Neuro App '94,
15-16 Dec. 1994,
Marseille.

UN ALGORITHME DE DIMENSIONNEMENT POUR ATTEINDRE UN RÉSEAU D'ARCHITECTURE MINIMALE

Christian Louis, Bernard Gitter & Fabien Moutarde

Alcatel Alsthom Recherche
Route de Nozay
91460 Marcoussis, France

Tel.: +33.1 64 49 16 97 - Telefax: +33.1 64 49 06 95
E-mail: louis@aar.alcatel-alsthom.fr

Résumé

Cette communication présente un algorithme d'élagage de neurones pour les réseaux multicouches. La méthode utilise les conditions de minimalité développées par Sussman, mais étendues à une analyse des dépendances linéaires entre les neurones d'une même couche cachée. Cette analyse est effectuée sur les exemples d'apprentissage. L'algorithme réduit automatiquement le nombre de neurones pour atteindre l'architecture minimale sans détériorer l'erreur d'apprentissage. Cet algorithme a été appliqué avec succès à la suite d'un algorithme constructif pour automatiser le dimensionnement de réseau sur un problème difficile de reconnaissance de formes lié au traitement du signal : la reconnaissance automatique de 10 modulations numériques à partir de signaux interceptés.

Mots clés

élagage de neurones, réseaux à couches, capacité, dépendances linéaires, traitement du signal, modulations numériques

Abstract

This paper introduces a new pruning algorithm for multilayer feedforward neural networks. Sussman demonstrated that under three conditions on its weights, a neural network is minimal, i.e. its input-output map cannot be obtained from another net with fewer hidden nodes. Our method uses the ideas of Sussman, restricting the input-output map to the learning set of examples, but extending the conditions to an analysis of the linear dependencies between the neurons of the same layer. This technique allows to automatically reduce the size of a network and obtain a minimal architecture without changing the learning error rate. This algorithm was used successfully after a constructive algorithm on a pattern recognition problem in the signal processing field: the automatic recognition of 10 types of numerical modulations.

Keywords

neural network pruning, back-propagation, linear dependencies, signal processing, modulation recognition

Cette étude a été en partie financée par le marché DRET n°89.301.01.024

INTRODUCTION

Une difficulté majeure dans l'apprentissage des réseaux multicouches est le dimensionnement de l'architecture. Les études menées sur la notion de capacité ne permettent pas encore de déterminer a priori la bonne architecture d'un réseau [10]. Des méthodes constructives ont été développées par divers auteurs ([1] et [2]) pour pallier cet inconvénient. Ces algorithmes [6] déterminent lors de l'apprentissage le nombre de neurones et de connexions nécessaires à la résolution du problème mais ont tendance à surdimensionner les réseaux.

Pour obtenir une architecture optimale, une méthode duale consiste à partir d'une architecture surdimensionnée, à effectuer un apprentissage du type rétro-propagation, puis à élaguer des connexions ou à supprimer des neurones inutiles afin d'obtenir le réseau minimal pour un problème donné. De nombreuses méthodes existent pour la suppression de connexions, fondées soit sur des calculs de sensibilité de la fonction d'erreur vis-à-vis de la mise à zéro de connexions, soit sur la mise en place de termes de pénalité [7]. Elles sont généralement coûteuses en temps de calcul. Les méthodes de suppression de neurones sont moins nombreuses. On peut citer entre autres des heuristiques de Siestma et Dow [8].

Le nouvel algorithme présenté ici est une méthode de suppression de neurones. Il utilise une analyse des dépendances linéaires entre les états des neurones cachés, et un report de la contribution des neurones supprimés, ce qui permet de ne pas augmenter l'erreur d'apprentissage.

En partie 1, la méthode est présentée de façon théorique à partir des considérations développées par Sussman. La deuxième partie montre comment on peut maîtriser théoriquement la dérive de l'erreur due à la suppression des neurones grâce à une majoration explicite de celle-ci. Enfin, la troisième et dernière partie illustre l'intérêt pratique de cette méthode, d'abord sur un problème simple d'approximation de fonction, ensuite sur une application industrielle de classification de signal : la reconnaissance de modulations numériques sur des signaux interceptés. Il est montré comment on peut automatiser ainsi la construction d'un réseau en utilisant un algorithme constructif puis la méthode de suppression de neurones présentée.

1. LA MÉTHODE DE SUPPRESSION DE NEURONES

1.1. Notion de réseau minimal

Dans cette partie, on se restreint à des réseaux de neurones sigmoïdes qui ne possèdent qu'une couche cachée de n neurones. Sussman définit dans [9] la notion de réseau minimal : un réseau minimal est tel qu'on ne peut pas trouver de réseau avec un nombre inférieur de neurones qui réalise la même fonction entrée-sortie. Pour qu'un réseau ne soit pas minimal, i.e. qu'il soit réductible, il faut et il suffit (démontré dans [9]) qu'une des trois conditions suivantes soit vérifiée :

- il existe un neurone caché qui produit une contribution nulle à la sortie, autrement dit, dont les connexions partantes sont toutes nulles,
- il existe deux neurones cachés distincts dont les états sont toujours égaux ou toujours opposés,
- il existe un neurone caché constant.

Un tel réseau que l'on peut élaguer sans modifier la fonction d'entrée-sortie peut être identifié par une simple analyse des poids. Les trois conditions montrent qu'à peu près tous les réseaux sont minimaux, i.e. deux réseaux tirés au hasard n'ont presque aucune chance de calculer la même fonction.

De même que l'apprentissage est effectué sur un ensemble fini d'exemples, nous proposons de définir une notion de minimalité moins restrictive que celle de Sussman, en ne considérant l'équivalence des réseaux que sur un ensemble fini de points :

Un ensemble d'apprentissage et un réel positif ε étant fixés, on dira qu'un réseau est *ε-minimal* s'il n'existe pas de réseau de taille inférieure avec une erreur quadratique moyenne inférieure à ε sur l'ensemble d'apprentissage.

Un réseau minimal (chaque réseau ou presque) peut ne pas être ε-minimal s'il existe un réseau d'architecture plus simple réalisant certes une fonction entrée-sortie générale différente, mais ayant les mêmes performances d'apprentissage.

Le but de l'algorithme présenté est la recherche d'un réseau ε-minimal à partir d'un ensemble d'apprentissage donné et d'un réseau surdimensionné approchant cet ensemble à ε près.

1.2. Principe de la méthode

La méthode d'élagage que nous proposons se compose des 3 étapes ci-dessous.

1. Trouver les dépendances linéaires entre neurones cachés.
   1.a Calculer la matrice de covariance sur la base des états des neurones cachés (voir définition explicite au §1.3).
   1.b Trouver une base orthonormée du noyau de cette matrice.
   1.c En déduire les dépendances linéaires.
2. Choisir les neurones à supprimer, et les exprimer en fonction des autres.
3. Compenser leur suppression en reportant leur contribution sur les autres neurones.

Chacune de ces étapes est explicitée dans les sous-sections suivantes. Si l'on applique exactement cette technique, la taille de la couche cachée du réseau est réduite, en laissant l'erreur d'apprentissage rigoureusement inchangée.

1.3. Recherche des dépendances linéaires

La recherche des dépendances linéaires entre les neurones cachés peut se faire à l'aide de la matrice de covariance des états des neurones sur la base. En effet, ces dépendances linéaires indépendantes correspondent exactement aux vecteurs du noyau de la matrice de covariance.

Théorème

Soit un réseau de neurones ayant une seule couche cachée de n neurones, et soit N exemples d'apprentissage. On note x_i^k la sortie du i-ème neurone caché pour l'exemple numéro k, $\overline{x_i} = \frac{1}{N}\sum_{k=1}^{N} x_i^k$ la moyenne des sorties du i-ème neurone caché sur les N exemples, et ξ_i le vecteur colonne ${}^t(x_i^1 \ \ldots \ x_i^N)$.
On définit par ailleurs la matrice $T=(t_{ij})$ de covariance des états des neurones cachés par $t_{ij}=(\xi_i - \overline{x_i}1_N)\bullet(\xi_j-\overline{x_j}1_N)$[1] où 1_N est le vecteur colonne à N composantes 1.

S'il existe un vecteur $a={}^t(a_1 \ \ldots \ a_n)$ non nul et un réel β tels que $\sum_{i=1}^{n} a_i\xi_i = \beta 1_N$, alors le vecteur a appartient au noyau de T.

Inversement, si on se donne une base orthonormée du noyau de T, alors les coordonnées de chacun des vecteurs de cette base sont les coefficients d'une dépendance linéaire (constante sur l'ensemble des N exemples) entre les états des neurones cachés.

[1] Le symbole • indique le produit scalaire de 2 vecteurs dans tout l'article.

La démonstration est donnée en annexe 1.
Un réseau et une base étant fixés, on peut donc calculer T, puis rechercher une base orthonormée de son noyau. Si on définit alors d'une part la matrice A dont les lignes sont les coordonnées de ces vecteurs propres du noyau de T, et d'autre part le vecteur b des valeurs constantes sur l'ensemble d'apprentissage, alors on a trouvé les dépendances linéaires entre les états des neurones cachés :

$$A\begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \cdot \\ \cdot \\ b_p \end{bmatrix} \qquad (1)$$

1.4. Choix des neurones à éliminer

Il reste à déterminer les neurones à supprimer parmi ceux de la dépendance, et à les exprimer en fonction des autres. On a en général p dépendances linéaires entre n neurones c'est à dire un noyau de T de dimension p, avec p<n si tous les neurones ne sont pas constants, i.e. si T n'est pas nulle. Ceci revient à résoudre (1) qui est un système linéaire de Cramer sous-contraint (A étant une matrice pxn de rang p), en utilisant comme variables-paramètres les états des neurones susceptibles de ne pas être supprimés.

Le choix de la matrice carré pxp inversible extraite de A peut être fait plus ou moins astucieusement. Pratiquement, on peut faire comme suit : au début de l'algorithme, les colonnes de A sont classées par ordre croissant de variance, pour que les neurones de plus faible variance soient supprimés de préférence à ceux qui varient beaucoup sur la base d'exemples. L'algorithme utilisé consiste, en supposant que les j-1 premières colonnes de A forment un système libre, à regarder si les j premières colonnes de A sont linéairement indépendantes. Si oui, l'algorithme continue au rang j+1, sinon, on change l'ordre des colonnes de A en mettant la $j^{ème}$ colonne de A en dernier et on teste de nouveau l'indépendance linéaire des j nouvelles premières colonnes de A. On est ainsi assuré d'extraire p colonnes libres de A, qui correspondent alors aux p neurones qui seront supprimés en inversant le système de Cramer.

1.5. Compensation des neurones éliminés

Supposons que les étapes précédentes nous aient permis de déterminer des réels (α_2, ..., α_n) tels que l'on ait sur toute la base d'exemples :

$$x_1 = \sum_{i=2}^{n} \alpha_i x_i \qquad \text{(où les xi sont les états des n neurones cachés)}$$

alors en appelant w_1, ..., w_n les poids reliant les neurones de la couche cachée à sortie, on a comme potentiel y de la sortie :

$$y = \sum_{i=1}^{n} w_i x_i = w_1 \sum_{i=2}^{n} \alpha_i x_i + \sum_{i=2}^{n} w_i x_i = \sum_{i=2}^{n} (w_i + \alpha_i w_1) x_i$$

Ainsi, il suffit de changer les poids w_i en ($w_i + \alpha_i w_1$) pour que les performances du réseau soient inchangées sur la base d'exemples après la suppression du neurone caché x_1.

1.6. Extensions de la méthode

En pratique, la matrice de covariance ne possède généralement pas la valeur propre zéro, i.e. aucun neurone n'est rigoureusement combinaison linéaire d'autres neurones. On choisit donc un seuil positif en deçà duquel une valeur propre de la matrice de covariance est considérée comme nulle, et on applique exactement le même algorithme

que si ces valeurs propres étaient exactement nulles[1]. On accepte alors une suppression de neurones qui va modifier l'erreur d'apprentissage. Plus le seuil choisi est important, plus les neurones supprimés seront en grand nombre, mais plus le risque de s'éloigner du réseau initial (et d'accroitre l'erreur) grandit. Mais nous verrons au §2. que la dérive de l'erreur peut être majorée.

Les résultats montrés ici se généralisent facilement à des réseaux comportant plus de couches cachées, au prix de notations plus lourdes. Cette méthode s'applique à tous les réseaux à couches et permet la suppression de neurones dans n'importe quelle couche cachée.

2. MAJORATION DE LA DÉRIVE DE L'ERREUR

Dans cette partie, comme pour la première, on conserve un réseau à une couche cachée et un seul neurone de sortie, possèdant n neurones cachés. L'ensemble d'apprentissage comprend N exemples.

On choisit un seuil positif ρ pour appliquer l'algorithme. Le but ici est de majorer la dérive de l'erreur quadratique moyenne E sur l'ensemble d'apprentissage. On part du système (1) et de la matrice A dont les p lignes correspondent aux p vecteurs orthonormés du sous-espace associé aux valeurs propres inférieures à ρ. Le fait de considérer des valeurs propres non nulles mais inférieures au seuil fixé modifie le système (1) de la partie 1. Le système (1) qui était exact pour tous les exemples k d'apprentissage devient l'égalité fonctionnelle suivante :

$$AX = b + \eta \qquad (2)$$

Le terme correcteur η dépend des exemples et n'est connu qu'en moyenne quadratique.
On montre (cf. annexe 2) que la dérive ΔE de l'erreur quadratique moyenne sur l'ensemble d'apprentissage due à la présence du terme correcteur peut être majorée ainsi :

$$|\Delta E| \leq K(n,p,A)\sqrt{\rho}\ \max_i |W_i|$$

où K est une fonction du réseau, mais aussi des exemples (via la matrice A).

On peut montrer que sous réserve d'un choix judicieux des p neurones supprimés, il existe une majoration ne dépendant pas de la matrice A mais seulement du nombre de neurones cachés (cf. annexe 2) :

$$|\Delta E| \leq R(n)\sqrt{\rho}\ \max_i |W_i| \qquad \text{avec } R(n) = n \times n!$$

La fonction R(n) obtenue est grossière et ne permet de déduire que des propriétés qualitatives. En dehors de cette fonction de la taille de la couche à élaguer, on remarque la dépendance en racine carrée du paramètre, homogène à une variance, ainsi que l'influence des amplitudes des poids entre la couche élaguée et la suivante. On en déduit les règles heuristiques suivantes :

- plus le nombre de neurones cachés est important, plus le seuil doit être faible pour garantir une dérive donnée,
- plus les poids sont faibles, plus on peut augmenter le seuil, donc être efficace en élagage, à dérive d'erreur constante.

Comme l'expérience montre que les réseaux fortement surdimensionnés ne nécessitent pas des poids de grandes amplitudes pour converger (du moins si on initialise avec des poids faibles et qu'on utilise un petit pas de gradient), ceux-ci seront ensuite facilement élagués.

[1] Les combinaisons linéaire des états des neurones cachés n'étant dans ce cas pas rigoureusement constantes sur la base d'apprentissage, les constantes b_i du système (1) doivent être prises égales chacune à la moyenne sur la base d'exemples de la combinaison linéaire définie par A.

3. ILLUSTRATIONS EXPÉRIMENTALES

Dans cette partie, la méthode présentée est illustrée sur deux problèmes, le premier est l'approximation de la fonction cosinus sur [0;2π], le second est son application sur un cas industriel qui concerne la reconnaissance de modulations numériques de signaux interceptés [5].

3.1. Approximation de la fonction cosinus sur [0; 2π]

Sur une base équirépartie de 100 exemples entre 0 et 2π, des apprentissages par rétropropagation du gradient ont été effectués sur des architectures croissantes de 0 à 50 neurones cachés. La convergence était considérée comme acquise quand l'erreur quadratique moyenne descendait sous 1%. Avec un nombre de neurones cachés inférieur à 4, la convergence n'a jamais été obtenue. A partir de 4, l'algorithme a toujours convergé. On peut en déduire que l'architecture minimale requise pour ce problème est 4 neurones cachés. L'algorithme d'élagage a été appliqué sur toutes les architectures ayant convergé en contraignant l'erreur quadratique moyenne à ne pas dépasser 2%. La valeur de ρ utilisée fut de 10^{-5} en général, même si la procédure fut rénouvelée avec une valeur de 10^{-4} pour 2 des réseaux étudiés. Les résultats sont présentés sur la figure 1.

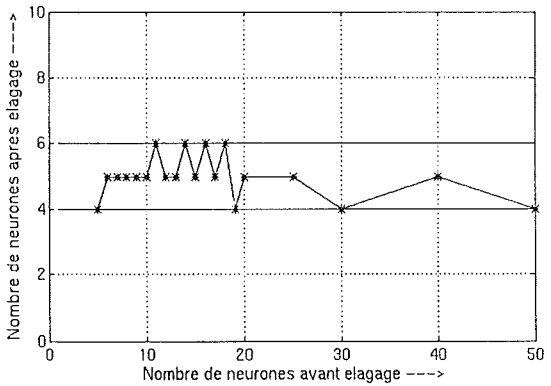

Figure 1 : ce graphique montre la constance de l'architecture après élagage, même quand le nombre de neurones initiaux est très important. Toutes les architectures finales sont comprises entre 4 et 6 neurones cachés.

On constate que notre technique d'élagage permet d'approcher assez bien l'architecture minimale. De plus la méthode se montre d'une grande robustesse vis-à-vis de l'architecture de départ puisque les réseaux les plus importants sont quasiment réduits à l'architecture minimale.

3.2. La reconnaissance de modulations numériques

Le besoin de surveiller l'activité radio-électrique et de reconnaître des signaux interceptés est très important dans le domaine militaire et fait partie de ce qu'on appelle le SIGINT (SIGnal INTelligence). La détermination du type de modulation des signaux numériques interceptés est un problème de classification indispensable à la compréhension de l'information véhiculée. Un simulateur a généré des signaux réalistes de 10 types de modulations numériques [5] : 2FSK, 4FSK, 8FSK, 2PSK, 4PSK, 8PSK, 16QAM, 64QAM, OQPSK et MSK. La base totale obtenue contient 3000 signaux indépendants, 300 par type de modulation. Parmi ces 300 signaux par modulation, 100 ont été générés avec un rapport signal/bruit entre 0 et 15 dB, 100 entre 15 et 35 dB, et 100 autres entre 35 et 50 dB. La base est ainsi équilibrée en modulations et rapports

signal/bruit. Après un prétraitement sur les signaux échantillonnés consistant à extraire les 4 premiers moments statistiques sur la phase, l'amplitude et la fréquence instantanées et à normaliser en puissance, les exemples possèdent 11 entrées et 10 classes codées suivant un codage "grand-mère". La base a été découpée en 1500 exemples d'apprentissage et 1500 autres exemples de test.

Une méthode fondée sur la hiérarchisation de réseaux développée dans [5] permet de résoudre un tel problème à 10 classes en le découpant en sous-problèmes. Chaque sous-problème est résolu par un réseau à une couche cachée et ces différents réseaux sont reliés entre eux par une couche cachée exécutant des portes "ET". Le réseau formé possède finalement 2 couches cachées (partiellement connectées) dont seule la première est à dimensionner. De nombreux essais avec la rétropropagation du gradient ont montré que l'architecture optimale pour le réseau global se situait aux alentours de 20 neurones cachés sur la première couche. Une automatisation complète de l'apprentissage a été mise en place avec un apprentissage par la méthode de Ash ("Dynamic Node Creation", voir [1]) pour chaque sous-réseau suivie de l'algorithme d'élagage de neurones. La méthode de Ash consiste à rajouter des neurones cachés dès que la courbe d'erreur se stabilise sur un plateau. Elle fournit une architecture généralement surdimensionnée (ici, 58 neurones cachés au lieu de 20) pour un taux de reconnaissance à peine inférieur au taux optimal. L'utilisation de l'algorithme d'élagage permet de diminuer l'architecture en conservant le taux d'apprentissage.

Architecture optimale, composée de 4 sous-réseaux de tailles "optimisées" par essais et erreurs : 11-20-13-10.
taux d'apprentissage : 89.9% taux de généralisation : 89.8%

Architecture obtenue en utilisant pour chaque sous-réseau la "Dynamic Node Creation" : 11-58-13-10.
taux d'apprentissage : 89.9% taux de généralisation : 88.5%

Architecture obtenue en appliquant, pour chaque sous-réseau, la "Dynamic Node Creation" suivie d'un élagage de la couche cachée : 11-27-13-10
taux d'apprentissage : 89.9% taux de généralisation : 88.5%

Le dernier réseau, obtenu automatiquement, est assez proche de l'architecture optimale, tant au niveau du nombre de neurones (par rapport au 2° réseau, notre technique a supprimé 31 des 38 neurones "de trop"), qu'au niveau du taux de reconnaissance. Avec les deux outils de construction et d'élagage, on peut donc réaliser un apprentissage de qualité de façon purement algorithmique.

4. CONCLUSIONS

La méthode d'élagage de neurones présentée dans cet article possède les avantages suivants :

- elle est fondée sur les notions mathématiques exactes de minimalité et d'irréductibilité de Sussman [9] qui a énoncé les conditions nécessaires et suffisantes pour qu'un réseau de neurones ait l'architecture minimale,
- il existe une majoration explicite de la dérive de l'erreur d'apprentissage après la phase d'élagage en fonction du paramètre libre de l'algorithme,
- la méthode ne nécessite aucun ré-apprentissage postérieur à la phase d'élagage, ce qui rend son utilisation particulièrement rapide.

Les deux applications présentées illustrent son efficacité pratique, en approximation et en classification. Son utilisation en collaboration avec une méthode constructive permet une plus grande automatisation de l'apprentissage, même sur le problème reconnu comme difficile ([3] et [4]) de la reconnaissance de modulations numeriques de signaux inconnus.

On remarque, après utilisation de l'algorithme d'élagage, une augmentation systématique de la valeur absolue des poids du réseau. Il serait intéressant d'étudier

pourquoi un nombre de neurone inférieur nécessite des poids plus importants pour réaliser la même erreur d'apprentissage. Des majorations plus fines de la dérive de l'erreur doivent permettre de répondre partiellement à cette question, très probablement liée à la notion de capacité d'un réseau introduite par Vapnik [10].

5. BIBLIOGRAPHIE

[1] T. Ash, *Dynamic Node Creation in Backpropagation Networks,* Connection Science, Vol. 1, No. 4, p. 366-375, 1989.

[2] S. E. Fahlman, C. Lebiere, *The Cascade Correlation Architecture, School of Computer Science,* C.M.U., Pittsburgh, 1990.

[3] S. Ghani, R. Lamontagne, *Neural Networks applied to the classification of spectral features for automatic modulation recognition, MILCOM'93,*1993.

[4] R. Lamontagne, *Modulation recognition, an overview, Technical note 91-3,* Defence Research Establishment of Ottawa, 1991.

[5] C. Louis, P. Sehier, *Automatic Modulation Recognition with a Hierarchical Neural Network,* MILCOM, 1994.

[6] C. Louis, F. Moutarde, *Analyse et Construction des Réseaux de Neurones Formels, Rapport intermédiaire sur l'étude de nouveaux algorithmes de construction de réseaux,* Alcatel Alsthom Recherche, 1993.

[7] R. Reed, *Pruning Algorithms-A Survey,* IEEE Transactions on Neural Networks, Vol. 4, N° 5, September 93.

[8] J. Siestma, R.J.F. Dow, *Creating Artificial Neural Networks that Generalize,* Neural Networks, Vol. 4, n° 1, pp. 67-69, 1991.

[9] H. Sussman, *Uniqueness of the Weights for Minimal Feedforward Nets with a Given Input-Output Map,* Neural Networks, Vol.5, n° 4, pp. 589-593, 1992.

[10] V. Vapnik, *Estimation of dependencies based on empirical data,* Springer-Verlag, 1982.

6. ANNEXES

Annexe 1 : démonstration du théorème de la partie 1.

On reprend ici les notations du §1.3.

a/ Sens direct

Supposons qu'il existe un vecteur $a={}^t(a_1 \ \dots \ a_n)$ non nul et un réel β tels que $\sum_{i=1}^{n} a_i \xi_i = \beta 1_N$. Montrons alors que le vecteur a appartient au noyau de T.

Calculons l'image de a par T : $Ta={}^t\left(\sum_{j=1}^{n} t_{1j}a_j \ \dots \ \sum_{j=1}^{n} t_{ij}a_j \ \dots \ \sum_{j=1}^{n} t_{nj}a_j\right)$

Or, en utilisant la définition de t_{ij} (voir §1.3), et l'hypothèse, on obtient pour tout i :

$$\sum_{j=1}^{n} t_{ij}a_j = \sum_{j=1}^{n} \left(\xi_i - \overline{x_i}1_N\right)\bullet\left(a_j\xi_j - a_j\overline{x_j}1_N\right) = \left(\xi_i - \overline{x_i}1_N\right)\bullet\left(\sum_{j=1}^{n} a_j\xi_j - \sum_{j=1}^{n} a_j\overline{x_j}1_N\right)$$

$$= \left(\xi_i - \overline{x_i}1_N\right)\bullet\left(\beta - \sum_{j=1}^{n} a_j\overline{x_j}\right)1_N = \left(\beta - \sum_{j=1}^{n} a_j\overline{x_j}\right)\sum_{k=1}^{N}\left(x_i^k - \overline{x_i}\right) = 0$$

Donc $Ta=0$, et a appartient bien au noyau de T.

b/ Réciproque

Inversement, T étant symétrique positive, il existe une base orthonormée $[E_1, \dots, E_n]$ de R^n dans laquelle T est diagonale. On pose $P=[E_1, \dots, E_n]$, la matrice unitaire nxn de changement de base et on note $T'=P^{-1}TP$ la matrice diagonale obtenue. La base étant orthonormée, on a $P^{-1} = {}^tP$. On note X^k le vecteur des sorties des neurones pour l'exemple k et X'^k son expression dans la nouvelle base, on a $X'^k = P^{-1}X^k = {}^tPX^k$ pour tous les k.

Par définition, chacun des vecteurs $E_1, \dots, E_n$ est un vecteur propre de T, et le nombre de 0 sur la diagonale de T' est la dimension du noyau de T. Soit un des E_i correspondant à une valeur propre nulle; alors la ième colonne de T' est nulle. Or un calcul immédiat permet de montrer que T' est la matrice de covariance des "états transformés" X'. Le fait que la i-ème colonne de T' soit nulle indique donc que $x_i'^k$ est constant sur tous les exemples k. Or d'après les formules de changement de base $x_i'^k = E_i \bullet X^k$. On voit donc que les coordonnées des vecteurs propres E_i du noyau de T sont bien les coefficients de dépendances linéaires constantes sur la base entre les sorties des neurones cachés.

Annexe 2 : majoration de la dérive de l'erreur

On reprend ici les notations introduites dans les parties 1. et 2. de l'article.

On note L_i- la ième ligne de la matrice A.

On définit n fonctions différentes η_1 à η_n qui déterminent l'écart entre le système (1) (qui suppose $\rho = 0$) et les valeurs propres non nulles mais inférieures au seuil. On pose donc :

$$\forall i \in \langle 1, p\rangle, \forall k \in \langle 1, N\rangle, \ \eta_i^k = L_iX^k - b_i \ \text{ où } \ b_i = \frac{1}{N}\sum_{k=1}^{N} L_iX^k$$

Pour tout i entre 1 et p, les fonctions η_i sont de variance inférieure à ρ par choix de p. On a donc :

$$\forall i \in \langle 1, p\rangle, \ \frac{1}{N}\sum_{j=1}^{N}\left|\eta_i^j\right|^2 = \frac{1}{N}\sum_{j=1}^{N}\left|L_iX^j - b_i\right|^2 < \rho$$

On pose $\tilde{X}=\begin{bmatrix} x_1 \\ . \\ x_p \end{bmatrix}$, $\tilde{Y}=\begin{bmatrix} x_{p+1} \\ . \\ x_n \end{bmatrix}$, $b=\begin{bmatrix} b_1 \\ . \\ b_p \end{bmatrix}$, $\eta=\begin{bmatrix} \eta_1 \\ . \\ \eta_p \end{bmatrix}$, $A=\left[\tilde{A}_1\tilde{A}_2\right]$, $X'=\left[\tilde{X}^t\tilde{Y}^t\right]$,

On peut alors réécrire le système (2) (voir §2) sous la forme $\tilde{A}_1\tilde{X} = -\tilde{A}_2\tilde{Y} + b + \eta$

et le résoudre ainsi : $\tilde{X} = -\tilde{A}_1^{-1}\tilde{A}_2\tilde{Y} + \tilde{A}_1^{-1}b + \tilde{A}_1^{-1}\eta$

On pose $\tilde{A}_1 = (\alpha_{ij})^{-1}$. Calculons la dérive de l'erreur (causée par le choix d'un ρ non nul impliquant que les fonctions η ne sont pas nulles) en fonction des coefficients α_{ij}. Si on fixe un neurone d'indice $i \in \langle 1, p\rangle$ et un exemple d'apprentissage indicé par $k \in \langle 1, N\rangle$, on peut calculer la dérive d'erreur correspondant à la suppression du neurone d'indice i sur l'exemple k, δe^k. Si z^k est la sortie avant élagage et z'^k la valeur de la sortie après élagage à un seuil de ρ, on peut écrire :

$\left|\delta e_i^k\right| = \left|z^k - z'^k\right|$ avec z=tanh(y) et z'=tanh(y') donc $\left|\delta e_i^k\right| \le \left|y^k - y'^k\right| \le \left|W_i\left(X^k - X'^k\right)\right|$

et finalement $\left|\delta e_i^k\right| \le |W_i|\left|\tilde{A}_1^{-1}\eta\right| \le |W_i|\left|\sum_{j=1}^{p}\alpha_{ij}\eta_j^k\right|$

Si on somme sur les p neurones cachés élagués, on obtient Δe^k qui vérifie :

$$\left|\Delta e^k\right| \le \sum_{i=1}^{p}\left|\sum_{j=1}^{p}\alpha_{ij}\eta_j^k\right||W_i|$$

Enfin la dérive de l'erreur quadratique totale est donc :

$$\left|\Delta E\right|^2 \le \frac{1}{N}\sum_{k=1}^{N}\left|\Delta e^k\right|^2$$

Or d'après l'inégalité de Cauchy-Schwarz,

$$\left|\Delta e^k\right|^2 \le \left(\sum_{i=1}^{p}\left|\sum_{j=1}^{p}\alpha_{ij}\eta_j^k\right||W_i|\right)^2 \le \left(\sum_{i=1}^{p}|W_i|^2\right)\left(\sum_{i=1}^{p}\left|\sum_{j=1}^{p}\alpha_{ij}\eta_j^k\right|^2\right) \text{ et } \left|\sum_{j=1}^{p}\alpha_{ij}\eta_j^k\right|^2 \le \left(\sum_{j=1}^{p}|\alpha_{ij}|^2\right)\left(\sum_{j=1}^{p}\left|\eta_j^k\right|^2\right)$$

d'où, en utilisant la majoration de la variance des η_i :

$$\boxed{\left|\Delta E\right|^2 \le p\left\|\tilde{A}_1^{-1}\right\|^2\|W\|^2\rho}$$

En appliquant le résultat de l'annexe 3 à la matrice $\tilde{A}_1^{-1}$, on obtient finalement la majoration suivante:

$$\boxed{\left|\Delta E\right| \le n.(n!)\sqrt{\rho}\,\max_i|W_i|}$$

Annexe 3 : minoration d'un déterminant

On considère $P=(a_{ij})$ une matrice carré d'ordre n et unitaire, ie. qui vérifie $P^tP=I_n$. Son déterminant est de module 1.
Un résultat classique de géométrie permet de minorer le plus maximum des déterminants pxp extrait de P par l'expression

$$\sqrt{\frac{p!\,(n-p)!}{n!}}$$

Soit A une matrice extraite de P dont le déterminant vaut D. A est inversible et

$A^{-1} = \frac{1}{\det A}{}^t(comA)$ où comA est la matrice d'ordre p-1 des cofacteurs de A.

Par définition de comA, comme les coefficients de A sont de modules inférieurs à 1, on peut majorer ceux de com A par (p-1)!

d'où $\max_{i,j}\left|A_{ij}^{-1}\right| <= \sqrt{\frac{n!\,(p-1)!}{p\,(n-p)!}}$ et donc $\left\|A^{-1}\right\|^2 <= (n!)^2$