Les réseaux de neurones

Jerome SCHWINDLING
IRFU, CEA Saclay

1 Introduction

Les réseaux de neurones artificiels (ANN en anglais) sont utilisés en Physique des Particules depuis la fin des années 80 (voir par exemple [1]) mais, pendant une dizaine d'années, ils ont souvent été pris pour des boîtes noires et considérés avec méfiance. Depuis une dizaine d'années, les réseaux de neurones sont, dans notre domaine, beaucoup mieux compris, mieux utilisés, et de ce fait beaucoup mieux acceptés.

Le terme "réseau de neurones artificiels" est utilisé pour différents types de "réseaux" et d'algorithmes. Ce cours ne décrit qu'un seul type : le perceptron multi-couches (MLP = Multi-Layer Perceptron). Ce type de réseaux de neurones est à la fois très simple et très utile et, peut-être pour ces raisons, le plus utilisé.

Il existe deux visions très différentes mais complémentaires du perceptron multi-couches : la vision neurobiologique et la vision mathématique.

La vision neurobiologique est historiquement à l'origine des réseaux de neurones artificiels. Bien que moins rigoureuse que la vision mathématique, elle est souvent la seule décrite dans les livres ou les articles. Elle est parfois utile comme image du fonctionnement d'un réseau de neurones artificiel.

La vision mathématique est beaucoup plus rigoureuse, mais cependant simple à comprendre. Elle donne une meilleure explication du fonctionnement des perceptrons multi-couches et de l'apprentissage.

Ce cours présentera successivement ces deux visions. Il se terminera par quelques conseils pour l'utilisation des perceptrons multi-couches.

2 La vision neurobiologique

2.1 Le perceptron multi-couches

L'idée d'imiter le fonctionnement du cerveau en utilisant des unités de calcul simples (les neurones) reliées de façon plus ou moins complexe remonte aux années 40 (par exemple [2]).

Il existe une infinité de façon d'organiser et de connecter de tels neurones pour réaliser un réseau. Le perceptron multi-couches est un modèle simple dans lequel (voir figure 1) :

- les neurones sont organisés en couches, avec une couche d'entrée (où le nombre de neurones est égal au nombre de variables du problème considéré), une couche de sortie (qui peut avoir un ou plusieurs neurones), et zéro, une ou plusieurs couches intermédiaires, souvent appelées *couches cachées* [1] ;
- l'information ne se propage que dans un seul sens, de la couche d'entrée vers la couche de sortie. Ce type de réseau est appelé réseau *feed-forward*.
- chaque neurone reçoit des informations de tous les neurones de la couche précédente, calcule une combinaison linéaire $x_j = w_{0j} + \Sigma_i w_{ij} y_i$ de ces informations, et donne comme résultat $y_j = A(x_j)$ où $A(x)$, appelée *fonction d'activation*, est de la forme $A(x) = 1/(1+e^{-x})$ (fonction *sigmoïde*) [2]. La fonction d'activation des neurones de sortie peut être l'identité.

[1] Un réseau avec I neurones d'entrée, J_1,...J_n neurones dans n couches cachées et K neurones dans la couche de sortie sera noté $I - J_1 - ... - J_n - K$ dans la suite.

[2] D'autres fonctions de type "seuil" sont possibles, comme on le verra plus bas.

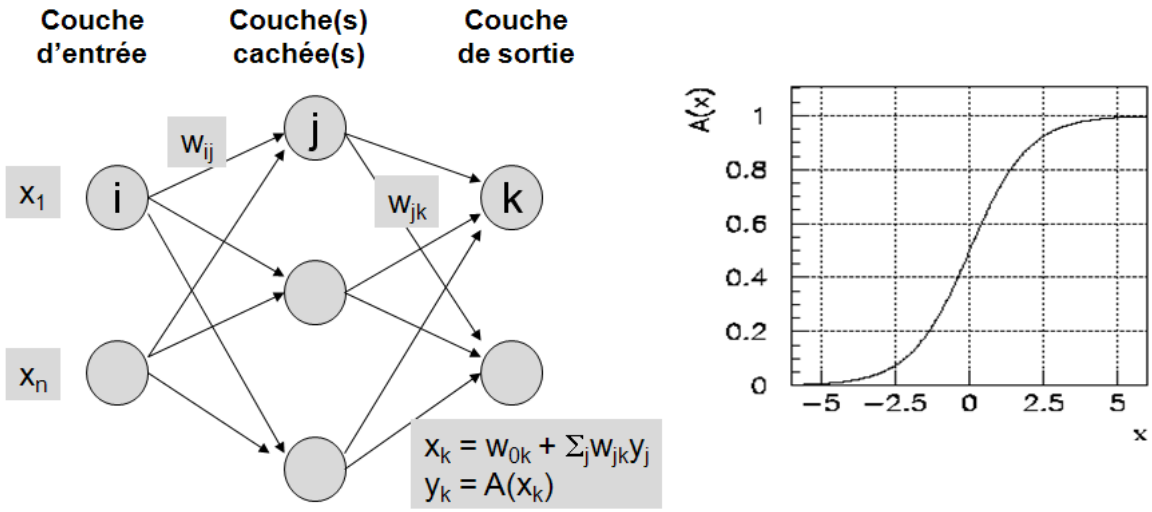

Fig. 1: Le perceptron multi-couches

Les paramètres w_{ij} sont appelés les *poids*. Le perceptron multi-couches réalise donc une fonction des entrées $\vec{x}$ et des poids $\vec{w}$: $\vec{y} = mlp(\vec{x}, \vec{w})$. Les poids sont déterminés durant une phase d'apprentissage (ou phase d'entraînement).

2.2 L'apprentissage

Les poids $\vec{w}$ sont déterminés de manière itérative à partir d'un *lot d'apprentissage* de la façon suivante :

- boucler de façon aléatoire sur tous les exemples du lot d'apprentissage. Un passage sur tous les exemples sera appelé dans la suite une *époque*.
- comparer pour chaque exemple la sortie du réseau et la sortie souhaitée (*apprentissage supervisé*). La réponse souhaitée est souvent 0 ou 1 pour un problème de classification de données.
- modifier les poids après chaque exemple, selon la méthode qui sera donnée plus bas
- recommencer au premier point "plusieurs fois"

Après la phase d'apprentissage, le réseau doit être capable de *généraliser*, c'est-à-dire de donner une réponse correcte pour un exemple nouveau.

La méthode pour modifier les poids du réseau, après chaque exemple, est souvent appelée *méthode de rétropropagation des erreurs* [3]. Cette appellation est en réalité un abus de langage comme nous le verrons plus tard [3]. Quoi qu'il en soit, la recette est la suivante :

- pour chaque exemple, calculer la réponse y_k [4] et la comparer à la réponse souhaitée d_k. Calculer $\delta_k = A'(x_k)(y_k - d_k)$, où $A'(x_k)$ est la valeur de la dérivée de la fonction d'activation en x_k
- pour chaque neurone j de la couche précédente, calculer $\delta_j = A'(x_j)\Sigma_k \delta_k w_{jk}$. C'est cette étape qui peut s'appeler *rétropropagation de l'erreur*.
- modifier chaque poids de la façon suivante : $w_{ij} = w_{ij} - \eta \delta_j y_i$

Le paramètre η est appelé *paramètre d'apprentissage*. Il est plus petit que 1 et doit être optimisé pour chaque problème.

L'évolution de la performance du réseau peut être suivie en calculant, après chaque époque, l'erreur $\Sigma_p (mlp(\vec{x_p}, \vec{w}) - d_p)^2$. Cette erreur peut être calculée à la fois sur le lot d'exemples utilisés pour l'apprentissage et sur un lot indépendant, appelé lot de test. Une évolution typique de l'erreur est représentée sur la figure 5. On constate que :

- l'erreur sur le lot d'apprentissage décroît, comme on peut l'espérer. En réalité, à cause du mélange aléatoire des exemples, l'erreur peut fluctuer autour d'une courbe décroissante, voire ne pas décroître si le paramètre d'apprentissage η est trop grand ;

[3] Si vous ne deviez retenir qu'une chose de ce cours, c'est pourquoi cette appellation est un abus de langage.

[4] l'indice k boucle sur les neurones de sortie, comme dans la figure 1. Très souvent il n'y a qu'un neurone de sortie, et cet indice est alors inutile.

– l'erreur sur le lot d'apprentissage ne décroît que très lentement entre 20 et 70 époques, puis décroît subitement entre 80 et 90 époques. Ce comportement, très fréquent, indique qu'on a fini par trouver une meilleure configuration des poids. Malheuresement il est impossible de prévoir quand une meilleure configuration sera trouvée, ou de savoir s'il n'existe pas de configuration encore meilleure ;
– l'erreur mesurée sur le lot de test a un comportement différent. Elle décroît également en moyenne jusqu'à 80-90 époques, puis se met à croître de façon continue au-dela de 120 époques. Ce comportement est symptomatique du *sur-apprentissage*, qui indique que le réseau a appris des caractéristiques particulières des exemples présentés : il commet alors une erreur très petite sur le lot d'apprentissage, mais est incapable de généraliser au lot de test.

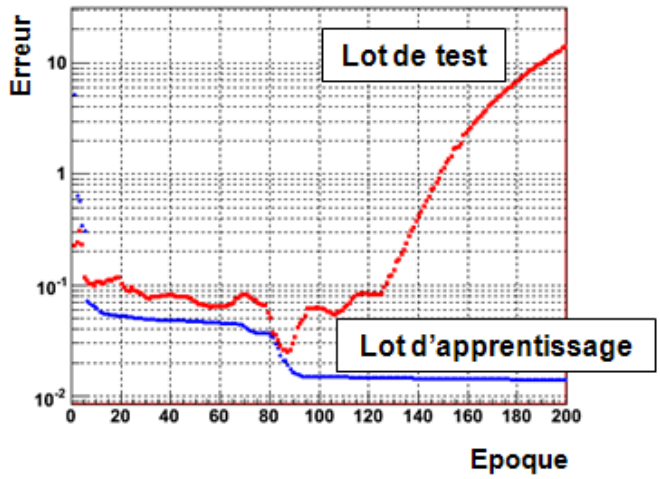

Fig. 2: Evolution de l'erreur lors de la phase d'apprentissage. L'erreur est calculée après chaque époque sur le lot d'apprentissage et sur un lot de test indépendant.

Le sur-apprentissage est illustré sur la figure 3. Il se produit en général lorsque le nombre d'exemples n'est pas assez grand par rapport au nombre de poids du réseau, mais il peut y avoir sur-apprentissage même si le nombre d'exemples est supérieur au nombre de poids et, inversement, il peut ne pas y avoir sur-apprentissage même si le nombre de poids est supérieur au nombre d'exemples. Suivre l'erreur sur un lot de tes indépendant est donc la seule façon d'éviter ce problème.

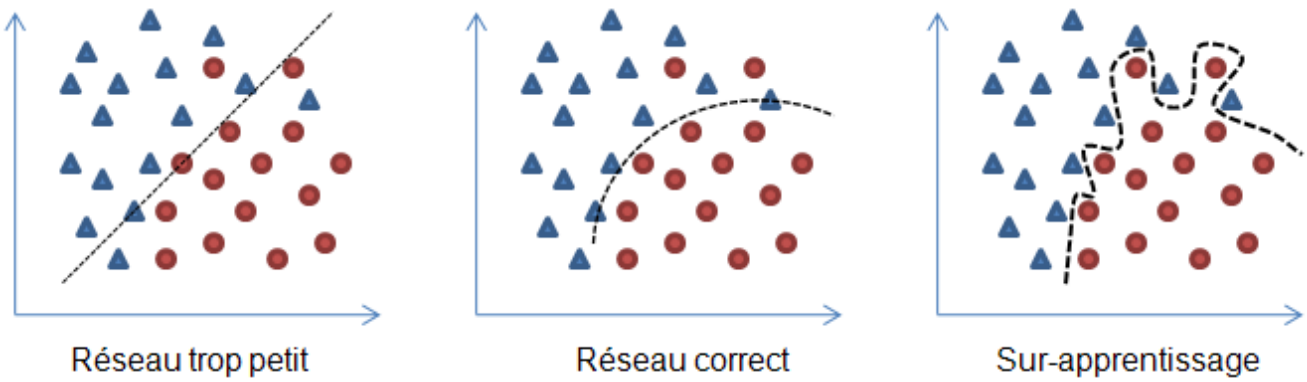

Fig. 3: Illustration du problème du sur-apprentissage.

2.3 Utilisation du perceptron multi-couche pour la classification de données

Traditionnellement, le perceptron multi-couches a été, et est toujours, utilisé pour la classification de données, comme la séparation d'un signal du bruit de fond ou la reconnaissance de caractères manuscrits [4]. Cette utilisation repose sur les propriétés suivantes.

Un perceptron sans couche cachée, correctement entrainé, réalise une séparation linéaire discrimante : la surface $w_0 + \Sigma_i w_i x_i = 0$ est un hyperplan dans l'espace des paramètres qui sépare cet espace en une zone de "signal" et une zone de "bruit de fond". Un tel réseau ne peut donc pas résoudre exactement des problèmes non linéairement séparable, tels que la séparation du signal et du bruit de fond dans la figure 4

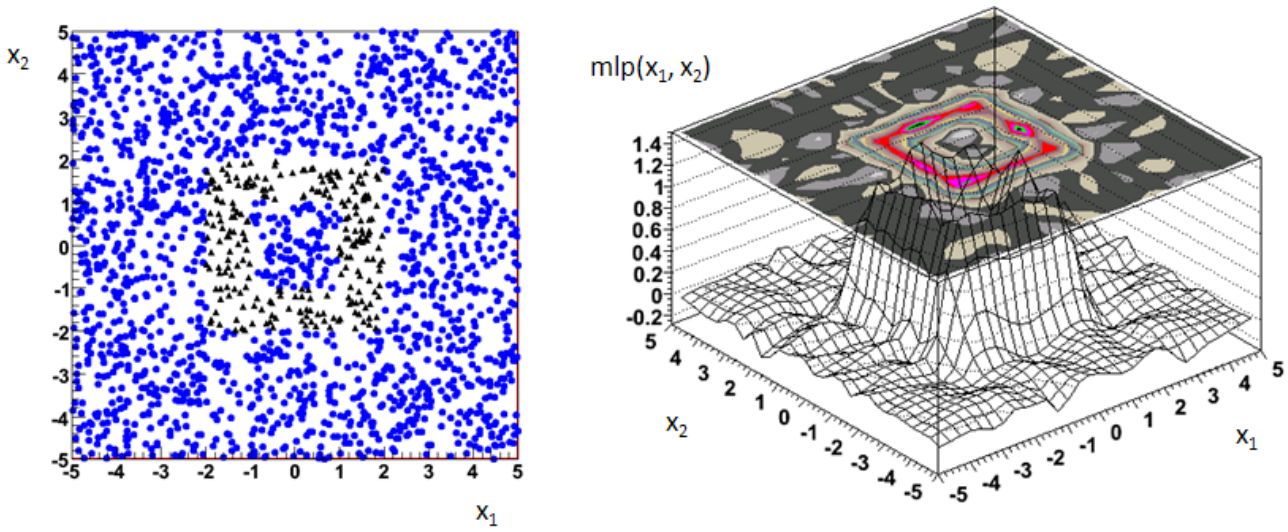

Fig. 4: Résolution d'un problème non linéairement séparable : à gauche le signal (triangles) à une forme de "carré creux" et est noyé dans la bruit de fond. A droite : la sortie du réseau de neurones après apprentissage.

Pour résoudre de tels problèmes, une couche cachée (au moins) est nécessaire : chaque neurone de cette couche sépare l'espace des variables par un hyperplan. Le ou les neurone(s) de la couche suivante réalise une fonction logique quelconque de ces demi-espaces. Un exemple est donnée sur la figure 4 où un réseau 2-50-1 a été entrainé à séparer un signal qui a la forme d'un "carré creux" noyé dans du bruit de fond. Après 1000 époques d'apprentissage sur 2000 exemples, le taux d'erreur (nombre d'exemples mal classés par une coupure à 0.5 sur la sortie du réseau) vaut environ 1%.

3 La vision mathématique

3.1 Le perceptron multi-couches

La fonction $mlp(\vec{x}, \vec{w})$ réalisée par un perceptron à une couche cachée et un neurone de sortie linéaire est une combinaison linéaire de fonctions sigmoïdes. L'utilisation d'un tel réseau repose entièrement sur le théorème suivant [5] :

Toute fonction continue de $\mathbb{R}^n \to \mathbb{R}$ *peut tre approximée, à une précision quelconque, par une combinaison linéaire de sigmoïdes.*

Ce théorème peut être démontré facilement, et appelle les remarques suivantes :

- la généralisation à une fonction de $\mathbb{R}^n \to \mathbb{R}^m$ est immédiate, puisqu'il suffit de superposer m perceptrons.
- ce théorème est vrai, plus généralement, avec une combinaison linéaire de fonctions croissantes bornées, ce qui explique que la sigmoïde est parfois remplacée par *atan* ou *tanh*.
- le théorème ne précise pas combien de neurones sont nécessaires pour obtenir une précision donnée, mais indique qu'une couche cachée est toujours suffisante.

3.2 Utilisation du perceptron multi-couche pour l'ajustement de fonctions

Si l'on dispose d'un logiciel de perceptrons multi-couches qui permet de choisir un neurone de sortie linéaire, on peut vérifier que le théorème précédent fonctionne sur des fonctions simples. Par exemple, essayons d'ajuster la fonction $f(x) = \sin(x)/x$, dans le domaine $[0 - 20]$ (représentée sur la figure 5, à gauche). Comme il s'agit d'une fonction de $\mathbb{R} \rightarrow \mathbb{R}$, nous utiliserons un réseau à un neurone d'entrée, un neurone de sortie, et nous choisissons 6 neurones dans la couche cachée [5]. Le lot d'apprentissage est constitué de 100 points équidistants dans le domaine 0 - 20, et le lot de test est constitué de 100 autres points sur la fonction.

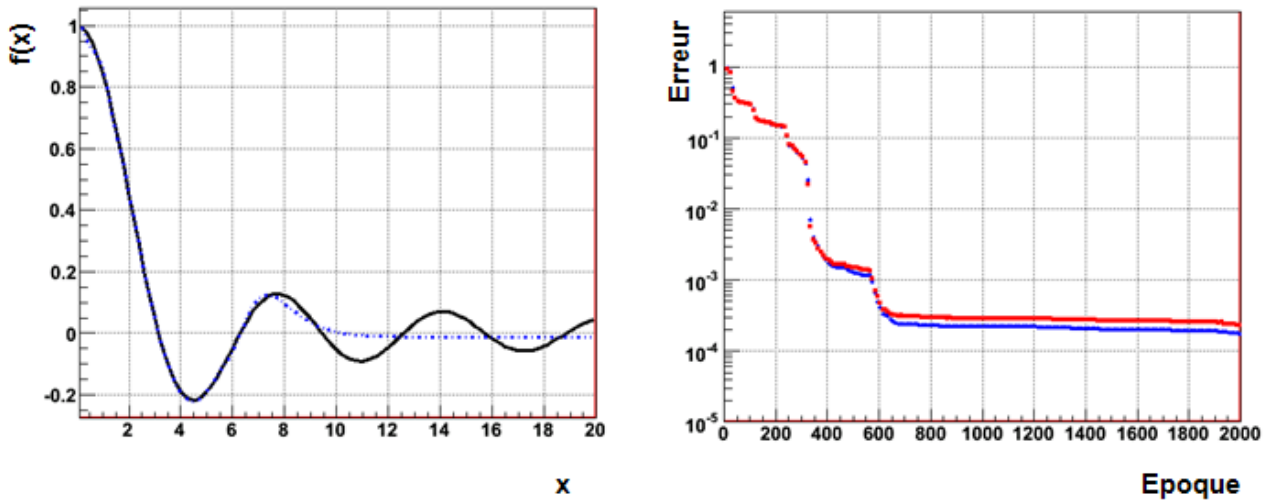

Fig. 5: Ajustement de la fonction $sin(x)/x$. La figure de gauche montre cette fonction (en noir) et la fonction $mlp(x)$ après 200 époques d'apprentissage. La figure de droite montre l'évolution de l'erreur sur 2000 époques.

La figure 5 montre l'évolution de l'erreur sur le lot de test. La fonction réalisée par le réseau de neurones est représentée en pointillés sur la figure de gauche après 200 époques. A ce moment-là, le réseau n'a approximé que la première période de la fonction, qui a l'amplitude maximum. Au fur et à mesure de l'apprentissage, le réseau va approximer les périodes suivantes, ce qui correspond aux brusques chutes dans l'erreur. Après 800 époques, le réseau a pu approximer toutes les périodes de la fonction et l'erreur ne décroît plus. Si l'on veut revenir à l'analogie avec le fonctionnement du cerveau, on voit sur cet exemple simple que le réseau apprend d'abord les traits les plus importants du problème puis, si le réseau est suffisamment complexe et avec plus de temps, il apprendra les détails.

Si l'exemple précédent est très académique, des applications pratiques sont également possibles. Nous avons ainsi utilisé un perceptron multi-couches pour ajuster une fonction de correction de la mesure de la position des photons dans le calorimètre électromagnétique d'ATLAS [6]. Cette correction dépend de trois variables (la position du photon par rapport aux bords des cellules de lecture, la position absolue dans le calorimètre et la profondeur de la gerbe). La résolution sur la position obtenue en appliquant cette fonction est 10 à 15% meilleure qu'une correction ajustée manuellement.

3.3 Ajustement d'une fonction sur des exemples bruités

Le plus souvent, les exemples fournis au réseau de neurones lors de l'apprentissage sont bruités, c'est-à-dire que la réponse y_i pour l'exemple i ne vaut pas exactement $f(x_i)$. Dans ce cas, on peut montrer facilement que le réseau va approximer la fonction $< y_i > (x)$, qui est égale à $f(x)$ dans la limite d'un grand nombre d'exemples et si le bruit à une valeur moyenne nulle.

[5] Il est très instructif d'essayer un nombre croissant de neurones sur ces exemples simples, et de voir comment l'ajustement de la fonction s'améliore avec le nombre de neurones.

Cette propriété est illustrée dans la figure 6. On a repris ici l'exemple de l'ajustement de la fonction $sin(x)/x$, mais on a ajouté aux 200 exemples d'apprentissage un bruit plat compris entre -0.05 et 0.05. L'erreur sur le lot d'apprentissage est limitée par la variance du bruit. Le lot de test n'a, lui, pas été bruité, et l'on constate que l'erreur sur ce lot est bien plus faible que sur le lot d'apprentissage. On voit ainsi que le réseau a bien appris la fonction sous-jacente.

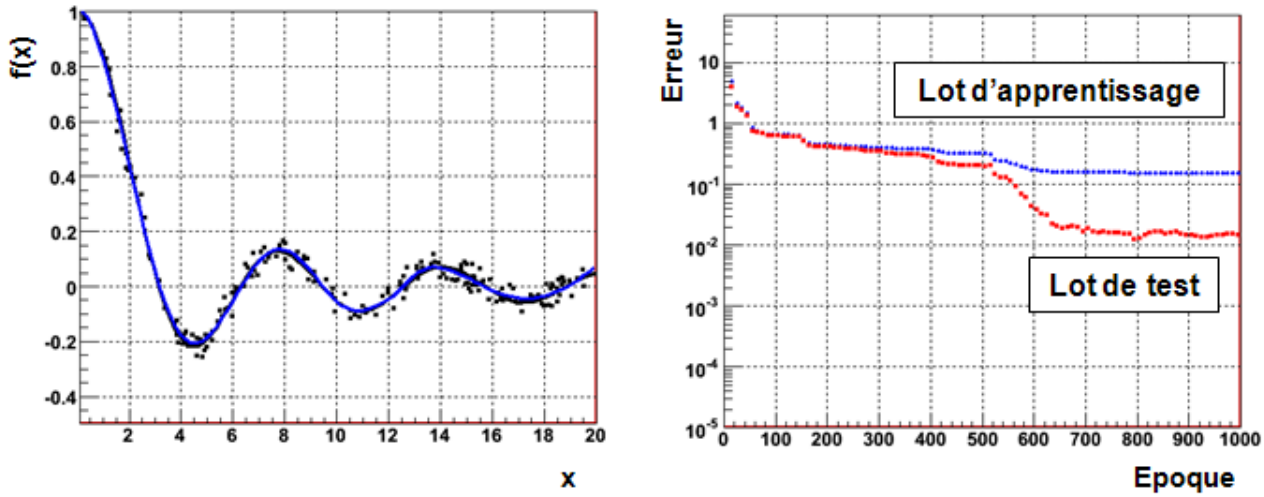

Fig. 6: Ajustement d'une fonction bruitée.

3.4 Utilisation du perceptron multi-couche pour la classification de données

L'utilisation d'un perceptron multi-couches pour la classification de données n'est qu'une application particulière de l'ajustement de fonctions. En effet, on montre facilement que lorsqu'on entraîne un tel réseau sur un lot d'apprentissage qui contient des exemples S de type signal, pour lesquels la réponse souhaitée est 1, et des exemples de type bruit de fond B, pour lesquels la réponse souhaitée vaut 0, alors le réseau approxime, en tout point $\vec{x}$ de l'espace des variables, la probabilité $p(S\|\vec{x})$,c'est-à-dire la fraction de signal en fontion de $\vec{x}$.

Pour vérifier cette propriété, on a réalisé l'exercice suivant : entraîner un réseau 1-2-1 à séparer un lot constitué de 70% de bruit de fond et 30% de signal, en utilisant une seule variable. Dans cette variable, le bruit de fond a une distribution gaussienne de sigma 1 centré à 0 alors que le signal a une distribution gaussienne de sigma 1 centrée à 1. La fonction $p(S\|\vec{x})$ est représentée sur la figure 7, ainsi que $mlp(x)$ après apprentissage [6]. On constate que, dans le domaine $[-1, 2]$, qui contient un nombre suffisant d'exemples, le réseau donne une bonne approximation de la fonction $p(S\|\vec{x})$.

Couper sur la valeur de sortie du réseau pour classer (ou non) un nouvel exemple dans la catégorie signal revient à sélectionner les régions de l'espace où la probabilité de signal est la plus grande. La coupure peut-être choisie de plusieurs façons :

- couper à 0.5 : cette coupure minimise le taux d'erreur total (probabilité de prendre du signal pour du bruit de fond et inversement)
- prendre du signal pour du bruit de fond ou inversement n'a pas toujours les mêmes conséquences [7]. Il faut alors ajuster la coupure pour minimiser le risque.
- on peut aussi choisir deux seuils : un seuil en-deçà duquel l'exemple est classé comme bruit de fond et un seuil au-delà duquel l'exemple est classé comme signal. Entre les deux seuils, un traitement supplémentaire doit être effectué [8].

Cette vision mathématique de l'utilisation des perceptrons multi-couches pour la classification de données permet de répondre à plusieurs questions pratiques :

[6] L'histogramme est, dans chaque bin, le rapport $N(S)/(N(S)+N(B))$ sur le lot d'exemple. Il est donc ce que voit le réseau de la fonction $p(S\|\vec{x})$ pour apprendre.

[7] L'exemple donné classiquement est l'analyse d'images médicales.

[8] Par exemple la lecture par un œil humain dans le cas de la reconnaissance de caractères manuscrits.

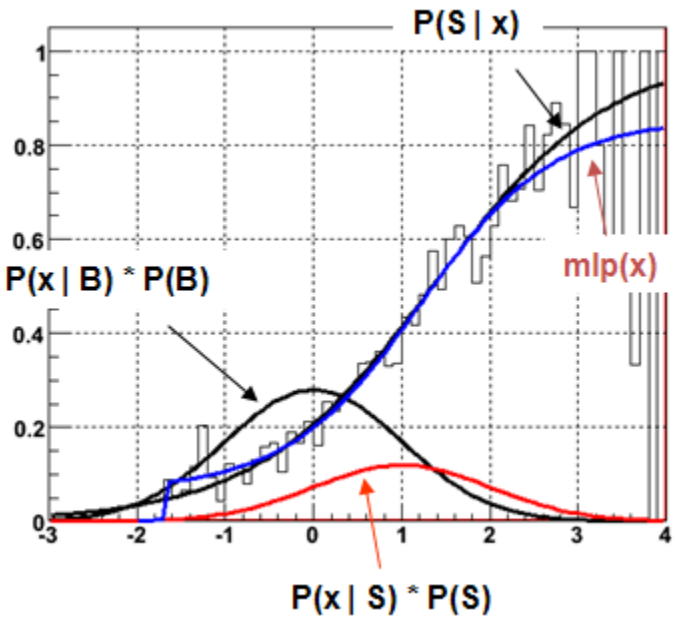

Fig. 7: Utilisation d'un perceptron multi-couches pour la classification de données.

- La sortie du réseau n'est pas nécessairement comprise strictement entre 0 et 1 : même si la probabilité de signal est effectivement comprise dans ces limites, le réseau n'en donne qu'une approximation.
- Un grand nombre d'exemples avec une sortie très différente de 0 ou de 1 ne signifie pas nécessairement que le réseau n'a pas convergé : il peut signifier que les exemples ne sont pas facilement séparables avec les variables considérées.
- En particulier, une sortie toujours différente de 1 pour le signal (par exemple) est possible (pas de région où il y a seulement du signal).

3.5 La vision mathématique de l'apprentissage

L'erreur $E = \Sigma_p(mlp(\vec{x_p}, \vec{w}) - d_p)^2$ est, étant donné un lot d'apprentissage, une fonction des poids $\vec{w}$. L'apprentissage n'est autre qu'un problème de minimisation, qu'on peut résoudre en utilisant des méthodes de minimisation standard.

On pourrait, en principe, utiliser Minuit pour minimiser cette fonction, mais ce ne serait sans doute pas la méthode la plus efficace car :

- le nombre de paramètres est très grand. Par exemple, un réseau 2-3-1 a déjà 13 paramètres.
- d'autre part, nous allons voir qu'on peut calculer la dérivée de E par rapport aux w_{ij} (c'est-à-dire le gradient ∇E). La connaissance du gradient permet d'utiliser des méthodes de minimisation plus efficaces.

3.5.1 Calcul du gradient

Chaque composante $\partial E/\partial w_{ij}$ est la somme sur les exemples $\Sigma_p \partial e_p/\partial w_{ij}$. En partant du neurone de sortie, on peut développer cette expression, et on arrive au résultat $\partial e_p/\partial w_{ij} = \delta_i y_j$, où les δ_i ne sont autres que les quantités calculées dans la section 2.2. Autrement dit, ce qui est traditionnellement appelé *méthode de rétropropagation des erreurs* n'est autre que le calcul du gradient de l'erreur.

A partir de la connaissance de ce gradient, on peut utliser plusieurs méthodes de minimisation.

3.5.2 Minimisation stochastique

En 1951, H.Robbins et S.Monro décrivent, dans un article intitulé *A Stochastic Approximation Method* [7], une méthode pour trouver le zéro d'une fonction inconnue à partir d'estimation successives $y_t \simeq f(x_t)$ de cette fonction (mesures entachées d'erreurs). Si dans leur méthode, on remplace *trouver le*

zéro d'une fonction par *trouver le zéro du gradient* ∇E et *estimation successives* y_t *de* $f(x_t)$ par *estimation de* ∇E *par* ∇e_p, on obtient exactement la méthode traditionnelle d'entraînement d'un perceptron multi-couches.

Autrement dit, la méthode traditionnelle d'entraînement d'un perceptron multi-couches n'est autre que le calcul du gradient de E par *rétropropagation des erreurs* et la recherche d'un zéro de ce gradient par utilisation de la méthode de minimisation stochastique de Robbins - Monro.

Les conditions nécessaires à la convergence de cette méthode sont connues [8]. Il faut en particulier :

- que les mesures successives ne soient pas corrélées. C'est pour cette raison qu'il faut mélanger les exemples du lot d'apprentissage.
- assurer la décroissance de l'erreur en faisant tendre le paramètre d'apprentissage η vers 0. En pratique, ce paramètre est souvent gardé fixe et il n'y a alors pas de garantie de convergence.

A cause de l'aspect stochastique de cette méthode, la convergence est souvent très lente. On peut illustrer le fonctionnement et la lenteur de cette méthode sur l'exemple suivant (qui n'a aucun rapport avec les réseaux de neurones) : trouver les paramètres a et b de la droite $y = 2x + 1$ à partir de 10 points (x_p, y_p) pris au hasard dans l'intervalle [0,1].

La fonction à minimiser est $E(a,b) = \Sigma_p(ax_p + b - y_p)^2$. On calcule donc, en partant par exemple de $a_0 = -3$ et $b_0 = -2$, pour chaque exemple $\partial e/\partial b = ax_p + b - y_p$ et $\partial e/\partial a = x_p * \partial e/\partial b$, puis on modifie a et b de la façon suivante : $a, b \rightarrow a, b - \eta\partial e/\partial a, b$. Les valeurs successives de a et b sont représentées sur la figure 8, en même temps que les iso-contours de E. On constate que a et b tendent bien vers le minimum de E, mais cette convergence nécessite de nombreux pas et présente de nombreuses oscillations.

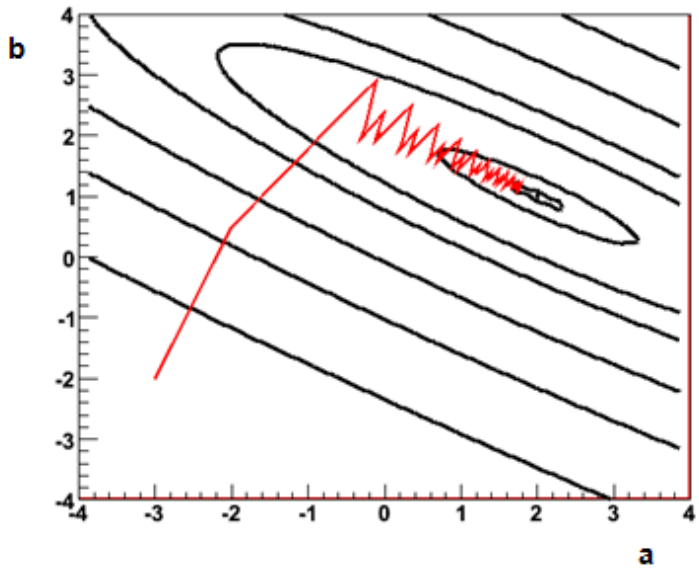

Fig. 8: Ajustement d'une droite par la méthode de minimisation stochastique.

Plusieurs variantes ont été proposées pour améliorer la vitesse de convergence de la méthode stochastique, comme l'ajout d'un terme de moment [9], adapter la longueur des pas [10], ne modifier les poids qu'après avoir vu tous les exemples [11]

Même avec ces variantes, la minimisation stochastique n'est que très rarement aussi efficace que les méthodes de minimisation sans contraintes que nous allons voir dans la section suivante.

[9] On modifie les poids selon la formule $\Delta w_{ij}^t = -\eta\partial e/\partial w_i j + \mu Deltaw_{ij}^{t-1}$ de façon à ne pas défaire ce qui a été fait au pas précédent.

[10] Adaptive Step Size Method

[11] *Batch* ou *offline* backpropagation. Cette méthode n'a en réalité que des inconvénients.

3.5.3 Utilisation des méthodes de minimisation sans contraintes

Les méthodes de minimisation [9] décrites ici fonctionnent toutes de la façon itérative suivante :
- Calculer à l'étape t le gradient ∇E_t, qui sera noté $g(w_t)$ dans la suite
- En déduire une direction d_t dans l'espace des w
- Rechercher le minimum de E le long de cette direction, c'est-à-dire trouver λ^* qui minimise $E(w_t + \lambda d_t)$
- Faire $w_{t+1} = w_t + \lambda^* d_t$

Elles diffèrent par le choix de la direction d_t. Leur convergence est démontrée pour des formes quadratiques, ce qui n'est bien sûr pas le cas en pratique, mais qui est toujours une approximation locale de $E(\vec{w})$. La recherche d'un minimum le long de la direction assure en pratique la décroissance de l'erreur. Notons enfin que, contrairement à la minimisation stochastique, les poids ne sont pas modifiés après chaque exemple [12].

Voici quelques possibilités pour choisir la direction :
- **Méthode de la plus grande pente** [10].
  La direction est simplement l'opposé du gradient. On peut démontrer que les pas successifs sont perpendiculaires, ce qui induit des oscillations dans la recherche du minimum.
- **Méthode des gradients conjugués** [11].
  Pour trouver le minimum d'une forme quadratique $q(x) = x^T Ax + b^T x + c$ (avec A définie positive), on peut montrer qu'il existe des directions successives d_t telles que, pour à chaque étape t, le gradient en tout point de $w_t + \lambda d_t$ est perpendiculaire au sous espace engendré par les d_i, $i < t$. Ainsi, après n étapes le gradient est nécessairement nul, et on atteint donc le minimum. Il existe plusieurs méthodes pour calculer les directions successives.
- **Méthode de Newton**
  Pour une forme quadratique $q(x) = x^T Ax + b^T x + c$, on peut, en partant du point x, trouver le minimum x^* par $x^* = x - A^{-1} g(x)$, où $g(x)$ est le gradient au point x. Plus généralement, si on connait la matrice H des dérivées secondes (le *Hessien*), on peut choisir $d_t = H^{-1} g_t$. Dans le cas des perceptrons multi-couches, on peut en principe effectivement calculer le Hessien en tout point, mais le calcul de H^{-1} est long (n^3 opérations) et sujet à des instabilités numériques.
- **Méthode BFGS (Broyden Fletcher Goldfarb Shanno)** [12].
  On calcule itérativement une approximation G_t de l'inverse du Hessien en partant d'une matrice G_0 définie positive (l'identité est l'exemple le plus simple). Cette méthode est souvent très puissante, c'est pourquoi je donne ici la formule qui permet de calculer G_t, pour le lecteur qui souhaiterait la programmer [13] :

$$G_{t+1} = G_t + [1 + \frac{\gamma_t^T G_t \gamma_t}{\delta_t^T \gamma_t}] \frac{\delta_t \delta_t^T}{\delta_t^T \gamma_t} - \frac{\delta_t \gamma_t^T G_t + G_t \gamma_t \delta_t^T}{\delta_t^T \gamma_t}$$

$$\text{avec } \delta_t = w_{t+1} - w_t$$

$$\text{et } \gamma_t = g_{t+1} - g_t$$

Signalons enfin que cette liste de méthodes de minimisation n'est pas exhaustive. Certains logiciels proposent par exemple la méthode de Levenberg Marquardt.

3.5.4 Comparaison des méthodes

Tout bon logiciel de réseaux de neurones doit donc proposer plusieurs "méthodes d'apprentissage", au minimum la minimisation stochastique traditionnelle et une méthode de type $BFGS$. On peut alors, sur un problème donné, comparer les deux méthodes d'apprentissage.

La méthode $BFGS$ est souvent la méthode la plus puissante. Sur les petits exemples de la section 3.2, elle est beaucoup plus rapide que la minimisation stochastique. La figure 9 montre une comparaison de plusieurs méthodes d'apprentissage sur l'ajustement de la fonction de correction de la mesure de position dans le calorimètre électromagnétique d'ATLAS (voir la section 3.2).

[12] La boucle sur les exemples consiste ici juste à calculer $E = \Sigma_p e_p$.
[13] La démonstration de cette formule est bien sûr en dehors du sujet de ce cours.

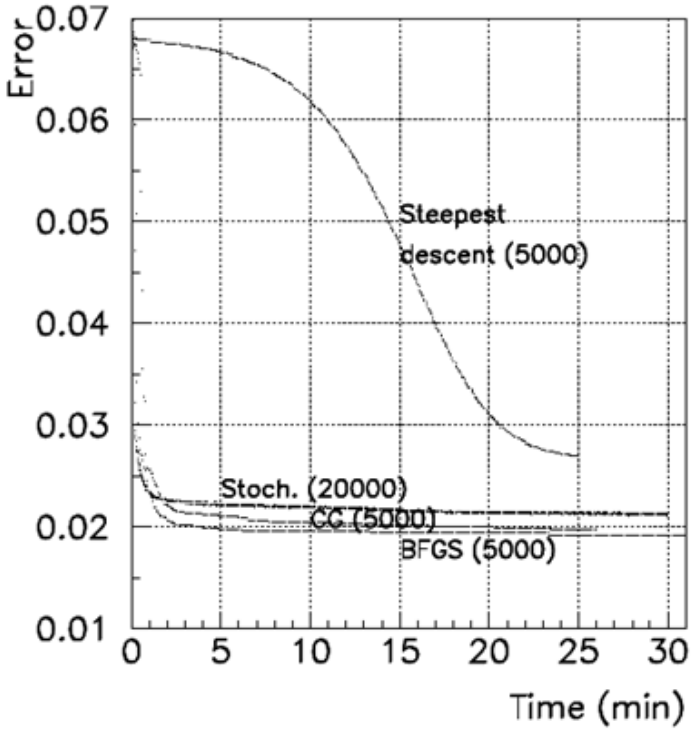

Fig. 9: Comparaison de plusieurs méthodes d'apprentissage sur l'ajustement de la fonction de correction de la mesure de position dans le calorimètre électromagnétique d'ATLAS.

L'échelle horizontale de cette figure est la durée de l'apprentissage (en minutes). Le nombre d'époques pour chaque méthode est indiqué entre parenthèses. La méthode de la plus grande pente a la moins bonne performance. La meilleure méthode est $BFGS$, qui conduit à une résolution 10% meilleure que la méthode stochastique.

3.5.5 Minima locaux

Quelle que soit la méthode d'apprentissage utilisée, il n'y a aucune garantie que la minimisation trouve le minimum global de $E(\vec{w})$. Si les paramètres $\vec{w}$ initiaux sont proches d'un minimum local, alors les méthodes de minimisation décrites précédemment vont conduire à ce minimum local : la recherche d'un minimum le long de la direction force en effet une décroissance de l'erreur, et rien n'autorise ces méthodes à "sauter" dans une vallée plus profonde. La minimisation stochastique n'impose pas une décroissance de l'erreur globale à chaque pas (ni même à chaque époque), ce qui permet éventuellement d'échapper par hasard à un minimum local.

La meilleure façon d'éviter un minimum trop mauvais est d'essayer plusieurs ensembles de poids initiaux, pris au hasard. La figure 10 montre que ceci est utile même sur des exemples très simples, comme l'ajustement de la fonction $sin(x)/x$. Dans cet exemple, l'un des ensembles de poids initiaux conduit à une erreur 1000 fois plus importante que les meilleures solutions.

Il faut enfin remarquer que la recherche d'un minimum dans une zone très plane peut échouer à cause de problèmes d'arrondis.

3.6 Méthode d'apprentissage évolutionnaire + gradient

Afin de limiter les problèmes de minimas locaux, des méthodes d'apprentissage hybrides évolutionnaire + gradient sont possibles (voir par exemple [13]). Elles considèrent un ensemble de poids comme un *individu* et partent d'une *population* constituée d'un grand nombre d'individus. On peut alors procéder de la façon suivante :

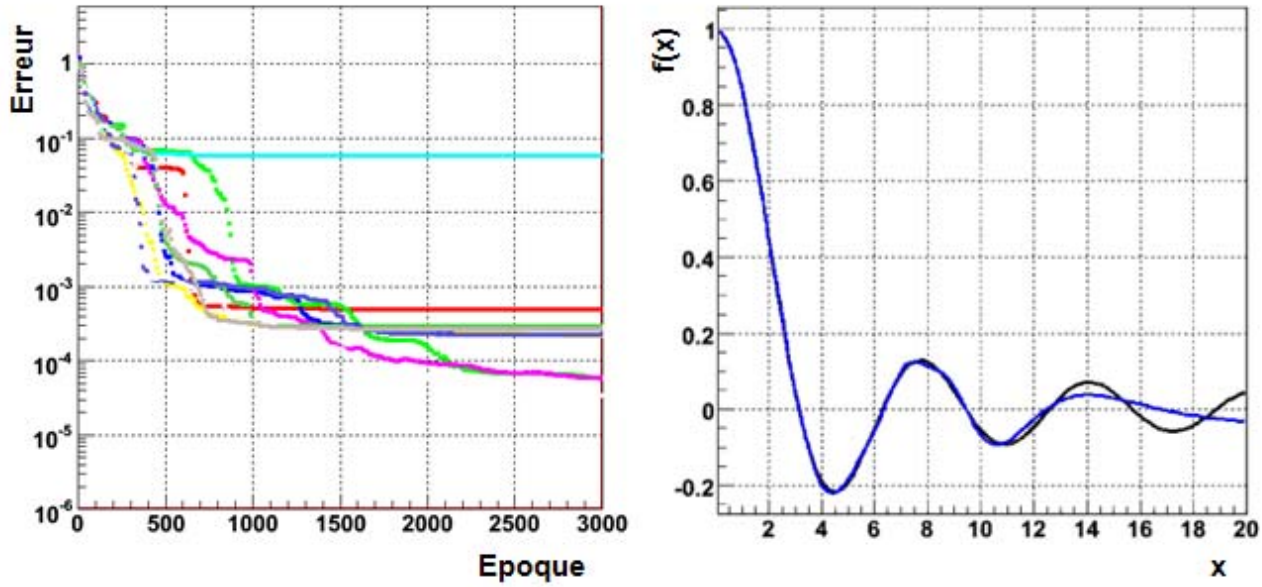

Fig. 10: Erreur dans l'apprentissage de la fonction $sin(x)/x$ par la méthode $BFGS$, en partant de plusieurs ensembles de poids initiaux. La fonction obtenue dans le plus mauvais cas est représentée sur la figure de droite.

- entraîner chaque individu et une mutation de cet individu (c'est-à-dire un autre individu obtenu en modifiant légèrement quelques poids)
- conserver les meilleurs individus, c'est-à-dire ceux qui conduisent à l'erreur la plus petite
- recommencer

4 Conseils pour l'utilisation d'un perceptron multi-couches

Le but de cette section est de répondre à certaines questions posées souvent sur l'utilisation des réseaux de neurones.

4.1 Choix des variables pour la classification des données

Le but est de trouver une zone de l'espace des variables où la probabilité de signal est plus grande que la probabilité de bruit de fond. Mais le perceptron ne pourra pas inventer une différence entre le signal et le bruit de fond si elle n'existe pas ! C'est pourquoi, il est utile de lui fournir les variables les plus discriminantes, lorsqu'on les connaît.

Cette évidence doit toutefois être pondérée par la considération suivante : des variables qui ne sont pas individuellement discriminantes peuvent le devenir lorsqu'elles sont considérées simultanément. Ceci est illustré sur la figure 11, où aucune des variables x_1 et x_2 n'est discriminante et où, pourtant, leur utilisation simultanée permet de séparer parfaitement le signal du bruit de fond.

4.2 Pré-processing des variables

Toute utilisation de connaissances *a priori* pour pré-processer les variables est utile. Par exemple, si l'on sait que le rapport de deux énergies E_1 et E_2 est plus pertinent que chacune des énergies, il vaut mieux donner au réseau le rapport E_1/E_2.

La référence [14] décrit l'utilisation des réseaux de neurones pour la reconnaissance de caractères manuscrits. Un pré-traitement des images (redressement) permet de réduire le taux d'erreur d'un facteur 3.

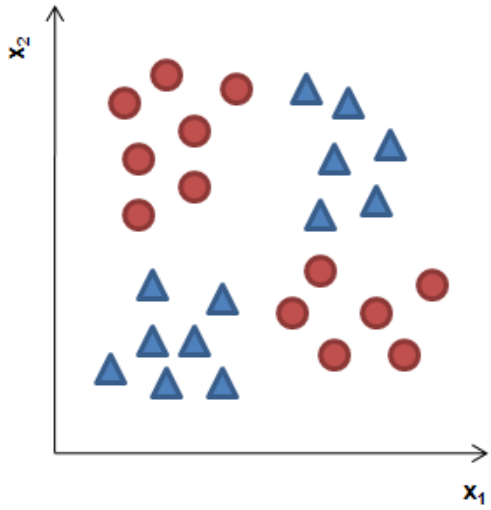

Fig. 11: Exemple de l'utilisation de variables non discriminantes pour séparer un signal (les triangles par exemple) d'un bruit de fond (les ronds).

4.3 Normaliser les entrées

La dérivée de la fonction sigmoïde est représentée sur la figure 12. Elle tend rapidement vers 0 quand la valeur de l'argument devient grand, ce qui peut poser deux problèmes :

- le gradient de la fonction d'erreur est très petit et la convergence de l'apprentissage est beaucoup plus lente.
- à cause de la précision numérique limitée, la dérivée apparaît comme nulle et l'apprentissage s'arrête.

C'est pourquoi les valeurs des variables d'entrée du réseau doivent être proches de zéro, typiquement dans le domaine $[-5, 5]$. Par exemple si les énergies valent plusieurs centaines de GeV, les diviser par 100, etc.

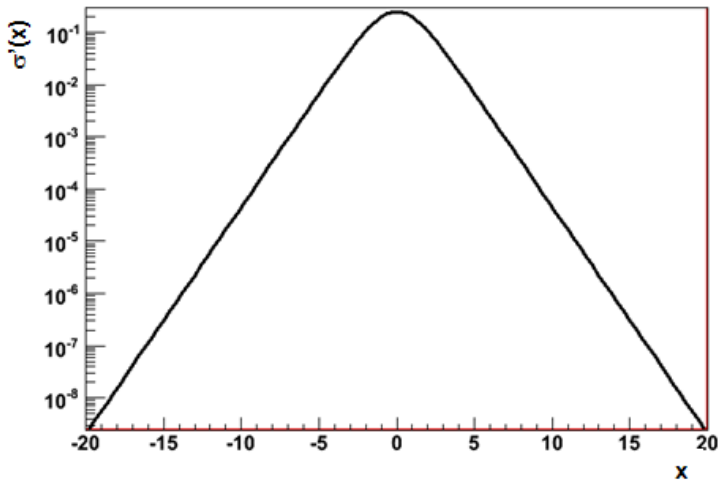

Fig. 12: Dérivée de la fonction sigmoïde.

4.4 Nombre d'exemples

Les exemples doivent être choisis pour couvrir tout l'espace des variables. Un réseau de neurones n'est pas un devin : il peut avoir du mal à extrapoler (et même à interpoler) dans des zones où il y a peu d'exemples.

Ce problème est illustré sur la figure 13 : les zones du plan où $x_1 x_2 > 0$ ne contiennent que du signal, le reste du plan ne contient que du bruit de fond. Les courbes sur la figure sont les contours $mlp(x_1, x_2) = 0.5$. A cause du faible nombre d'exemples, le réseau approxime mal la séparation entre le signal et le bruit de fond, et se trompe dans un certain nombre de régions.

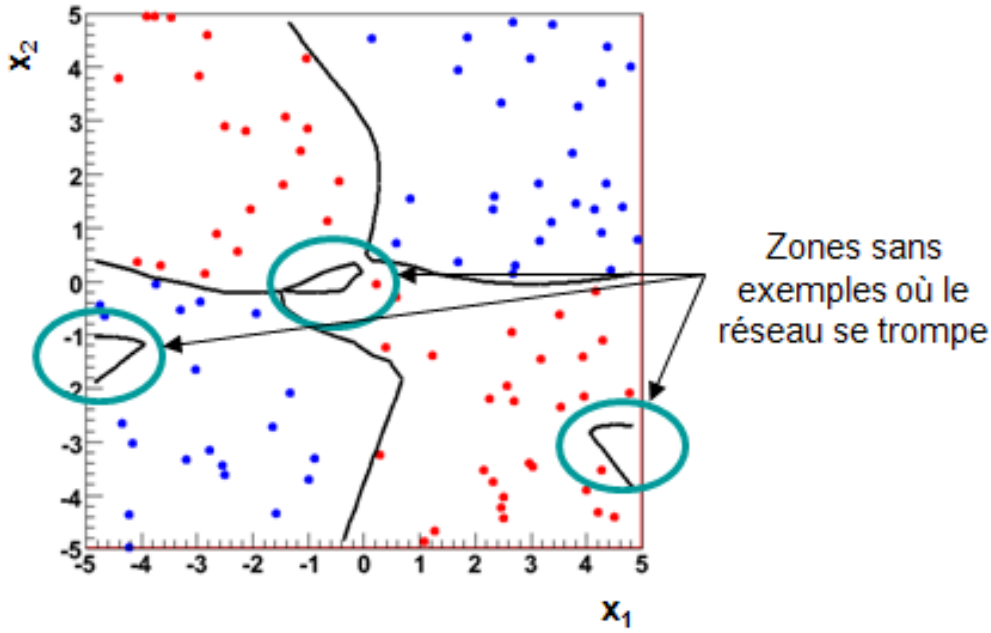

Fig. 13: Illustration du problème de l'interpolation dans des régions sans exemple.

Pour éviter ce problème, il faudrait donc donner au réseau un nombre d'exemples important. Malheureusement, le nombre d'exemples nécessaires croît comme A^{dim} où dim est la dimension du problème, et pour des problèmes avec un grand nombre de variables, il risque d'être difficile d'avoir un nombre suffisant d'exemples.

En pratique, il faut essayer de vérifier si la performance du réseau s'améliore avec le nombre d'exemples. La figure 14 montre le résultat sur un exercice de reconnaissance de chiffres manuscrits [14]. On constate que la performance s'améliorer effectivement, pour une durée d'apprentissage fixe.

4.5 Nombre de couches cachées

Nous avons vu dans la section 3.1 qu'une couche cachée suffit en principe pour résoudre tous les problèmes. En pratique, il peut s'avérer plus efficace d'utiliser deux couches cachées. La figure 15 compare l'utilisation d'un réseau à une couche cachée et un réseau à deux couches cachées sur le problème de la séparation d'un signal qui a la forme d'un "carré creux" (voir section 2.3). Un réseau 2-20-1 donne une approximation de la solution, mais avec beaucoup de fluctuations dans les zones planes. Pour diminuer ces fluctuations, ont peut augmenter le nombre de neurones de la couche cachée. Mais la solution la plus efficace, sur ce problème, est d'utiliser une deuxième couche cachée.

4.6 Nombre de neurones

Il n'existe pas de règle pour choisir le nombre de neurones dans la ou les couche(s) cachée(s). Ce nombre dépend du problème et de la précision souhaitée. En pratique, il vaut mieux partir d'un réseau relativement petit, pour éviter les problèmes de sur-apprentissage, et augmenter la taille du réseau en vérifiant l'amélioration de la performance.

[14] Les exemples de chiffres manuscrits ont été obtenues de la base de données disponible sur le site http://yann.lecun.com/exdb/mnist/. Les chiffres sont des images de 28 x 28 pixels. Le réseau utilsé est un réseau 784-300-10, soit environ 238000 paramètres.

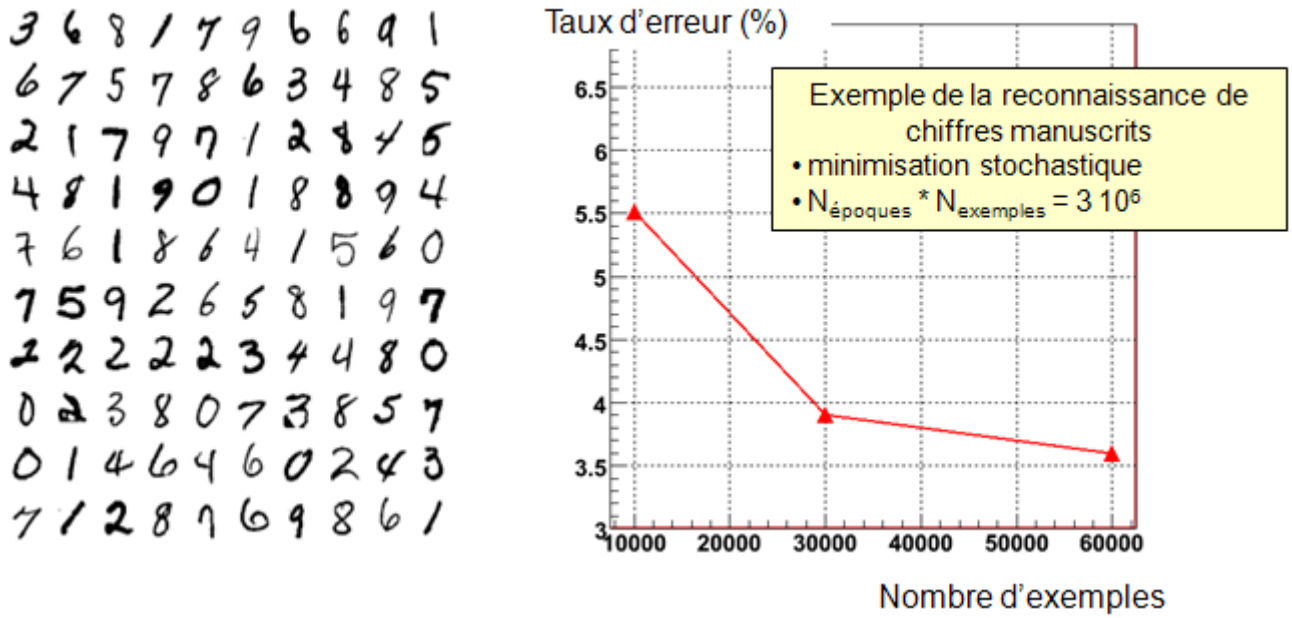

Fig. 14: Performance d'un réseau en fonction du nombre d'exemples.

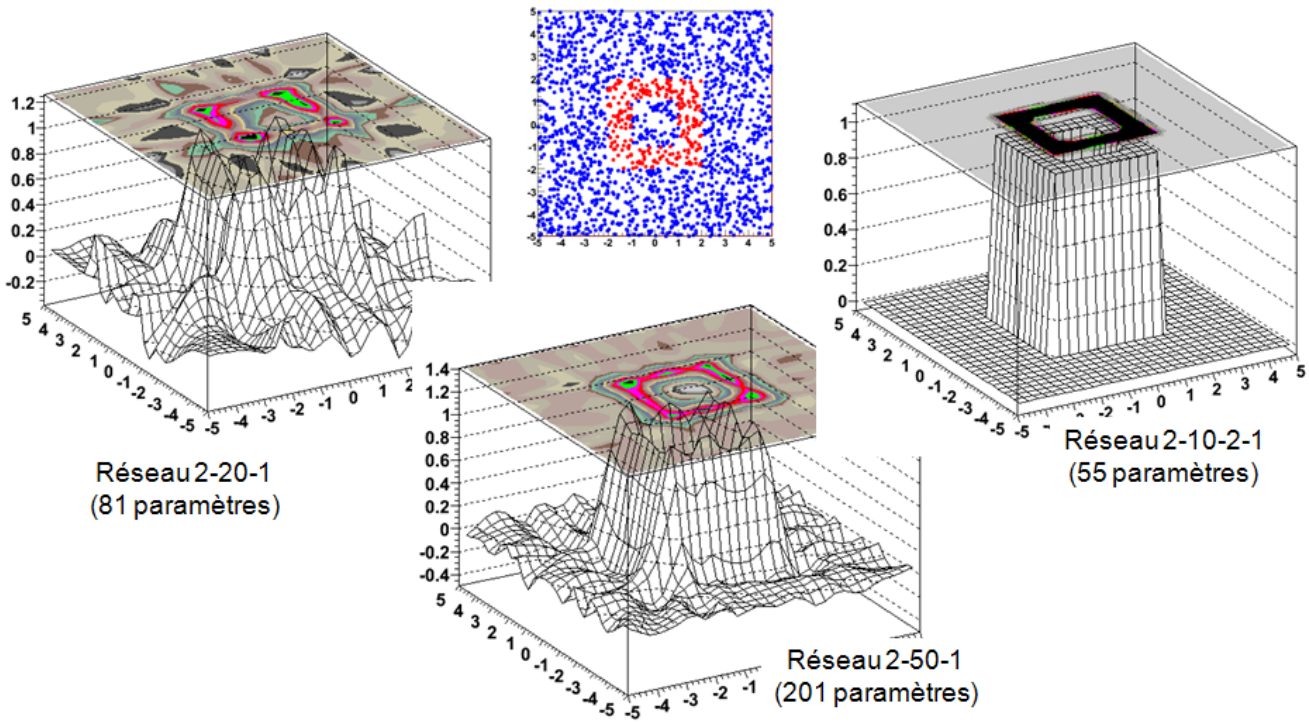

Fig. 15: Illustration de l'utilisation de plus d'une couche cachée.

5 Conclusion

Ce cours décrit le fonctionnement et l'utilisation des perceptrons multi-couches. Historiquement, la description de ce type de réseau de neurones et de l'apprentissage est basée sur l'analogie avec le fonctionnement du cerveau. J'ai voulu montrer dans ce cours qu'une vision mathématique est très importante pour comprendre le fonctionnement de ces réseaux. J'espère avoir en particulier enseigné au lecteur que :

- un perceptron multi-couches est avant tout un outil d'approximation de fonctions. La classification de données n'est qu'une application particulière de cet outil.
- l'apprentissage n'est autre qu'un problème de minimisation. A côté de la minimisation stochastique, qui devrait être la vraie appellation de la méthode traditionnelle d'apprentissage, d'autres méthodes de minisation sont souvent beaucoup plus performantes.

Ce cours doit beaucoup à la lecture des deux ouvrages suivants, que le lecteur motivé pourra consulter avec profit :

- C.M. Bishop : *Neural Networks for Pattern Recognition*, Clarendon Press Oxford (1995)
- R.Fletcher : *Practical Methods of Optimization*, 2nd edition, Wiley (1987)

Références

[1] B.H.Denby : *Neural Networks And Cellular Automata In Experimental High-Energy Physics*, Comput.Phys.Commun.49 :429-448,1988

[2] W.S. McCulloch, W. Pitts (1943) : *A logical calculus of the ideas immanent in nervous activity*, Bulletin of Mathematical Biophysics, 5, 115-137.

[3] D.E.Rumelhart et al. : *Learning representations by back-propagating errors*, Nature vol. 323 (1986), p. 533

[4] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner : *Gradient-Based Learning Applied to Document Recognition*, Intelligent Signal Processing, 306-351, IEEE Press, 2001

[5] K.Hornik et al. : *Multilayer Feedforward Networks are Universal Approximators*, Neural Networks, Vol. 2, pp 359-366 (1989)

[6] J.Schwindling : *S-shape correction using a neural network* , ATL-LARG-98-104, 1998.

[7] H.Robbins et S.Monro : *A Stochastic Approximation Method*, Annals of Math. Stat. 22 (1951), p. 400

[8] A.Dvoretsky : *On Stochastic Approximation*, Proc. 3rd Berkeley Sym. on Math. Stat. and Prob., J.Neyman (ed.) (Berkeley : University of California Press, 1956), p. 39

[9] R.Fletcher : *Practical Methods of Optimization*, 2nd edition, Wiley (1987)

[10] A.L. Cauchy : *Méthode générale pour la résolution des systèmes d'équations simultanées*, Comptes Rendus Acad. Sc. Paris, XXV, (1847), p.536

[11] M.R.Hestenes, E.Stiefel : *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bureau of Standards,49, (1952), p.409

[12] Broyden, C. G. : *The Convergence of a Class of Double-rank Minimization Algorithms*, Journal of the Institute of Mathematics and Its Applications 1970, 6, 76-90
Fletcher, R. : *A New Approach to Variable Metric Algorithms*, Computer Journal 1970, 13, 317-322
Goldfarb, D. : *A Family of Variable Metric Updates Derived by Variational Means*, Mathematics of Computation 1970, 24, 23-26
Shanno, D. F. : *Conditioning of Quasi-Newton Methods for Function Minimization*, Mathematics of Computation 1970, 24, 647-656

[13] Xin Yao : *Evolving Artificial Neural Networks*, Proceedings of the IEEE, vol. 87, no 9, 1999

[14] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner : *Gradient-Based Learning Applied to Document Recognition*, Intelligent Signal Processing, 306-351, IEEE Press, 2001