TIPE 2009

Réseaux de neurones artificiels

L'objet de ce TIPE est d'étudier et de comprendre les mécanismes d'apprentissage des réseaux de neurones de manière théorique dans un premier temps, pour ensuite évaluer et optimiser leurs performances par une approche expérimentale (au moyen d'un réseau de neurones que l'on aura mis au point).

Table des matières

Première partie

Présentation générale

1 Introduction

Un réseau de neurones artificiel est une structure composée d'entités capables de calcul et intéragissant entre eux, les neurones. Il permet de traiter, par le biais de l'informatique, des problèmes de différentes natures que les outils classiques ont du mal à résoudre. En effet, son fonctionnement s'inspire de celui des cellules neuronales animales, et est donc différent des méthodes de calcul analytiques que l'on utilise ordinairement. Il s'avère très puissant dans des problèmes de reconnaissance, classification, approximation ou prévision.

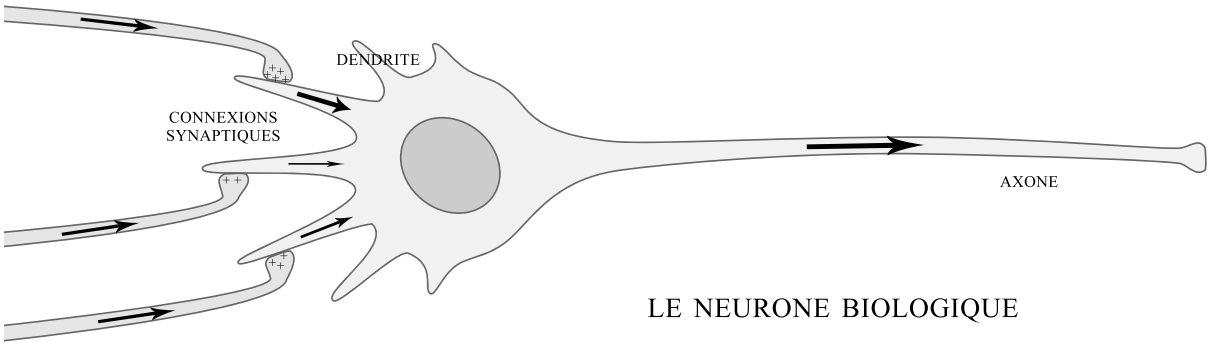
Étudions de plus près le fonctionnement d'un réseau neuronal classique.

2 Le modèle du neurone

2.1 Neurone biologique

En biologie, un neurone est une cellule nerveuse dont la fonction est de transmettre un signal électrique dans certaines conditions. Il agit comme un relai entre une couche de neurones et celle qui la suit. Les caractéristiques des neurones sont encore mal connues (et font l'objet de recherches) mais on connait leur principe d'action.

Le *corps* d'un neurone est relié d'une part à un ensemble de *dendrites* (entrées du neurones) et d'autre part à un *axone*, partie étirée de la cellule, qui représentera pour nous sa sortie. Le neurone étudié est connecté aux neurones qui l'environnent : il reçoit au niveau de ses dendrites les signaux électriques des neurones "en amont", propagés par les axones de ces derniers. Les charges électriques s'accumulent dans le neurone jusqu'à dépasser un certain seuil : à ce moment la transmission du signal électrique se déclenche *via* son axone vers d'autres neurones "en aval".



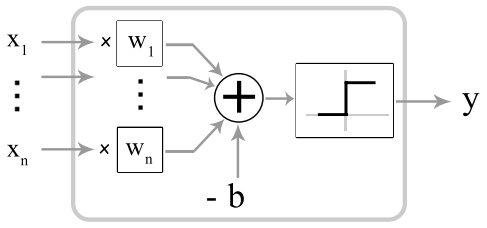
LE NEURONE BIOLOGIQUE

On remarque que les liaisons axone/dendrite entre deux neurones (connexions synaptiques) ne sont pas toutes de la même efficacité. Ainsi l'entrée associée à une certaine dendrite du neurone pourra avoir plus d'influence qu'une autre sur la valeur de sortie. On peut représenter la qualité de la liaison par un *poids*, sorte de coefficient s'appliquant au signal d'entrée. Le poids sera d'autant plus grand que la liaison est bonne. Un poids négatif aura tendance à inhiber une entrée, tandis qu'un poids positif viendra l'accentuer.

2.2 Neurone formel

Pour reproduire le neurone biologique, on se sert d'un modèle mathématique : le neurone formel. Il doit être capable de :

- recevoir en entrée différentes informations provenant des neurones environnants,
- analyser ces informations, de manière à envoyer en sortie une réponse,
- ajuster cette réponse avant de l'envoyer aux neurones suivants.



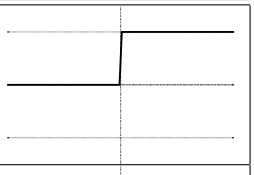
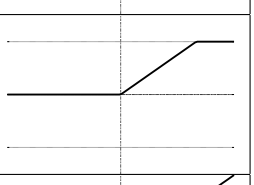
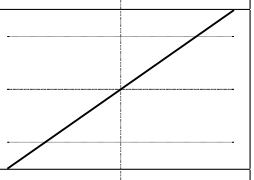
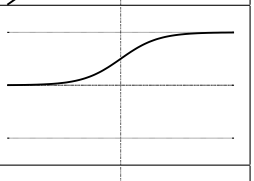
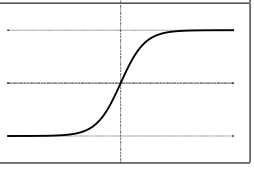
LE NEURONE FORMEL

Il est donc tout naturel d'assimiler un neurone à un triplet ($\overrightarrow{poids}$, *biais* , fonction d'activation f) :

- on multiplie chaque valeur d'entrée par la composante des $\overrightarrow{poids}$ correspondante, ce qui revient à faire le produit scalaire $\overrightarrow{entrées} \cdot \overrightarrow{poids}$,

– on compare la valeur obtenue à une valeur de référence : le biais, ce qui revient à soustraire le scalaire *biais*,
– enfin on applique la fonction d'activation à cette différence ; la fonction d'activation est souvent de façon à avoir une sortie comprise entre 0 et 1. Par exemple dans le cas d'une fonction d'activation de type seuil, la sortie sera :
0 si $\overrightarrow{entrées} \cdot \overrightarrow{poids} - biais \leq 0$
1 si $\overrightarrow{entrées} \cdot \overrightarrow{poids} - biais > 0$.

Le modèle mathématique sera plus amplement repris dans la deuxième partie, mais on peut déjà donner des exemples de fonctions d'activation utilisées pour les réseaux de neurones. Elles ressemblent le plus souvent à des fonctions *tout ou rien* :

Nom	Valeur	Représentation
Seuil	$f(x) = 0 \quad \text{si} \quad x \leq 0$ $f(x) = 1 \quad \text{si} \quad x > 0$	
Seuil symétrique	$f(x) = -1 \quad \text{si} \quad x < 0$ $f(0) = 0$ $f(x) = 1 \quad \text{si} \quad x > 0$	
Linéaire saturée	$f(x) = 0 \quad \text{si} \quad x < 0$ $f(x) = x \quad \text{si} \quad 0 \leq x \leq 1$ $f(x) = 1 \quad \text{si} \quad x > 1$	
Linéaire	$f(x) = x$	
Sigmoïde	$f(x) = \dfrac{1}{1+e^{-x}}$	
Tangente hyperbolique	$f(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	

3 Structure d'un réseau neuronal

En superposant dans une même *couche*, la couche 1, s_1 neurones formels à e_1 entrées, on obtient un système admettant e_1 entrées et s_1 sorties (chaque neurone produisant une sortie scalaire). Puis lorsqu'on concatène N couches connectées les unes à la suite des autres ($s_{n-1} = e_n$: le nombre d'entrées d'une couche étant égal au nombre de sorties de la précédente), on obtient un *réseau* de N couches de neurones. La N^e couche est appelée *couche de sortie*, tandis que les couches précédentes sont des *couches cachées*.

Le choix du nombre de couches et de neurones dépend du problème que l'on souhaite résoudre. En effet un réseau sera apte à traiter ce problème si les données de ce dernier fournissent e_1 variables d'entrées et nécessitent s_N variables en sortie.

Un exemple : on souhaite classer des images de $5 \times 5 = 25$ pixels dans 2 catégories, selon des critères dépendant de la couleur et de la disposition de ces pixels. La première couche de notre réseau devra comporter 25 entrées, une pour la valeur de chaque pixel, et la couche de sortie fournira 2 valeurs : les "affinités" respectives qu'aura l'image avec les deux catégories (si la première affinité est la plus grande, l'image sera classée dans la catégorie 1).

4 Entraînement d'un réseau

La particularité du réseau de neurones comme outil mathématique repose sur sa capacité d'apprentissage : grâce à un processus d'entraînement, on peut améliorer les performances du réseau, c'est-à-dire qu'en réponse à une certaine entrée, il fournisse la "bonne" sortie, le sortie qu'on voudrait obtenir.

Le principe est le suivant : on peut modifier les caractéristiques du réseau (les poids de ses liaisons par exemple) en réaction aux stimuli extérieurs qu'on lui soumet (les valeurs d'entrée), de manière à ce qu'il réagisse différemment si un même stimulus lui est appliqué ultérieurement. Le réseau s'*améliore* ainsi car à chaque erreur qu'il fait, il subit une correction qui le fait réagir différemment s'il est confronté à la même situation. Le but étant qu'une fois l'apprentissage terminé, le réseau effectue la tâche pour laquelle il a été concu sans se tromper, c'est-à-dire qu'il fournisse pour chaque stimulus d'entrée la "bonne" sortie, la sortie désirée par l'opérateur.

L'apprentissage n'est bien sûr pas réalisé par l'opérateur par modification des paramètres du réseau *à la main*, mais par un algorithme d'entraînement. De nombreux algorithmes existent à ce jour, que l'on peut séparer en deux types :

- Les apprentissages supervisés : un "professeur" fournit un grand nombre de couples ($\overrightarrow{entrée}$, $\overrightarrow{sortie}_{désiréepourcetteentrée}$), et la correction s'effectue selon l'erreur obtenue pour chaque couple ($\overrightarrow{erreur} = \overrightarrow{sortie}_{obtenue} - \overrightarrow{sortie}_{désirée}$). Si l'apprentissage est efficace, la norme de l'erreur diminue globalement. On l'appelle aussi apprentissage par des exemples.
- Les apprentissages non-supervisés : sans professeur définissant la sortie désirée pour une entrée donnée, donc sans signal d'erreur, le réseau apprend par lui-même à classer des entrées similaires en trouvant des points communs aux stimuli appliqués.

La répétition de la correction en réaction à des stimuli en entrée constitue le processus d'apprentissage. Il faut parfois des milliers d'itérations de l'algorithme pour arriver à un résultat.

Nous verrons un algorithme d'entraînement plus en détail dans la deuxième partie.

5 Applications

Un réseau de neurones est avant tout un outil de traitement de l'information. À force d'apprentissage, on peut le former à des tâches de reconnaissance, classification, approximation, prévision...

On utilise par exemple des réseaux de neurones pour réduire l'écho dans les communications téléphoniques outre-mer (réseau Adaline). Ils peuvent aussi servir pour l'élimination du bruit, les prévisions météorologiques, la compression de données, la modélisation du marché monétaire, le développement d'Intelligence Artificielle...

Grâce à leur capacité d'apprentissage, les réseaux de neurones fournissent de très bons résultats pour des tâches de reconnaissance. Il existe de nombreuses applications dans le traitement du son et de l'image, par exemple dans des logiciels de reconnaissance optique de caractères. C'est parce qu'un tel programme est assez facile à mettre en oeuvre que nous avons choisi de le développer et de le tester dans la troisième partie.

Une application qui a fait parler d'elle est le projet *NetTalk*, réseau de 309 neurones relié en entrée à un scanner et branché en sortie à un haut-parleur, qui a appris à lire un texte à haute voix ! Alors que les outils informatiques traditionnels sont inadaptés à ce genre de problème, les réseaux constituent une nouvelle approche prometteuse. Car une fois entraîné, un réseau est capable de lire des nouveaux mots qu'ils n'a jamais vu auparavant (grâce à sa capacité d'adaptation). Ainsi, l'entraînement de *NetTalk* qui a consisté à lui rabâcher un dictionnaire de 50 000 mots pendant toute une nuit, a donné de bons résultats : au matin il lisait sans faute 95% des mots du dictionnaire d'apprentissage et 75% des nouveaux mots.

Enfin, on peut citer des applications de réseaux faits de vrais neurones "animaux" cette fois-ci. Des chercheurs de l'Université de Reading en Angleterre ont réussi à développer une intelligence artificielle pour robot à l'aide de neurones de rats : mis en culture et connectés par des électrodes au système moteur de la machine, celle-ci apprend à éviter les murs à force de se cogner !

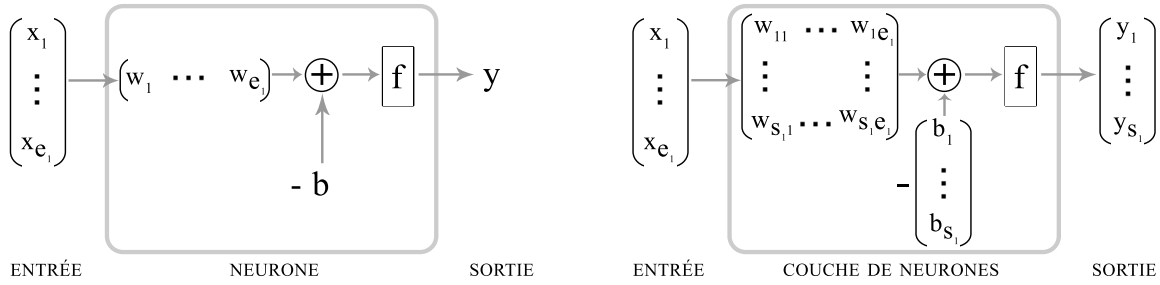
Deuxième partie

Approche théorique

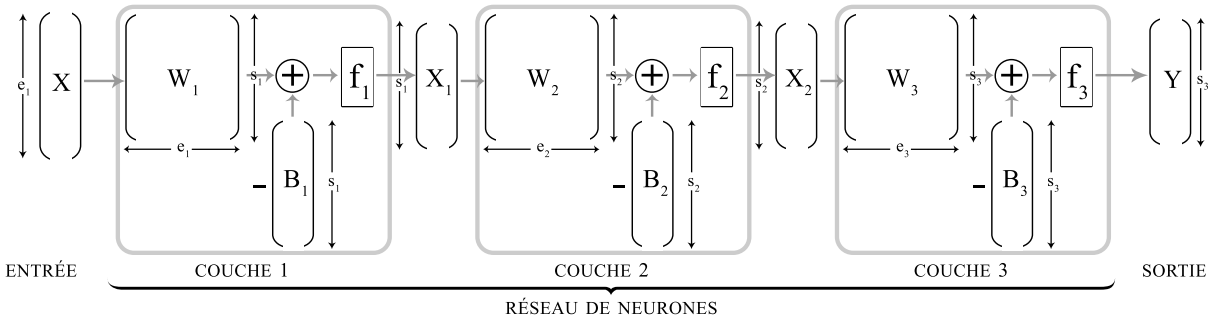
6 Modélisation mathématique du réseau

Comme on l'a entraperçu dans la première partie, l'effet d'un neurone à e_1 entrées peut être modélisé par une forme linéaire de $\mathbb{R}^{e_1}$ dans $\mathbb{R}$ (qui multiplie les composantes de l'entrée par des poids w bien choisis), à laquelle on soustrait un scalaire *biais* et l'on applique enfin une fonction d'activation f (de type seuil, tangente hyperbolique, etc. *a priori* non linéaire) au résultat.

L'action d'une couche de s_1 neurones est donc assimilable à une application linéaire de $\mathbb{R}^{e_1}$ dans $\mathbb{R}^{s_1}$, que l'on représentera ici par une matrice à e_1 colonnes et s_1 lignes : une entrée à e_1 composantes aura pour image un vecteur à s_1 composantes (une pour chaque neurone). On soustrait à cette première sortie un biais $\overrightarrow{B_1}$ qui est cette fois-ci vectoriel, et l'on applique une fonction d'activation f_1 à chaque composante du vecteur obtenu.



Un réseau de neurones tout entier (N couches empilées les unes à la suite des autres) est donc une successions de N matrices de tailles $e_i \times s_i$ $(1 \leq i \leq N)$, auxquelles on associe, à chaque fois, un vecteur biais $\overrightarrow{b_i}$ et une fonction d'activation f_i. Par souci de simplicité, on prendra la même fonction d'activation f_i pour tous les neurones d'une même couche. En revanche, utiliser des fonctions différentes pour les différentes couches peut s'avérer assez intéressant (selon l'usage du réseau).



Exemple de l'action de la couche 1 :

$$\begin{array}{ccccc} X & \longmapsto & W_1 \cdot X - B_1 & \longmapsto & f_1(W_1 \cdot X - B_1) = X_1 \\ \mathbb{R}^{e_1} & \longmapsto & \mathbb{R}^{s_1} & \longmapsto & \mathbb{R}^{s_1} \end{array}$$

Remarque : chaque couche a ses propres paramètres (nombre d'entrées, nombre de sorties, fonction d'activation) ; mais puisque chaque couche prend en entrée les valeurs de sortie de la couche précédente, on doit toujours avoir $e_i = s_{i-1}$.

7 Le perceptron multicouche

Le perceptron multicouche est un type de réseau de neurones parmi les plus utilisés. Son fonctionnement est le suivant : l'information se propage de couche en couche, suivant le modèle décrit plus haut, toujours dans le même sens jusqu'à la sortie. On donne au réseau l'information sous forme d'un vecteur de $\mathbb{R}^{e_1}$ et on récupère l'information traitée sous forme d'un vecteur de $\mathbb{R}^{s_N}$.

Dans certains réseaux autres que le perceptron, plus complexes, l'information ne circule pas en sens unique : il peut y avoir *rétroaction*. Nous ne nous intéresserons qu'au cas du perceptron.

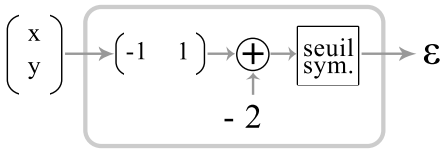
Un exemple très simple : on considère un perceptron monocouche, dont la fonction sera de détecter si un point du plan $\mathbb{R}^2$ est au-dessus ou en dessous de la droite d'équation $y = x + 2$.

On veut :

– en entrée, les coordonnées x et y du point de $\mathbb{R}^2$ ($\Longrightarrow e_1 = 2$)

– en sortie, la valeur $+1$ si le point est au-dessus de la droite, -1 s'il est en dessous ($\Longrightarrow s_1 = 1$).

Utilisons alors la fonction d'activation *seuil symétrique*, qui à tout réel positif associe $+1$, à tout réel négatif -1 et à 0 lui-même ; un biais $b = 2$; et une matrice de poids $W = \begin{pmatrix} -1 & 1 \end{pmatrix}$.



Si notre point est au dessus de la droite, c'est-à-dire si $y > x + 2$, alors $W \cdot X - b = -x + y - 2 > 0$ et la fonction d'activation donnera $\varepsilon = +1$. Si le point sur la droite, $y = x + 2$ et on obtient $\varepsilon = 0$. Si le point est en dessous, $y < x + 2$ et alors $\varepsilon = -1$.

Cet exemple de réseau à un neurone est peu intéressant : on a du déterminer nous-même W et b tels qu'il fonctionne. Le principal intérêt d'un réseau de neurones est sa capacité d'auto-apprentissage.

8 Apprentissage par rétropropagation du gradient

Soit un réseau de type perceptron, dont les N couches ont chacune un nombre d'entrées, de sorties, et une fonction d'activation bien déterminés, mais dont les poids et biais sont initialement inadaptés (les matrices W_i et les vecteurs B_i sont remplis de coefficients aléatoires par exemple). Le réseau est *ignorant* et dépourvu d'expérience : pour une entrée donnée, il ne fournit pas la sortie qu'on souhaiterait obtenir.

(schéma : réseau ignorant, réseau en cours d'apprentissage, réseau apte)

Pour rendre le réseau performant, et donc utilisable sans qu'on n'ait à contrôler si les sorties sont bonnes, on doit l'entraîner en lui imposant un apprentissage de type supervisé. Rappelons le principe de l'apprentissage supervisé : un "professeur" donne un bon nombre de couples $(\overrightarrow{entrée}, \overrightarrow{sortie}_{désirée})$ pour que le réseau s'entraîne à associer à chaque entrée, la sortie correcte.

À chaque stimulus d'information qu'on soumet au réseau, si la sortie obtenue n'est pas la sortie théorique du professeur, on modifie les coefficients de telle sorte que si on recommence, la sortie soit cette fois plus proche de la sortie désirée. La correction selon la règle de moindre carré permet de faire cela.

L'algorithme d'entraînement est la réitération de ce procédé sur de multiples exemples et de nombreuses fois. Il est possible de montrer que sous certaines conditions, les réponses du réseau sont sans fautes en un nombre fini d'itérations. Ce sont les effets de ces conditions que nous avons étudiées dans ce TIPE (*cf* troisième partie).

8.1 Règle du moindre carré pour un perceptron monocouche linéaire

Étudions le cas d'une couche ($N = 1$) ayant une fonction de tranfert linéaire ($f_1 = Id_{\mathbb{R}}$).

Pour commencer, on va s'intéresser à une couche ne contenant qu'un seul neurone ($s_1 = 1$). Notre modélisation du réseau étant linéaire, on passera facilement au cas de plusieurs neurones en appliquant nos résultats à chacun d'entre eux (plus tard).

À chaque stimulus qu'on applique en entrée du réseau (sous la forme d'un vecteur $X \in \mathbb{R}^{e_1}$), on obtient une sortie $Y \in \mathbb{R}$ qui est *a priori* différente de la sortie désirée Y_d. On peut donc définir l'erreur pour chaque stimulus :

$$E = Y - Y_d$$

L'erreur quadratique étant le carré de cette erreur :

$$\varepsilon = E^2$$

La règle de moindre carré consiste à modifier les coefficients (poids et biais) de la couche de manière à réduire l'erreur quadratique.

Pour cela on va "descendre le gradient", c'est-à-dire modifier les coefficients de W et b dans la direction de diminution de l'erreur quadratique.

(schéma avec les minima local et global)

Calcul du vecteur gradient de l'erreur quadratique $\nabla\varepsilon$: en écrivant $W = [w_{ij}]$ et $X = [x_j]$ ($i = 1$, $1 \leq j \leq e_1$)

$$\forall 1 \leq j \leq e_1, \quad [\nabla\varepsilon]_j = \frac{\partial\varepsilon}{\partial w_{1j}} = \frac{\partial E^2}{\partial w_{1j}} = 2E\frac{\partial E}{\partial w_{1j}} = 2E\frac{\partial(Y - Y_d)}{\partial w_{1j}} = 2E\frac{\partial((W\cdot X - b) - Y_d)}{\partial w_{1j}}$$
$$= 2E\frac{\partial(W\cdot X)}{\partial w_{1j}} = 2E\frac{\partial\sum_{k=1}^{e_1} w_{1k}x_k}{\partial w_{1j}} = 2x_jE$$

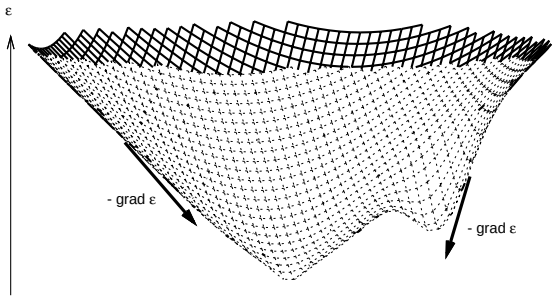
Si bien que $\nabla\varepsilon = 2EX$ et

$$\frac{\partial\varepsilon}{\partial b} = 2E\frac{\partial((W\cdot X - b) - Y_d)}{\partial b} = -2E$$

Il devient donc très facile d'obtenir le gradient de l'erreur : il suffit de multiplier l'erreur par le stimulus d'entrée.

Pour diminuer l'erreur, la correction consiste à retrancher aux coefficients les gradients obtenus multipliés par une quantité positive η appelée *vitesse d'apprentissage*. On affecte ces nouvelles valeurs aux coefficients de la couche :

$$\begin{aligned} W &\quad\hookleftarrow\quad W - \eta\,\nabla\varepsilon = W - 2\eta E\,{}^tX \\ b &\quad\hookleftarrow\quad b - \eta\frac{\partial\varepsilon}{\partial b} = b + 2\eta E \end{aligned}$$



Ci contre, la représentation de l'erreur quadratique en fonction des 2 poids d'un perceptron 2×1. Cette méthode consiste en fait à *descendre le gradient* pour s'approcher d'un minimum de l'erreur ε.

On peut faire l'analogie avec l'énergie potentielle en physique : les coordonnées d'une particule dans un champ "descendent" le potentiel.

Dans le cas plus général où on a un nombre s_1 de neurones dans notre couche, ce qui impose un biais $B \in \mathbb{R}^{s_1}$ et une matrice de poids $W \in \mathcal{M}_{s_1,e_1}(\mathbb{R})$, l'erreur quadratique est un réel positif qui s'exprime comme le carré de la norme du vecteur erreur $E = Y - Y_d$, soit dans $\mathbb{R}^{s_1}$:

$$\varepsilon = {}^tE\cdot E$$

Pour chaque neurone i on modifie la composante du vecteur B et la ligne de la matrice W correspondantes de la même manière :

$$\begin{aligned} \forall 1 \le i \le s_1, \qquad w_{ij} &\quad\hookleftarrow\quad w_{ij} - 2\eta E_i x_j \qquad (1 \le j \le e_1) \\ \forall 1 \le i \le s_1, \qquad b_i &\quad\hookleftarrow\quad b_i + 2\eta E_i \end{aligned}$$

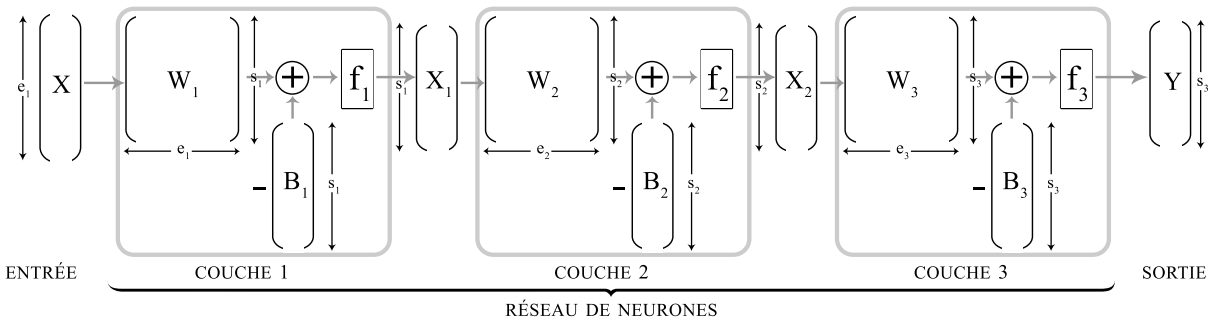
Soit sous forme matricielle :

$$\begin{aligned} W &\quad\hookleftarrow\quad W - 2\eta E\,{}^tX \\ B &\quad\hookleftarrow\quad B + 2\eta E \end{aligned}$$

8.2 Rétropropagation pour un perceptron multicouche

Dans le cas d'un perceptron quelconque à N couches, on observe deux différences par rapport à l'étude précédente :

- les fonctions d'activation de chaque couche ne sont pas forcément linéaires,
- les données du "professeur" qui permettent la correction ne concernent que la sortie de la dernière couche (couche *de sortie*).



Il va alors falloir trouver un moyen de corriger les couches une par une en partant de la dernière et en remontant pas à pas jusqu'à la première. C'est en quoi consiste la méthode de rétropropagation du gradient.

L'algorithme de correction est toujours basé sur la descente du gradient, par exemple sur la couche de sortie N :

$$\begin{aligned} W_N &\longmapsto W_N - \eta \nabla \varepsilon_N \\ B_N &\longmapsto B_N - \eta \nabla' \varepsilon_N \end{aligned}$$

Mais ici les dérivées partielles ont changé car la fonction d'activation n'est pas toujours linéaire. Cependant, si les fonctions d'activation sont dérivables (ce qui n'est pas le cas de la fonction seuil : sa dérivée est nulle partout sauf en 0 où elle n'est pas définie), et en notant f'_n la dérivée de f_n, on peut montrer qu'il faut corriger les poids et biais de la sorte :

$$\begin{aligned} \forall 1 \leq n \leq N, \qquad W_n &\longmapsto W_n - \eta S_n \, {}^t X_n \\ B_n &\longmapsto B_n + \eta S_n \end{aligned}$$

Où S_n est un vecteur qui représente la sensibilité de l'erreur au changement de $A_n = [a_i^{(n)}] = W_n \cdot X_{n-1} - B_n$ dans la couche n et se calcule par rétro-récurrence :

$$S_N = 2 \begin{pmatrix} f'_N(a_1^{(N)}) & 0 & \cdots & 0 \\ 0 & f'_N(a_2^{(N)}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f'_N(a_{s_N}^{(N)}) \end{pmatrix} \cdot (Y - Y_d)$$

$$\forall 1 \leq n \leq N-1, \quad S_n = \begin{pmatrix} f'_n(a_1^{(n)}) & 0 & \cdots & 0 \\ 0 & f'_n(a_2^{(n)}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f'_n(a_{s_n}^{(n)}) \end{pmatrix} \cdot {}^t W_{n+1} \cdot S_{n+1}$$

L'algorithme d'apprentissage par rétropropagation du gradient peut être résumé par les étapes :

- on propage le vecteur d'entrée X à travers les couches, ce qui donne $X_1, \ldots, X_{N-1}, Y$,
- on calcule la dernière sensibilité S_N ce qui nous permet de corriger les poids et biais de la dernière couche ; puis on calcule S_{N-1} pour corriger l'avant dernière, ..., et ce jusqu'à la première couche par rétropropagation.

Étapes que l'on répète pour chaque couple entrée/sortie désirée (X, Y_d) donné par le professeur. Chaque itération (correspodant à une certaine entrée) modifie les coefficents du réseau d'une manière qui est censée diminuer l'erreur quadratique pour l'entrée donnée.

Pour arriver à un résultat, il convient d'entraîner notre réseau plusieurs fois pour chaque couple entrée/sortie désirée, pour arriver à une erreur quadratique moyenne la plus basse possible.

Remarque : il existe d'autres méthodes de correction de l'erreur, telle la méthode de Newton ou celle du gradient conjugué qui consiste à chercher le minimum de l'erreur selon des droites dans l'espace des coefficients.

Troisième partie

Étude expérimentale, mesures, optimisation

9 Élaboration d'un programme "testeur de réseaux" simple

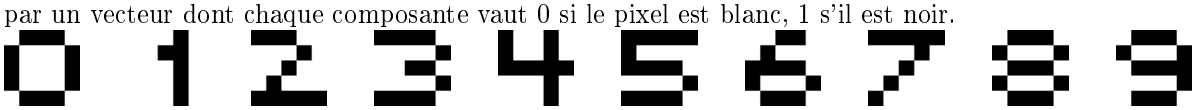
Pour faire l'étude expérimentale des réseaux neuronaux et déterminer les conditions pratiques qui limitent leurs performances, j'ai mis au point un programme informatique qui simule un réseau de neurones accompagné d'un algorithme d'apprentissage.

J'ai d'abord codé ce programme en PHP, langage servant usuellement à générer des pages web dynamiques et qui était intéressant pour la simplicité de son interface avec l'utilisateur : il s'agit d'hypertexte HTML. Il a donc l'apparence d'une page web, et vous pouvez tester ce réseau à l'adresse internet `http://www.k-netweb.net/projects/tipe/reseau/`.

J'ai mis le code source à disposition à l'adresse `http://www.k-netweb.net/projects/tipe/reseau/codesource.zip`.

Pour des raisons de performances, j'ai du ultérieurement recoder mon programme avec un langage plus puissant : le JAVA (chaque entraînement nécessite de répéter l'algorithme des centaines de fois).

L'étude des performances se fera sur un exemple : un réseau de neurones dont la fonction est de reconnaître 10 images de $5\times5 = 25$ pixels représentant les chiffres de zéro à neuf. Chaque image est *codée* par un vecteur dont chaque composante vaut 0 si le pixel est blanc, 1 s'il est noir.



Par exemple, pour l'image du chiffre zéro le vecteur est (01110100011000110001011 10).

Le réseau prend alors en entrée un vecteur de $\mathbb{R}^{25}$ et donne en sortie un vecteur de $\mathbb{R}^{10}$. La couche de sortie comporte donc 10 neurones, qui donnent les "affinités" de l'image avec les différents chiffres. Idéalement, le quatrième neurone (par exemple) de la couche de sortie d'un réseau correctement entraîné doit donner 1 si l'image représente le chiffre quatre, 0 sinon.

$$(1001010010111110001000010) \quad \longmapsto \quad \underset{\text{vecteur caractérisant quatre}}{\begin{pmatrix}0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\end{pmatrix}}$$

En pratique, on n'obtiendra jamais un tel vecteur avec uniquement des 0 et un 1. Dans notre étude, on considérera que le réseau est *opérationnel* lorsqu'il associe à chacune des 10 images un vecteur de $\mathbb{R}^{10}$ tel que sa plus grande composante est celle correspondant au chiffre. Par exemple, une sortie $\begin{pmatrix}0.01 & 0 & 0.2 & 0 & 0.98 & 0.3 & 0.1 & 0.02 & 0 & 0.07\end{pmatrix}$ pour l'image du chiffre 4 nous conviendra.

Remarque : on aurait pu décider d'avoir une sortie scalaire, qui donnerait idéalement 0 pour la première image, 1 pour la deuxième, ...et 9 pour la dixième. Nous avons testé cette variante mais les résultats sont médiocres. On a donc préféré "séparer" en 10 affinités distinctes.

Pour chacune de nos expériences, on crée un nouveau réseau de type perceptron (dont les coefficients sont remplis par des nombres pris au hasard entre -2 et 2), puis on l'entraîne avec l'algorithme de rétropropagation du gradient.

Dans la suite nous allons voir quelles sont les meilleures conditions pour obtenir rapidement un réseau opérationnel. Pour cela on évaluera la durée de l'apprentissage nécessaire en fonction de différents paramètres que l'on fera varier.

Le programme informatique permet de créer et d'entraîner un réseau en choisissant :
- le nombre de couches et la dimension de celles-ci (la dernière contient toujours 10 neurones),
- les fonctions d'activation des couches.

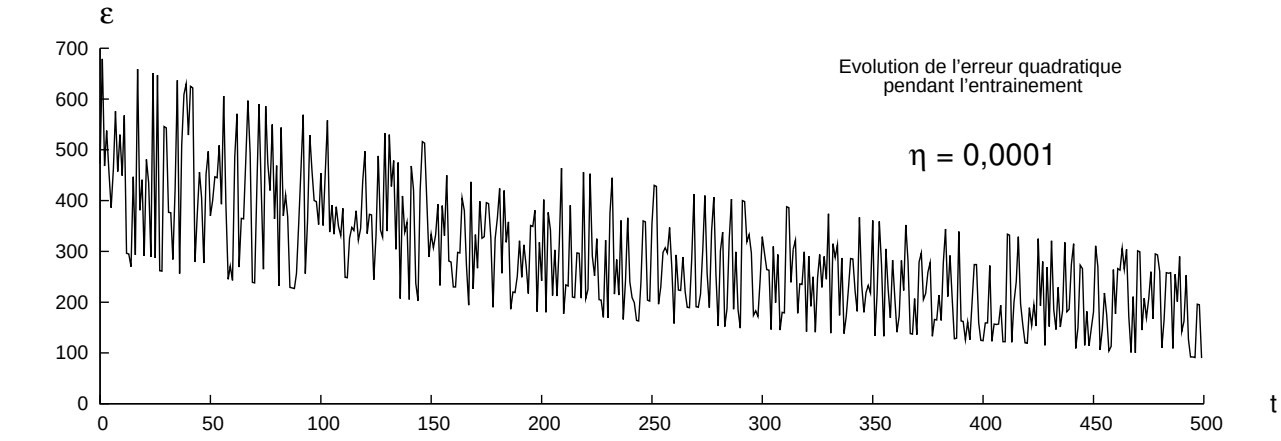
De même, pour l'apprentissage par rétropropagation du gradient on peut :
- entraîner le réseau pendant $T = 500$ itérations de l'algorithme sur des exemples présentés dans un ordre aléatoire ou périodique,
- choisir la valeur de la vitesse d'apprentissage η.

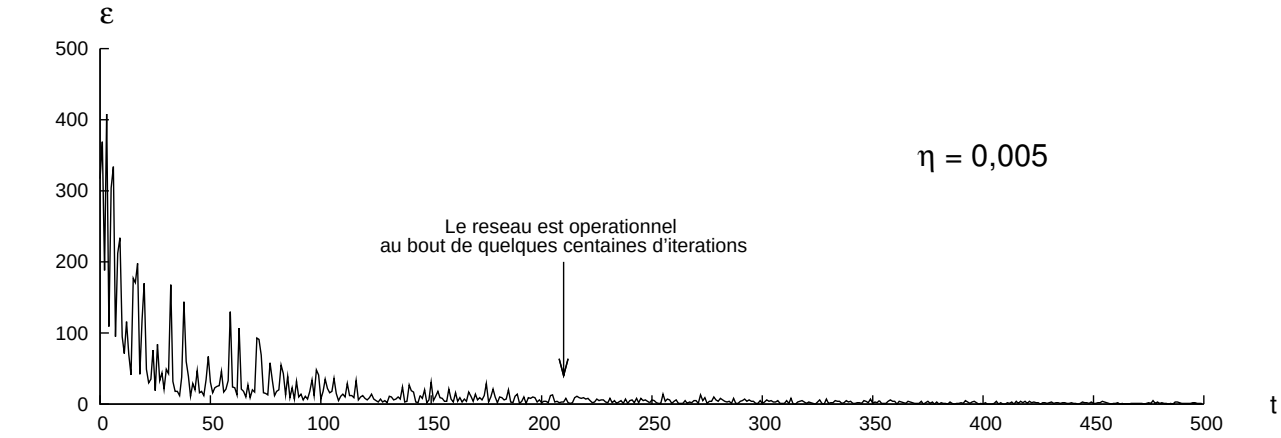
À chaque itération t de l'algorithme d'entraînement, qui est faite sur un des dix vecteurs de test (par exemple le chiffre 4), on calcule l'erreur et on l'affiche sur un graphe.

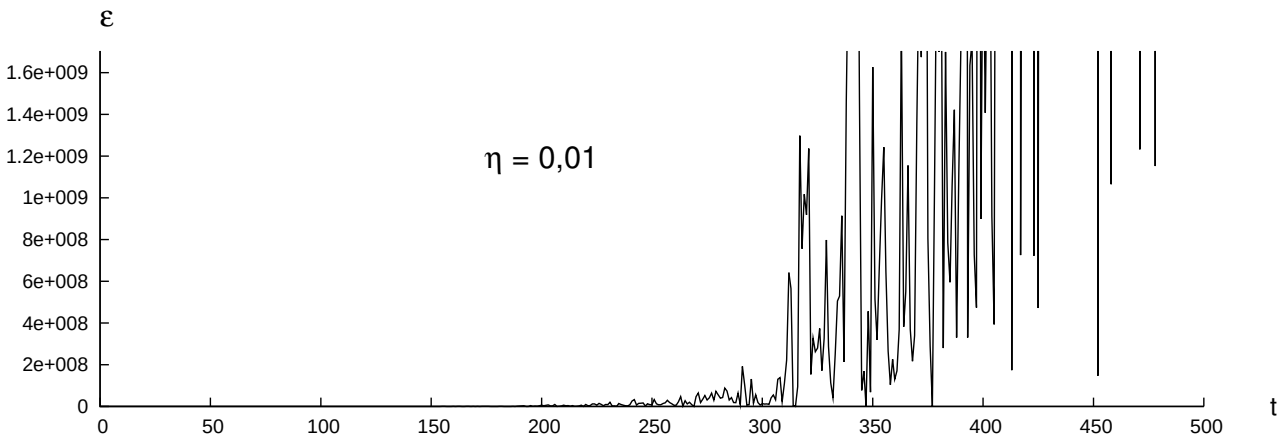
10 Influence de la vitesse d'apprentissage

Nous travaillons ici avec des réseaux de type perceptron à une seule couche, dont la fonction d'activation est linéaire ($f = Id_{\mathbb{R}}$).

On a donc appliqué à des réseaux "ignorants" un apprentissage de 500 itérations de l'algorithme, en testant différentes valeurs de la vitesse d'apprentissage η.







Mon groupe et moi avons remarqué que l'erreur ne convergeait vers un minimum que si la valeur de la vitesse d'apprentissage était inférieure à une certaine limite η^{max}. Au-dessus, les coefficients du réseau divergent vers l'infini, en-dessous ils convergent vers une certaine valeur.

Si l'on entraîne le réseau pour reconnaître uniquement l'image 8 : $X = (0111010001011101000101110) \in \mathbb{R}^{25}$ dont la norme au carré vaut $\|X\|^2 = 13$, la vitesse limite vaut $\eta^{max} \simeq 0.071 \simeq \frac{1}{\|X\|^2+1}$. Pour les autres vecteurs, on obtient la même règle : convergence des coefficients du réseau pour $\eta < \frac{1}{\|X\|^2+1}$, divergence pour $\eta > \frac{1}{\|X\|^2+1}$.

On a constaté qu'expérimentalement, cette loi est toujours vraie pour des réseaux de tailles différentes (par exemple avec une entrée dans $\mathbb{R}^{15}$ et une sortie dans $\mathbb{R}^{13}$).

Ne trouvant aucune mention de ce phénomène dans les documents que nous consultions pour ce TIPE, nous avons voulu démontrer cette loi.

Considérons donc un perceptron monocouche linéaire, que l'on souhaite entraîner pour reconnaître un certain vecteur d'entrée X, c'est-à-dire lui associer la sortie désirée Y_d. Si l'on appelle $W^{(t)}$ la matrice des poids et $B^{(t)}$ le vecteur des biais du réseau à l'itération t, on veut en fait que la sortie effective $Y^{(t)} = W^{(t)}X - B^{(t)}$ s'approche le plus possible de Y_d.

La correction par *descente du gradient* impose une modification des coefficients :

$$W^{(t+1)} = W^{(t)} - 2\eta(Y^{(t)} - Y_d)\,{}^tX$$
$$B^{(t+1)} = B^{(t)} + 2\eta(Y^{(t)} - Y_d)$$

Si l'on multiplie la première ligne par X et que l'on y soustrait la deuxième :

$$\underbrace{W^{(t+1)}X - B^{(t+1)}}_{Y^{(t+1)}} = \underbrace{W^{(t)}X - B^{(t)}}_{Y^{(t)}} - 2\eta(Y^{(t)} - Y_d)(\|X\|^2 + 1)$$

Donc :

$$Y^{(t+1)} - Y_d = \alpha(Y^{(t)} - Y_d)$$
$$\text{avec} \quad \alpha = 1 - 2\eta(\|X\|^2 + 1)$$

Et l'on obtient finalement pour cette suite arithmético-géométrique :

$$Y^{(t)} = \alpha^t \cdot (Y^{(0)} - Y_d) + Y_d$$

Conclusion : la sortie $Y^{(t)}$ converge si et seulement si $|\alpha| < 1$, autrement dit si $0 < \eta < \frac{1}{\|X\|^2+1}$.

Il apparaît finalement qu'il existe une valeur limite de la vitesse d'apprentissage :
- **au-dessus de celle-ci, les coefficients divergent à coup sûr,**
- **en dessous de cette vitesse, le réseau converge plus ou moins lentement vers un état qui est le bon.**

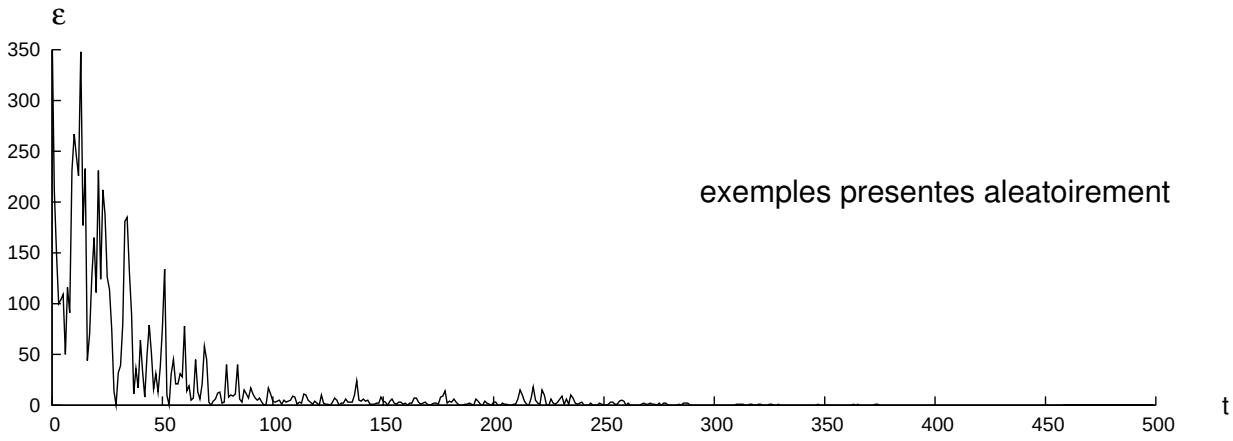
Remarque : dans le cas d'une fonction d'activation non linéaire, le résultat précédent n'est pas directement appliquable. On remarque même qu'une trop faible valeur de η fait converger le réseau vers une *cuvette* de l'erreur (un minimum local qui n'est pas le minimum global). On comprend qu'une vitesse trop petite ne permet pas de faire le "saut" pour sortir de la cuvette de l'éventuel minimum local.

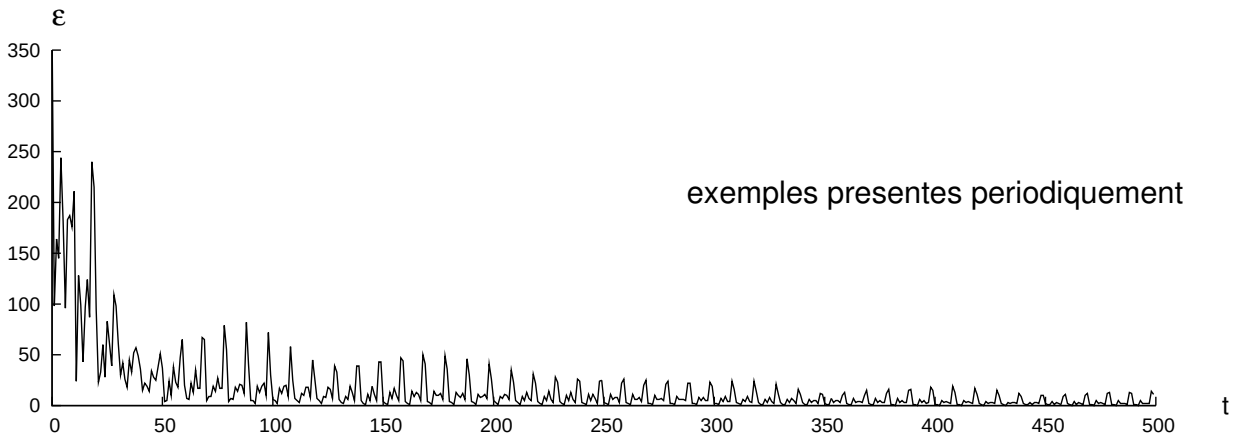
Pour empêcher ce phénomène, on peut introduire un vitesse d'apprentissage variable. C'est le principe du momentum : on donne une *inertie* à la vitesse de descente du gradient de l'erreur, c'est-à-dire que si la modification des poids était grande à l'itération t, elle restera assez grande à $t+1$. On peut s'imaginer une bille qui, du fait de son inertie, garde de l'élan pour sortir de la cuvette.

11 Influence de l'ordre de présentation des exemples

On considère toujours un perceptron linéaire dont la tâche est de classer nos chiffres de zéro à neuf, auquel on applique comme d'habitude une *correction* pour chaque exemple présenté, de manière à réduire l'erreur pour chacun de ces exemples.

Pour une même valeur de la vitesse d'apprentissage η, on a évalué la diminution de l'erreur en fonction de l'ordre de présentation de ces vecteurs d'entrée : soit de manière périodique (0 1 2 3 . . . 8 9 0 1 2 . . .), soit de manière aléatoire.





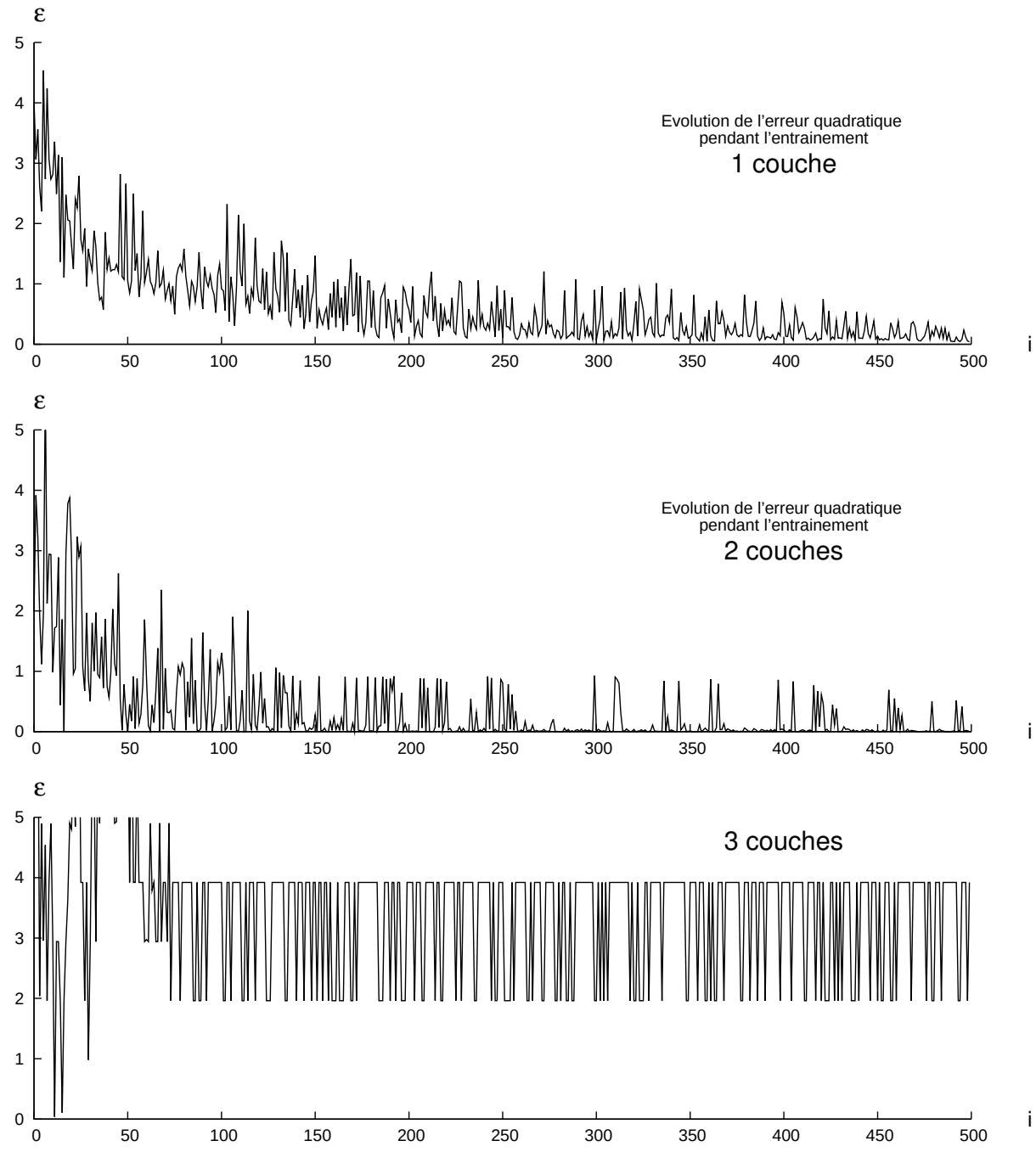
Il apparaît plus fructueux de présenter les exemples dans un ordre aléatoire plutôt que dans un ordre donné, pour deux raisons :

- l'erreur diminue en général plus rapidement,
- cela évite de rester bloqué dans un minimum local de l'erreur au cours de l'entraînement. En effet, chaque ordre de présentation engendre une trajectoire de descente de l'erreur différente dans l'espace des paramètres du réseau, et certaines trajectoires peuvent nous amener à des minima locaux au lieu du minimum global. Un ordre aléatoire réduit ce risque.

12 Influence du nombre de couches

La première couche peut servir à extraire certaines caractéristiques des données brutes qu'on lui soumet, c'est-à-dire donner des sorties similaires pour des entrées ayant certains points communs. Les couches suivantes peuvent comparer ces caractéristiques pour les classer correctement.

Le nombre de couche que l'on donne à un réseau dépend donc de la complexité du problème que l'on veut traiter.



Les expériences faites montrent qu'il est beaucoup plus dur d'entraîner un réseau multicouche qu'un monocouche. De plus, il vaut mieux donner des fonctions d'activation linéaires pour les couches cachées (pour éviter que les sorties intermédiaires soient comprises entre -1 et 1).

Quatrième partie

Bibliographie

Frédéric Bourg, « Ordinateurs, enfin doués pour la lecture »
Science & Vie, n°1082 de nov. 2007, p. 78

David Delaunay, « Le perceptron »
Tangente Sup, n°44-45, p. 24

Marc Parizeau, « Réseaux de neurones »
thèse à l'université Laval, 2004

François Denis et Rémi Gilleron, « Apprentissage à partir d'exemples », chap. 3
http ://www.grappa.univ-lille3.fr/polys/apprentissage/

Wikipedia, « Le neurone biologique »
http ://fr.wikipedia.org/wiki/Neurone