-t} - x_0) \ \Leftrightarrow \ \psi(x+tx_0) \leq p(x+tx_0) \\
&si \ t = 0 \ : \ \varphi^*(x) \leq p(x) \text{ car } \varphi^* \in B
\end{aligned}$$

Donc (1) $\Leftrightarrow$ (2)

$$\begin{aligned}
&(1) \Leftrightarrow \forall x \in D, \begin{cases} \alpha \leq p(x_0+x) - \varphi^*(x) \\ \alpha \geq \varphi^*(x) - p(x-x_0) \end{cases} \\
&\Leftrightarrow sup\{\varphi^*(x) - p(x-x_0)\} \leq \alpha \leq inf\{p(x_0+y) - \varphi^*(y)\} \\
&\text{Or, } \forall (x,y) \in D^2 \ , \ \varphi^*(x) + \varphi^*(y) = \varphi^*(x+y) \leq p(x+y) = p(x-x_0+y+x_0) \leq p(x-x_0) + p(y+x_0) \\
&\Rightarrow \quad \varphi^*(x) - p(x-x_0) \leq p(y+x_0) - \varphi^*(y)
\end{aligned}$$

Donc on peut choisir un α qui convient, on a alors prouvé le théorème.

4.1.3 Corollaire

Pour tout sous espace G fermé de E il existe une forme linéaire φ sur E continue non nulle telle que $G \subseteq ker(\varphi)$
Preuve :

Soit $x_0 \in E \setminus G$, Posons A $= G + \mathbb{R}x_0$ $\forall \lambda \in \mathbb{R}$ $\forall x \in G$ Posons
$\psi(x + \lambda x_0) = \lambda$ Comme G est fermé, on peut poser $\delta = d(x_0, G) > 0$ Et alors :

$$|\psi(x + \lambda x_0)| = |\lambda| = \frac{1}{\delta} d(\lambda x_0, G) \leq \frac{1}{\delta} \|x + \lambda x_0\|$$

Donc ψ est continue, donc par le théorème de Hahn Banach, on peut la prolonger continuement sur E CQFD

4.2 Pente maximale d'une fonction différentiable

$\forall d$ direction de descente telle que $\|d\| = \|\nabla f(x)\|$,

$${}^t(-\nabla f(x)) \cdot \nabla f(x) \leq {}^t d \cdot \nabla f(x)$$

En effet :

$\forall d$ direction de descente telle que $\|d\| = \|\nabla f(x)\|$,
Par l'inégalité de Cauchy Schwartz :

$$\begin{aligned} {}^t d \cdot \nabla f(x) &\leq \|\nabla f(x)\| \cdot \|d\| \\ &\leq \|\nabla f(x)\|^2 = {}^t(\nabla f(x)) \cdot \nabla f(x) \text{ CQFD} \end{aligned}$$

A norme fixée, l'opposé du gradient est la direction de pente la plus forte.

4.3 Calcul du gradient de la fonction de coût d'un RNF

Gradient de la fonction de coût d'un réseau de neurones :

$$F_E(R) = \|g(R, E) - S\|^2$$

On étudie un réseau à n couches, avec toutes les notations introduites (t_i etc...)
Nous allons introduire quelques nouvelles notations.
La sortie du réseau se calcule de la façon suivante, avec E en entrée :

$Y_0 = E = (e_i)_i$

$1 \leq i \leq n-1$, $H_i = W^{(i)} \cdot Y^{(i-1)} + B^{(i)}$ et $Y^{(i)} = f_i(H_i)$

On a alors $Y^{(n)}$ en sortie

En notant $W^{(b)} = (w_{ij}^{(b)})_{ij}$ et $B^{(b)} = (B_i^{(b)})_i$,

$$F_E(R) = \sum_{k=1}^{t_n} (S_k - f_n \left(\sum_{a=1}^{t_n} w_{ka}^{(n)} e_a + B_k \right))$$

$$\text{Donc : } \frac{\partial F_E(R)}{\partial w_{ij}^{(n)}} = -2(S_i - f_n(H_i^{(n)})) f_n'(H_i^{(n)}) e_j$$

$$\text{Et : } \frac{\partial F_E(R)}{\partial B_i^{(n)}} = -2(S_i - f_n(H_i^{(n)})) f_n'(H_i^{(n)})$$

Etudions maintenant la dérivée par rapport à un poids dans une couche inférieure (la $b$ième), par la formule de la chaîne :

$$\frac{\partial F_E(R)}{\partial w_{ij}^{(b)}} = \sum_{a=1}^{t_b} \frac{\partial F_E(R)}{\partial Y_a^{(b)}} \cdot \frac{\partial Y_a^{(b)}}{\partial w_{ij}^{(b)}}$$

$$\frac{\partial F_E(R)}{\partial B_i^{(b)}} = \sum_{a=1}^{t_b} \frac{\partial F_E(R)}{\partial Y_a^{(b)}} \cdot \frac{\partial Y_a^{(b)}}{\partial B_i^{(b)}}$$

Calculons ces grandeurs :

$$\frac{\partial F_E(R)}{\partial Y_a^{(b)}} = \sum_{k=1}^{t_{b+1}} \frac{\partial F_E(R)}{\partial Y_k^{(b+1)}} \cdot \frac{\partial Y_k^{(b+1)}}{\partial Y_a^{(b)}} (\text{ Par la règle de la chaine})$$

$$\text{On a : } Y_a^{(b+1)} = f_{b+1}\left(\sum_{k=1}^{t_b} w_{ak}^{(b+1)} \cdot Y_k^{(b)} + B_a^{(b+1)}\right)$$

$$\text{Donc : } \frac{\partial Y_a^{(b+1)}}{\partial Y_i^{(b)}} = w_{ai}^{(b+1)} f'_{b+1}(H_a^{(b+1)})$$

Et (calcul trivial): $\frac{\partial Y_i^b}{\partial w_{ij}^{(b)}} = Y_j^{(b-1)} f'_b(H_i^{(b)})$

$$V^{(k)} = \left(\frac{\partial(S)}{\partial Y_i^{(k)}}\right)_i$$

Qui suit alors la relation de récurence descendante :

$$V^{(k)} = {}^tW^{(k+1)} \cdot \left(V^{(k+1)} * f'_{k+1}(H^{(k+1)})\right)$$

$$\text{Et : } V^{(n)} = -2(S - Y^{(n)}) * f'_n(H^{(n)})$$

Le gradient de notre fonction de coût sera alors un réseau de neurones, qu'on noteras $(W'^{(1)}, \cdots, W'^{(n)}, B'^{(1)}, \cdots, B'^{(n)})$, avec

$$W'^{(b)} = \left(\frac{\partial F_E(R)}{\partial w_{ij}^{(b)}}\right)_{ij}$$

$$B'^{(b)} = \left(\frac{\partial F_E(R)}{\partial B_i^{(b)}}\right)_i$$

On les a alors par la relation :

$$W'^{(b)} = (V^{(b)} * f'_b(H^{(b)})) \cdot {}^tY^{(i-1)}$$

$$B'^{(b)} = (V^{(b)} * f'_b(H^{(b)}))$$

5 Code de mon programme

J'ai codé mon programme en c++, qui a l'avantage d'avoir une vitesse d'exécution plus rapide que le python.
Le code total gérant une liaison avec Python, la sauvegarde et la chargement des réseaux, il fait à peu près un millier de ligne de code, qui ne sont pas interessante à toutes détailler ici, voici certains extraits choisis.
Le calcul du gradient pour un réseau :

```
ReseauStruct calculerGradient(ReseauStruct& reseau, arma::vec E, arma::vec sortieAttendue){
    unsigned& n = reseau.n;
    std::vector<arma::vec> Y(n+1), H(n+1);
    Y[0] = E;
    H[0] = E;
    for(unsigned i=1; i<=n; ++i){
        H[i] = reseau.poids_couches[i-1]*Y[i-1] + reseau.coefBiais[i-1];
        Y[i] = appliquerFonction(H[i], reseau.activ[i-1], 0);
    }

    std::vector<arma::vec> V(n+1);

    ReseauStruct gradient;

    gradient.n = n;
    gradient.activ = reseau.activ;
    gradient.poids_couches.resize(n);
    gradient.coefBiais.resize(n);
    gradient.tailles = reseau.tailles;

    V[n] = -2*(sortieAttendue-Y[n])%(appliquerFonction(H[n], reseau.activ[n-1], 1));
    gradient.poids_couches[n-1] = V[n]*(Y[n-1].t());
    gradient.coefBiais[n-1] = V[n];

    for(unsigned k = n-1; k >= 1; k--){
        V[k] = (reseau.poids_couches[k].t())*(V[k+1]%
                                    (appliquerFonction(H[k+1], reseau.activ[k], 1)));
        auto Hp = appliquerFonction(H[k], reseau.activ[k-1], 1);
        gradient.poids_couches[k-1] = (V[k]%Hp)*(Y[k-1].t());
        gradient.coefBiais[k-1] = V[k]%Hp;
    }

    return gradient;
}
```

Explication : les "std::vector" sont des tableaux, la multiplication matricielle (et scalaire) est notée avec le symbole "*", la multiplication terme à terme de vecteurs/matrice est notée par "%"

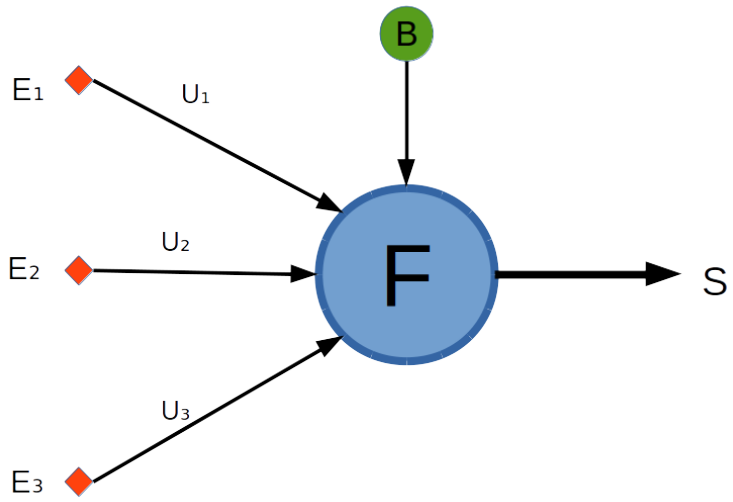
La descente de gradient :

```
double Reseau::descente_gradient(Ensemble& ens, double epsilon, unsigned pas){
    unsigned i = 0;
    double err = 0;
    for(auto exemple : ens.ens){
        vec sortieAtt = exemple.second;
        vec entree = exemple.first;

        vec sortie = resultat(entree);
        double e = norm(sortie - sortieAtt)*norm(sortie - sortieAtt);
        err += e;

        auto grad = calculerGradient(structure, entree, sortieAtt);
        double alpha = rechercher_alpha(epsilon, structure, grad, entree, sortieAtt, pas);
        mult_Reseau(grad, -alpha);
        add_Reseau(structure, grad);
    }
    return err;
}
```

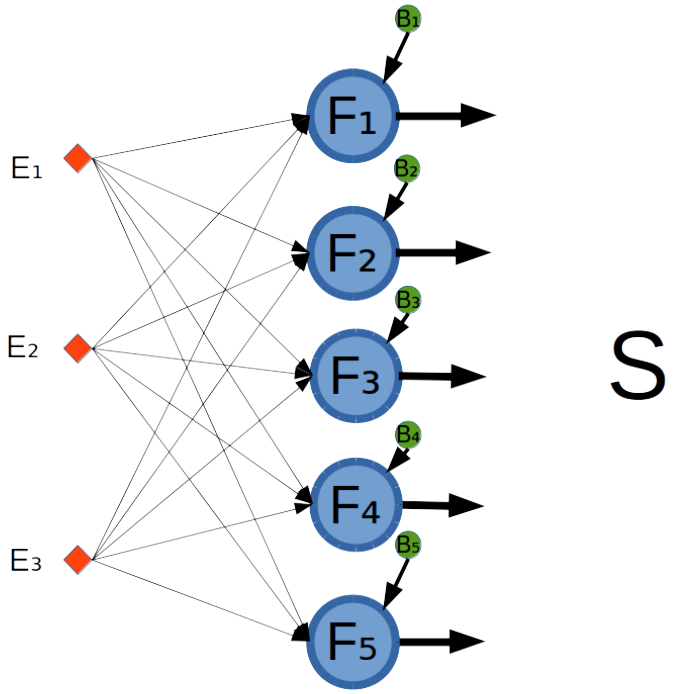
Explication : chaque "exemple" est un couple (entrée, sortie voulue), "rechercher_alpha" recherche un alpha minimisant la fonction de coût et la fonction calcule l'erreur globale sur l'ensemble d'apprentissage, qui est renvoyée.

6 Figures

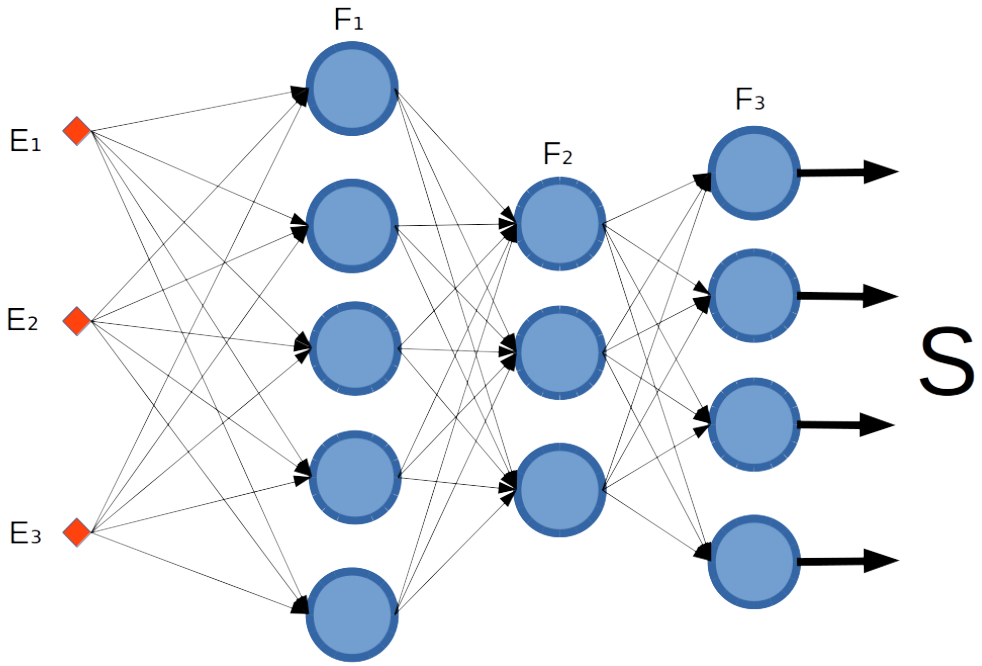
Un neurone



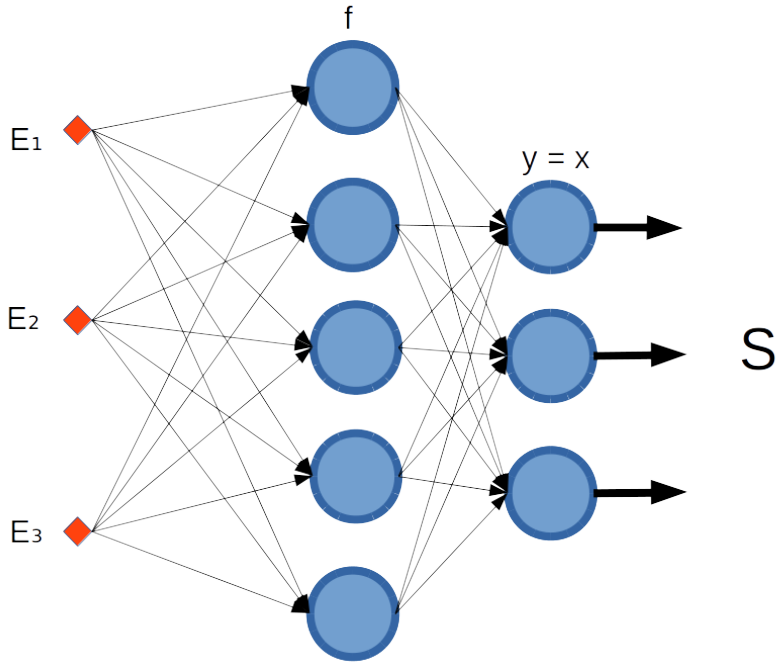
Une couche



Un RNF

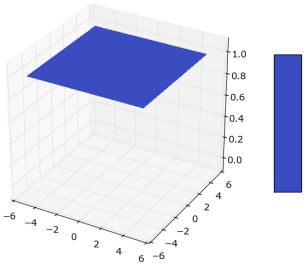


Le perceptron à deux couches

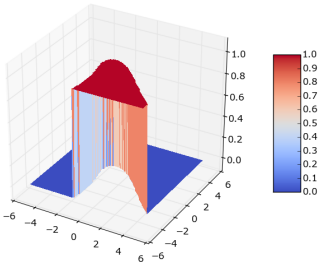


Quelques résultats

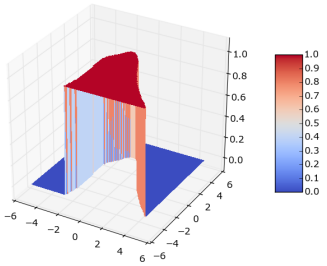
Mes réseaux sont des perceptrons à deux couches, ils leur entrée est dans $\mathbb{R}^2$ et leur sortie dans $\mathbb{R}$. Je leur ai à chaque fois présenté un certain nombre de points du plan (à peu près 2000) pour leur faire apprendre une forme. On remarque que plus la forme demandée est complexe, plus le nombre de neurones et le nombre d'itération necessaire pour avoir un apprentissage correct.
Apprentissage d'un disque (8 neurones dans la couche cachée):



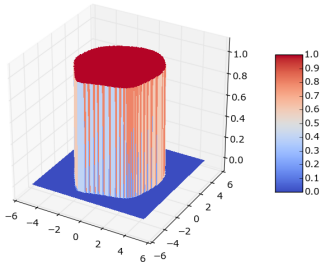
Avant apprentissage:



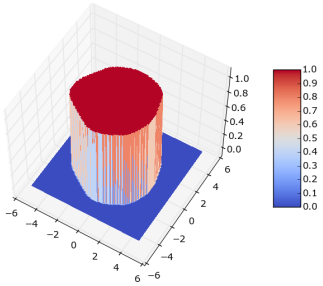
3 itérations :



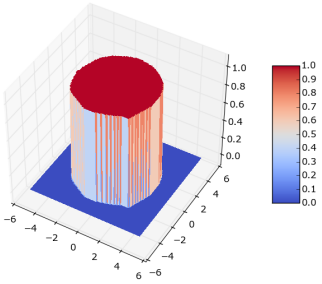
10 itérations :



50 itérations :

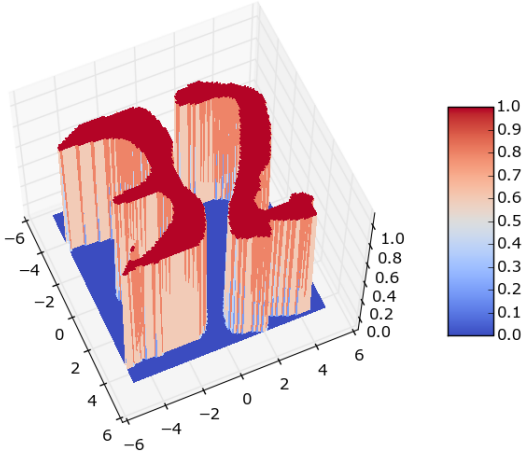


100 itérations:



1000 itérations:

Une forme plus complexe, avec 300 neurones cachés et à peu près 5000 itérations :



7 Bibliographie

Approximation by Superpositions of a Sigmoidal Function (G.Cybenko, 1989)
Analyse Fonctionnelle (Master 1 Mathématiques-Informatique) (Faculté des Sciences Jean Perrin) - pour la preuve du théorème d'Hahn Banach
Universite des Sciences et Technologies de Lille (U.F.R. de Mathematiques Pures et Appliquees - pour la notion de mesure
Page Wikipedia des réseaux de neurones formels