Une nouvelle approche pour les réseaux de neurones : la représentation scalaire distribuée

A New Approach for Neural Networks : the Scalar Distributed Representation

Gilles BUREL
Thomson CSF
Laboratoires Electroniques de Rennes
avenue de Belle Fontaine
F-35510 Cesson-Sévigné

Gilles Burel est ingénieur aux Laboratoires Électroniques de Rennes de Thomson-CSF depuis 1988. Il a obtenu le diplôme d'ingénieur de l'École Supérieure d'Électricité en 1988, et le doctorat de l'Université de Brest en 1991. Ses travaux de recherche portent actuellement sur l'apprentissage automatique, la reconnaissance de formes, et la robotique. Il est l'auteur de plusieurs articles et brevets relatifs à ces domaines techniques, et assure actuellement un cours de Traitement d'Images à l'Université de Brest.

RÉSUMÉ

Les réseaux de neurones sont actuellement d'usage courant en traitement du signal et de l'image. Nous proposons un nouveau modèle de neurone qui utilise un codage particulier de sa sortie que nous nommerons « Représentation Scalaire Distribuée » (RSD). Cette représentation repose sur l'idée de représenter la sortie d'un neurone par une fonction et non par un scalaire. Nous montrons que la RSD induit un comportement non linéaire des connexions entre neurones. La RSD est décrite dans toute sa généralité, puis particularisée pour sa mise en œuvre pratique. Nous considérons notamment la mise en œuvre de la RSD dans un réseau de neurones de type Perceptron Multi-Couches, et nous proposons un algorithme d'apprentissage. Enfin, nous validons le modèle sur deux applications : la réduction de dimensionnalité et la prédiction. Dans les deux cas, un gain important par rapport au modèle classique est obtenu.

MOTS CLÉS

Réseaux de neurones, perceptron multi-couches, apprentissage, problèmes non linéaires, réduction de dimensionnalité, compression d'images.

ABSTRACT

Nowadays, neural networks are largely used in signal and image processing. We propose a new neuron model that uses a special coding for its output, which we will call « Scalar Distributed Representation » (SDR). This representation is based on the idea of representing the neuron's output by a function, and not only by a scalar. We show that SDR produces a non-linear behaviour of connections between neurons. The SDR is described in general and then adapted on practical considerations. We consider the use of SDR for a Multi-Layer Perceptror and we propose a learning algorithm. Finally, we validate the model on two applications : dimensionality reduction, and prediction. In both cases, an important benefit is obtained over the classical model.

KEY WORDS

Neural Networks, Multi-Layer Perceptron, Machine Learning, Non-Linear Problems, Dimensionality Reduction, Image Compression.

1. Introduction

Les réseaux de neurones sont des modèles de traitement de l'information qui s'inspirent plus ou moins directement d'observations relatives au fonctionnement des systèmes nerveux biologiques. Ce domaine de recherche connaît depuis quelques années un regain d'intérêt, et les réseaux de neurones sont actuellement d'usage courant en traitement du signal et de l'image. Pour une introduction aux techniques neuronales, on pourra consulter par exemple [7] et [8].

Beaucoup de modèles neuronaux reposent sur un modèle de neurone constitué d'un sommateur pondéré suivi d'une

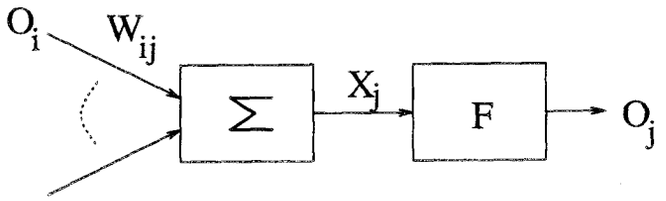

Figure 1. — Modèle classique du neurone.

fonction non linéaire (*fig.* 1). C'est le cas notamment du modèle du Perceptron Multi-Couches (PMC) [10], qui a connu un développement industriel et universitaire important.

Dans la suite de l'exposé nous utiliserons les notations suivantes :

O_j = état de neurone j (valeur de sortie).
F = fonction de transition du neurone.
W_{ij} = coefficient de pondération associé à la connexion ij.
X_j = potentiel du neurone (résultat de la somme pondérée).
pred(j) = ensemble des prédécesseurs du neurone j.
succ(j) = ensemble des successeurs du neurone j.

Conformément à ces notations, nous avons :

$$X_j = \sum_{i \in \text{pred}(j)} W_{ij}\, O_i$$

$$O_j = F(X_j)\,.$$

Les poids W_{ij} constituent des gains adaptatifs, qui sont réglés par un algorithme d'apprentissage.

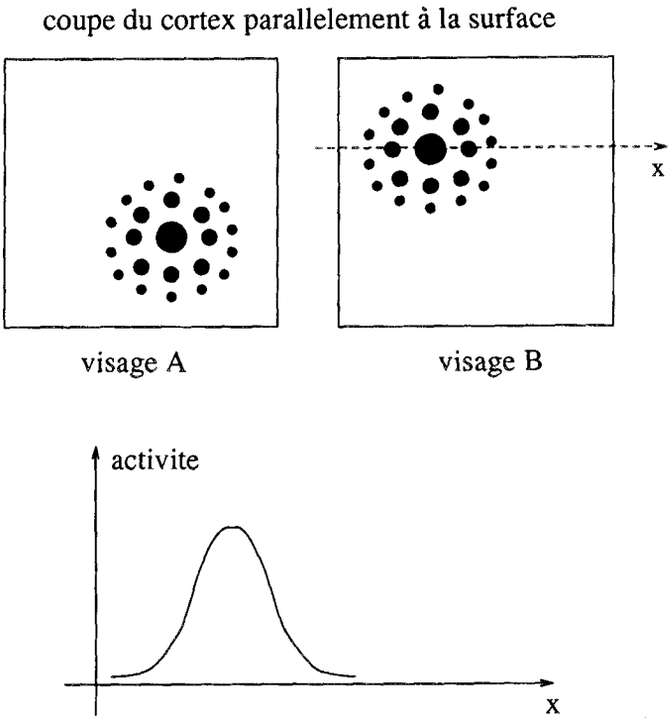

Figure 2. — Réponse à différentes stimulations dans une couche de neurones (ici, la présentation de visages).

Nous proposons ici un nouveau modèle de neurones qui accroît fortement les capacités du réseau à traiter des problèmes non linéaires. Ce modèle est une généralisation de l'opérateur « neurone ». L'idée consiste à représenter la sortie du neurone non pas par un scalaire, mais par une fonction. Pour nommer ce passage de l'espace des scalaires à l'espace des fonctions, nous parlerons de « Représentation Scalaire Distribuée ».

Dans le paragraphe 2, nous présentons le modèle RSD, et nous étudions le comportement d'une connexion entre deux neurones de ce type. Dans le paragraphe 3, nous montrons comment ce modèle de neurone peut être mis en œuvre dans un réseau de type Perceptron Multi-Couches, et nous proposons un algorithme d'apprentissage. Nous montrons également sur un cas simple comment un Perceptron Multi-Couches utilisant la RSD exploite ses degrés de liberté par rapport à un PMC classique. Enfin, le paragraphe 4 présente des résultats expérimentaux et des comparaisons entre les performances du modèle RSD et celles du modèle classique. Les applications considérées sont la réduction de dimensionnalité (dans un cas non linéaire) et la prédiction (prédiction d'index dans un cadre de compression d'images).

2. Le modèle proposé

2.1. JUSTIFICATION DU MODÈLE

Les observations biologiques montrent qu'en réponse à une stimulation sensorielle (par exemple la présentation d'un visage), ce n'est pas un neurone particulier qui s'active dans une couche donnée de neurones, mais toute une population de neurones. Selon la stimulation présentée, la zone d'activation se déplace (*fig.* 2). Kohonen [5] a étudié ce type de phénomène et a montré qu'il peut s'expliquer par l'existence de connexions latérales à l'intérieur d'une couche de neurones. Un neurone a tendance à exciter ses voisins et à inhiber les autres neurones. Comme l'a montré Kohonen, l'effet de ces connexions est de corréler fortement la réponse de neurones voisins.

Dans le cadre d'une structure en couche, vu d'une certaine couche, l'effet de la corrélation des réponses de neurones voisins est de créer une représentation largement redondante des informations provenant de la couche précédente. Il est connu qu'une représentation redondante de l'information est généralement plus aisée à exploiter qu'une représentation compacte.

Une représentation fortement redondante de l'information en sortie d'un ensemble de neurones peut être obtenue de différentes façons :

- Pour les réseaux organisés en couches, en établissant des connexions latérales entre neurones.
- Pour les réseaux dont l'algorithme d'apprentissage consiste à minimiser une fonction d'erreur, en modifiant la fonction à minimiser de manière à forcer des neurones voisins à réagir de manière similaire.

Ces deux approches sont toutefois coûteuses (nombre de neurones, nombre de connexions, complexité algorithmi-

que). La mise au point d'un algorithme d'apprentissage pour réseau à connexions latérales pose des difficultés théoriques dues à la réinjection d'information provenant des sorties d'une couche en entrée de la même couche. Une approche basée sur une modification de la fonction d'erreur a été proposée dans [11]. Nous reviendrons sur cette approche dans le paragraphe concernant la réduction de dimensionnalité. La modification de la fonction d'erreur peut poser des problèmes d'interprétation : en effet, pour beaucoup d'applications dans lesquelles les réseaux de neurones sont utilisés, il existe une interprétation physique de la fonction d'erreur. Sa modification peut donc être gênante. De plus, la fonction d'erreur modifiée semble augmenter fortement la probabilité de piégeage dans un minimum local, ainsi qu'il est mentionné dans [11].

Nous proposons ici une approche différente qui repose sur l'idée suivante : comme l'objectif est de créer une représentation redondante de l'information en sortie d'un ensemble de neurones, nous proposons de modifier le modèle du neurone lui-même pour qu'une telle représentation redondante soit inhérente à sa structure. Pour cela, nous proposons de remplacer la traditionnelle sortie scalaire d'un neurone par une fonction.

Nous proposons plus loin, dans le contexte d'une structure de Perceptron Multi-Couches, un algorithme d'apprentissage adapté au modèle de neurone proposé. L'examen des équations de l'apprentissage montre que le coût en nombre d'opérations reste (comme dans le cas de l'algorithme classique de rétropropagation) proportionnel au nombre de poids dans le réseau. Il en est de même dans la phase de relaxation. Si l'on souhaite comparer un Perceptron Multi-Couches classique avec le modèle proposé au niveau du coût en nombre d'opérations, il suffit donc de comparer le nombre de poids.

2.2. LA REPRÉSENTATION SCALAIRE DISTRIBUÉE

Pour présenter la notion de Représentation Scalaire Distribuée, nous noterons $\mathscr{R}$ l'ensemble des réels, $\mathscr{F}$ l'ensemble des applications de $\mathscr{R}$ dans $\mathscr{R}$, et $\mathscr{D}$ l'ensemble des *injections* de $\mathscr{R}$ dans $\mathscr{F}$. Une injection d de $\mathscr{D}$ sera nommée « fonction de distribution » :

$$d : \mathscr{R} \to \mathscr{F}$$
$$x \to f_x .$$

Cette fonction de distribution étant injective, elle n'introduit pas de perte d'information. La fonction f_x sera nommée « Représentation Scalaire Distribuée » de x.

Nous considérerons à présent l'injection d_h, qui à tout réel x associe le résultat du filtrage de l'impulsion δ_x (impulsion de Dirac centrée en x) par une fonction paire et apériodique h :

$$d_h : \mathscr{R} \to \mathscr{F}$$
$$x \to f_x = \delta_x * h .$$

Dans ce cas, on a $f_x(y) = h(y - x) = h(x - y)$. On vérifiera aisément que l'apériodicité de h est une condition nécessaire et suffisante pour que d_h soit injective. Nous verrons plus loin qu'il est souhaitable pour l'efficacité de l'apprentissage que h soit une fonction de lissage.

L'opération de distribution est entièrement déterminée par la connaissance de la fonction de filtrage h. En pratique, nous prendrons pour h une fonction d'allure gaussienne. La figure 3 illustre la RSD des scalaires $x = 0$, $x = 0.3$ et $x = 0.7$, lorsque la fonction h est une gaussienne.

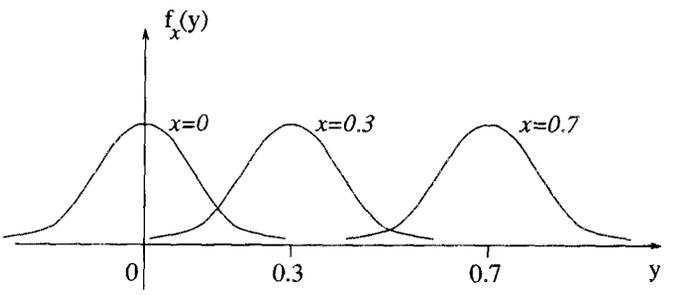

Figure 3. — Représentation Scalaire Distribuée.

Le modèle du neurone est fourni figure 4. Il reçoit en entrée les fonctions f_{x_i} provenant des neurones i de la couche précédente, et calcule une somme pondérée de ces fonctions :

$$X_j = \sum_{i \in \text{pred}(j)} \int_{-\infty}^{+\infty} W_{ij}(u)\, f_{x_i}(u)\, du$$

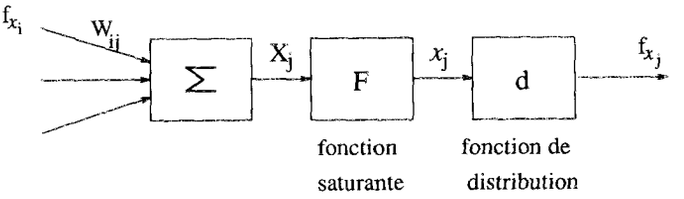

Figure 4. — Modèle du neurone (en continu).

W_{ij} est un continuum de poids caractérisant la connexion ij. Le résultat de cette somme pondérée passe dans une fonction saturante F dont l'intérêt apparaîtra plus tard. Au résultat $x_j = F(X_j)$, on applique la fonction de distribution d_h, afin d'obtenir la fonction de sortie du neurone : $f_{x_j} = d_h(x_j) = \delta_{x_j} * h$.

Pour mieux comprendre le fonctionnement du réseau, il est intéressant d'étudier une connexion ij. D'après l'expression de X_j, on peut écrire :

$$X_j = \sum_{i \in \text{pred}(j)} C_{ij}(x_i)$$

avec

$$C_{ij}(x_i) = \int_{-\infty}^{+\infty} W_{ij}(u)\, f_{x_i}(u)\, du$$
$$= \int_{-\infty}^{+\infty} W_{ij}(u)\, h(x_i - u)\, du$$
$$= (h * W_{ij})(x_i) .$$

Ainsi, en regroupant la fonction de distribution du neurone i avec le continuum de poids caractérisant la connexion ij, on peut voir la connexion ij comme une fonction $C_{ij} = h * W_{ij}$, résultat du filtrage du continuum de poids par la fonction génératrice de la RSD. On peut donc décrire une propagation dans le neurone par la succession d'opérations suivantes :

$$X_j = \sum_{i \in \text{pred}(j)} C_{ij}(x_i)$$
$$x_j = F(X_j) .$$

Ainsi, si l'on souhaite garder le modèle classique du neurone, on peut rendre compte de l'effet de la RSD en considérant que la connexion est une fonction adaptative, et non un simple gain adaptatif comme dans le cas du neurone classique.

2.3. ÉCHANTILLONNAGE DE LA RSD

En pratique il n'est pas possible de représenter un continuum de poids. Les poids seront donc représentés par la fonction échantillonnée suivante (n désigne un entier, et δ est la fonction de Dirac) :

$$W^*_{ij}(u) = \sum_{n=-\infty}^{+\infty} W_{ij,n}\, \delta(n-u) .$$

On a alors :

$$\begin{aligned} C_{ij}(x_i) &= (h * W^*_{ij})(x_i) \\ &= \int_{-\infty}^{+\infty} \left(\sum_{n=-\infty}^{+\infty} W_{ij,n}\, \delta(n-u) \right) f_{x_i}(u)\, du \\ &= \sum_{n=-\infty}^{+\infty} W_{ij,n} \int_{-\infty}^{+\infty} \delta(n-u)\, f_{x_i}(u)\, du \\ &= \sum_{n=-\infty}^{+\infty} W_{ij,n}\, f_{x_i}(n) . \end{aligned}$$

Le modèle du neurone en échantillonné est représenté figure 5. On a posé $O_{j,n} = f_{x_j}(n)$.

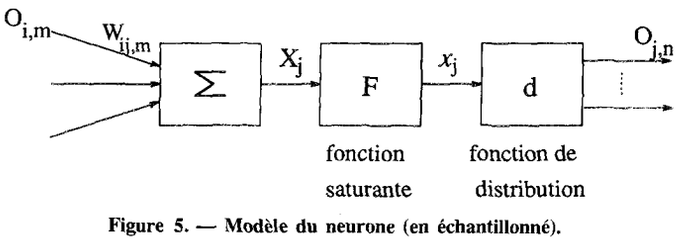

Figure 5. — Modèle du neurone (en échantillonné).

2.4. LIMITATION DE LA RSD A UN INTERVALLE

On fixera à présent à 0 les poids $W_{ij,n}$ pour les valeurs de n extérieures à un intervalle $[n_1, n_2]$. En pratique, les bornes de l'intervalle seront choisies de telle sorte quelles encadrent les valeurs asymptotiques de la fonction F. Ceci se justifie par le fait que le filtre h est en pratique une gaussienne d'écart type assez faible, et les poids d'indice extérieur à $[n_1, n_2]$ ont donc peu d'influence.

Notons que le cas d'un pas d'échantillonnage non constant n'a pas été envisagé car on peut toujours se ramener à un pas unité par un choix convenable de la fonction F.

On a donc à présent :

$$C_{ij}(x_i) = \sum_{n=n_1}^{n_2} W_{ij,n}\, O_{i,n} .$$

3. Mise en œuvre dans un Perceptron Multi-Couches

3.1. LE PERCEPTRON MULTI-COUCHES

Le modèle du Perceptron Multi-Couches a été proposé en 1986 par Rumelhart *et al.* [10]. Le neurone est un sommateur pondéré suivi d'une non-linéarité de type tangente hyperbolique (*fig.* 1). Les neurones sont organisés en couches (*fig.* 6). Il n'y a pas de connexions à l'intérieur

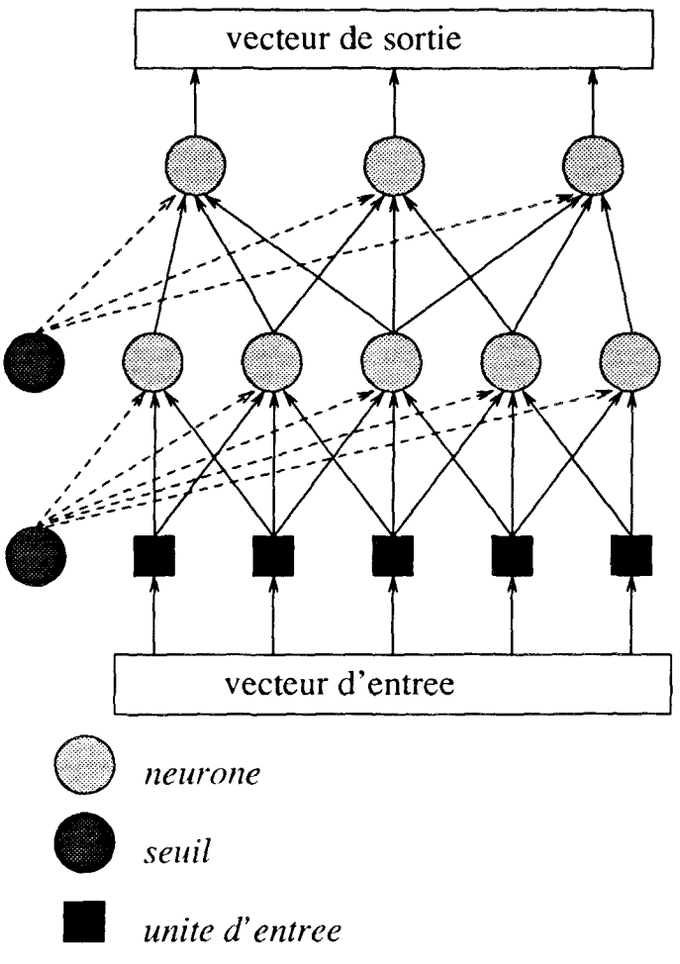

Figure 6. — Perceptron Multi-Couches.

d'une même couche. Chaque neurone reçoit ses entrées de la couche directement inférieure. Un neurone seuil (neurone sans entrées et dont la sortie vaut toujours 1), est souvent disposé sur chaque couche (sauf en sortie), afin d'améliorer les capacités du réseau. En effet, en leur absence, la présentation d'un vecteur nul à l'entrée du réseau conduirait systématiquement à des sorties nulles, quelles que soient les valeurs des poids.

Au départ, les poids W_{ij} des neurones sont initialisés aléatoirement. Lorsque l'on présente un vecteur d'entrée au réseau, il peut calculer par propagation couche après couche les états des neurones de la couche de sortie.

L'apprentissage consiste à adapter les valeurs des poids afin d'obtenir le comportement désiré du réseau. Pour cela, on se crée une base d'exemples. Chaque exemple est constitué d'un vecteur d'entrée et du vecteur de sortie approprié. Un algorithme d'apprentissage connu sous le nom d'algorithme de rétropropagation du gradient a été proposé dans [10]. On se définit une fonction d'erreur (on choisit en général l'erreur quadratique moyenne), et on présente plusieurs fois tous les exemples dans un ordre aléatoire. A chaque présentation d'un exemple, on fait varier les poids W_{ij} dans le sens inverse du gradient de l'erreur. L'algorithme de rétropropagation du gradient est en fait une méthode efficace pour calculer le gradient, qui tire profit de la structure en couches.

3.2. ALGORITHME D'APPRENTISSAGE POUR UN PERCEPTRON MULTI-COUCHES COMPOSÉ DE NEURONES RSD

L'algorithme d'apprentissage que nous proposons est un algorithme de gradient, inspiré de l'algorithme de rétropropagation [10].

On se définit par exemple une erreur quadratique moyenne entre les sorties obtenues (x_j) et les sorties souhaitées (t_j) :

$$e_{QM} = E\{e_Q\}$$

avec

$$e_Q = \frac{1}{2} \sum_{j \in \text{sortie}} (x_j - t_j)^2 .$$

On présente plusieurs fois tous les exemples dans un ordre aléatoire. A chaque présentation d'un exemple, on fait varier les poids $W_{ij,n}$ dans le sens inverse du gradient de l'erreur :

$$\Delta W_{ij,n} = - \alpha \frac{\partial e_{QM}}{\partial W_{ij,n}}$$

α étant une constante positive, on s'assure ainsi une diminution de l'erreur. Nous noterons à présent $g_{ij,n}$ le gradient local de l'erreur quadratique :

$$g_{ij,n} = \frac{\partial e_Q}{\partial W_{ij,n}} .$$

Comme $e_{QM} = E\{e_Q\}$, on peut écrire :

$$\Delta W_{ij,n} = - \alpha E \left\{ \frac{\partial e_Q}{\partial W_{ij,n}} \right\}$$

car la dérivation est un opérateur linéaire. On a donc :

$$\Delta W_{ij,n} = - \alpha E\{g_{ij,n}\} .$$

Une estimation $\bar{g}_{ij,n}$ de $E\{g_{ij,n}\}$ est obtenue par filtrage passe bas de $g_{ij,n}$. Utilisons un indice t qui est incrémenté chaque fois qu'un nouvel exemple est présenté :

$$\bar{g}_{ij,n}(t) = (1 - \beta)\, g_{ij,n}(t) + \beta \bar{g}_{ij,n}(t-1) .$$

En posant $\delta_j = - \frac{\partial e_Q}{\partial X_j}$, le gradient instantané de l'erreur par rapport à un poids s'écrit :

$$\begin{aligned} g_{ij,n} &= \frac{\partial e_Q}{\partial W_{ij,n}} \\ &= \frac{\partial e_Q}{\partial X_j} \frac{\partial X_j}{\partial W_{ij,n}} \\ &= - \delta_j O_{i,n} . \end{aligned}$$

Il nous reste à trouver une méthode pour calculer ce terme δ_j. En utilisant les règles élémentaires du calcul différentiel, on montre que δ_j s'exprime en fonction des δ_k relatifs aux neurones k de la couche suivante :

$$\begin{aligned} \delta_j &= - \frac{\partial e_Q}{\partial X_j} \\ &= - \frac{\partial e_Q}{\partial x_j} \frac{\partial x_j}{\partial X_j} \\ &= - \left(\sum_{n=n_1}^{n_2} \frac{\partial e_Q}{\partial O_{j,n}} \frac{\partial O_{j,n}}{\partial x_j} \right) F'(X_j) \\ &= - \left(\sum_{n=n_1}^{n_2} \left(\sum_{k \in \text{succ}(j)} \frac{\partial e_Q}{\partial X_k} \frac{\partial X_k}{\partial O_{j,n}} \right) \frac{\partial O_{j,n}}{\partial x_j} \right) F'(X_j) \\ &= \left(\sum_{n=n_1}^{n_2} \left(\sum_{k \in \text{succ}(j)} \delta_k W_{jk,n} \right) h'(x_j - n) \right) F'(X_j) . \end{aligned}$$

Pour le cas particulier de la dernière couche, δ_j s'exprime aisément en fonction du type d'erreur choisi. Pour l'erreur quadratique, on a :

$$\begin{aligned} \delta_j &= - \frac{\partial e_Q}{\partial x_j} \frac{\partial x_j}{\partial X_j} \\ &= - (x_j - t_j) F'(X_j) . \end{aligned}$$

3.3. COMPARAISON AVEC LE PMC CLASSIQUE

Nous proposons dans ce paragraphe une comparaison entre un PMC utilisant un neurone classique et un PMC utilisant la RSD, sur un cas suffisamment simple pour permettre une interprétation du comportement des deux types de

réseaux. On compare les réseaux à nombre de poids égaux. On s'intéresse au problème de l'approximation par un réseau d'une fonction de $\mathscr{R}$ dans $\mathscr{R}$.

Considérons le PMC classique de la figure 7. Le nombre de paramètres libres est 7 (nombre de poids). Pour simplifier le raisonnement, la fonction non linéaire du neurone est la fonction :

$$F(X) = -1 \quad \text{si} \quad X < -1$$
$$F(X) = X \quad \text{si} \quad -1 \leqslant X \leqslant +1$$
$$F(X) = +1 \quad \text{si} \quad X > 1 .$$

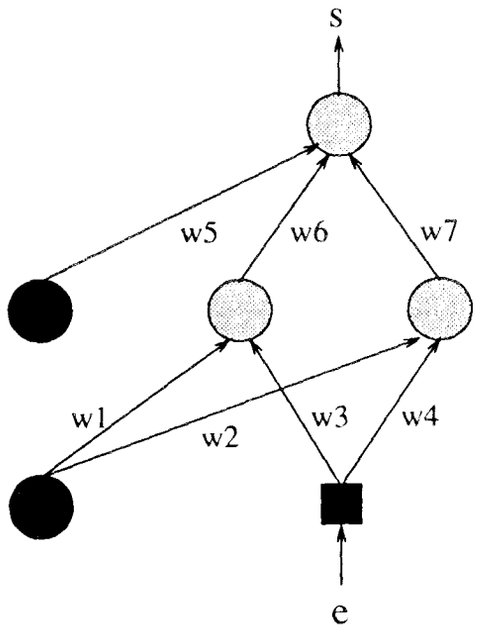

Figure 7. — PMC classique à 7 poids.

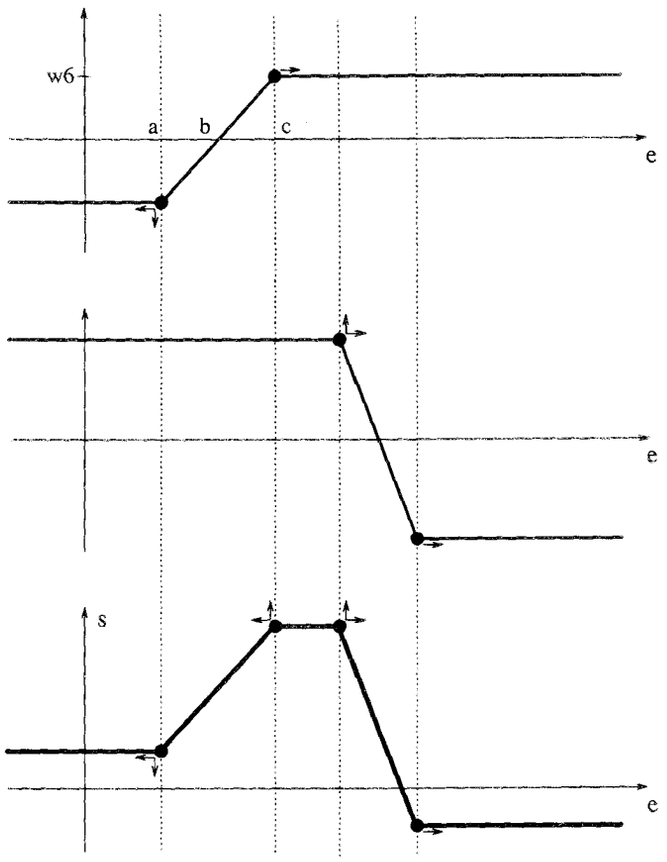

Figure 8. — Courbe entrée-sortie obtenue avec le PMC classique à 7 poids.

Le neurone de sortie est linéaire afin d'éviter une limitation de la valeur de sortie. On a alors :

$$s = w_5 + w_6 F(w_1 + w_3 e) + w_7 F(w_2 + w_4 e) .$$

Si l'on trace la valeur de $w_6 F(w_1 + w_3 e)$ en fonction de e (*fig.* 8, haut), il apparaît immédiatement que cette courbe possède 2 degrés de liberté horizontaux et 1 degré de liberté vertical. On a en effet :

$$a = -\frac{1 + w_1}{w_3} \quad c = \frac{1 - w_1}{w_3} = a + \frac{2}{w_3} \quad b = -\frac{w_1}{w_3} = \frac{a + c}{2} .$$

On obtient bien sûr le même résultat pour $w_7 F(w_2 + w_4 e)$ (*fig.* 8, milieu). La courbe donnant s en fonction de e (*fig.* 8, bas) s'obtient en additionnant les deux courbes précédentes, et en ajoutant la valeur w_5. On a donc finalement 4 degrés de liberté en horizontal et 3 en vertical.

Le réseau RSD de la figure 9 est le réseau le plus simple qu'il soit possible de considérer, car il contient un seul neurone. Ce réseau contient également 7 poids. Pour simplifier le raisonnement, nous prendrons ici comme

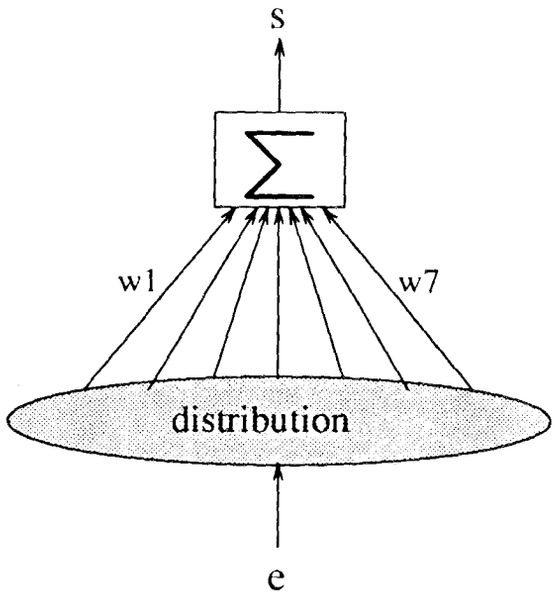

Figure 9. — Réseau RSD à 7 poids.

fonction de distribution une fonction en triangle. On a alors :

$$s = \sum_{i=1}^{7} w_i f_e(i)$$
$$= \sum_{i=1} w_i h(e-i)$$

où

$h(x) = 0$ si $x < -1$ ou $x > +1$
$h(x) = x + 1$ si $-1 \leq x \leq 0$
$h(x) = 1 - x$ si $0 \leq x \leq 1$.

La figure 10 montre, en haut, les différentes fonctions $w_i h(e-i)$, et, en bas, le type de courbe entrée-sortie obtenue. Cette courbe possède donc 7 degrés de liberté en vertical, et aucun degré de liberté en horizontal.

En conséquence, par rapport au modèle classique, la RSD revient à exploiter de manière différente les degrés de liberté. Sur le cas simple présenté ci-dessus, il en résulte que la RSD permettra en général une approximation plus fine d'une fonction, sous réserve d'échantillonner dans la zone correcte (car on ne dispose d'aucun degré de liberté horizontal lorsque le réseau RSD est composé d'un seul neurone). Il faut noter que dans beaucoup d'applications pratiques, il est d'usage de normaliser les entrées d'un prédicteur ou d'un classifieur pour les ramener dans une plage fixée a priori (par exemple entre – 1 et + 1). De plus, au niveau des couches intermédiaires, l'échantillonnage de la RSD est déterminé pour occuper au mieux la plage de variation de l'activation d'un neurone (échantillonnage à pas uniforme entre les bornes de la fonction saturante).

La comparaison proposée ci-dessus permet de montrer certaines différences de fonctionnement entre le modèle classique et le modèle RSD. On s'est toutefois limité à un réseau RSD comportant un seul neurone. En conséquence, seule la comparaison des performances obtenues avec les deux modèles sur des applications importantes permet une véritable évaluation dans des situations réalistes. C'est l'objet de la section suivante.

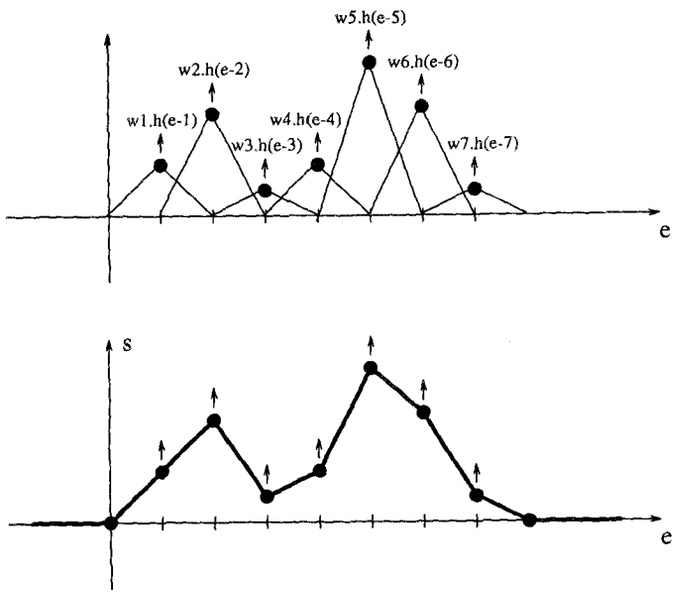

Figure 10. — Courbe entrée-sortie obtenue avec le PMC utilisant la RSD.

4. Applications

4.1. RÉDUCTION DE DIMENSIONNALITÉ

Dans de nombreuses applications de traitement du signal, l'observation est un vecteur appartenant à un espace $\mathscr{E}$ de dimension N. Très souvent, les composantes de ce vecteur ne sont pas indépendantes, et sont en réalité restreintes à une hyper-surface $\mathscr{S}$ de dimension M inférieure à N.

La méthode d'analyse en composantes principales (ACP) permet de trouver la meilleure approximation de $\mathscr{S}$ par un hyperplan (meilleure approximation au sens des moindres carrés).

Lorsque l'hyper-surface $\mathscr{S}$ est fortement courbée, une approximation par un hyperplan n'est pas satisfaisante, et il n'existe actuellement aucune méthode classique pour traiter ce cas (sauf peut-être si l'on connaît a priori une équation paramétrique de $\mathscr{S}$, ce qui n'est généralement pas le cas). Une méthode basée sur une approche neuronale a été proposée dans [11]. Elle permet de traiter des problèmes de réduction de dimensionnalité non linéaires, mais nécessite

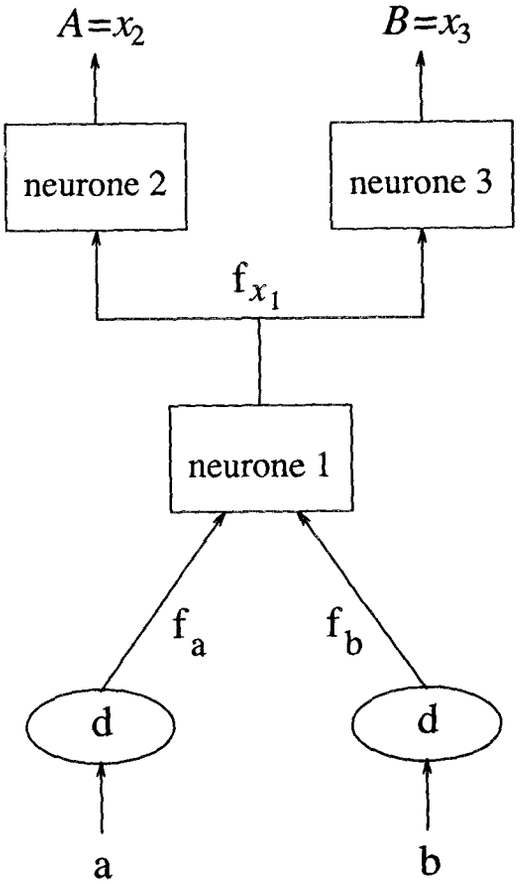

Figure 11. — Réseau de neurones pour réduction de dimensionnalité.

la mise en œuvre d'un grand nombre de neurones, et la modification de la fonction d'erreur à minimiser. En effet, son principe consiste à imposer une représentation fortement redondante de l'information en sortie de chaque couche de neurone en modifiant la fonction d'erreur, de manière à obliger des neurones voisins à avoir des niveaux de sortie voisins. En ce sens, cette méthode s'apparente à la RSD, mais elle est plus coûteuse à mettre en œuvre car la représentation redondante est obtenue sur un ensemble de neurones, alors que la RSD fournit une représentation redondante de l'information au niveau du neurone lui même. De plus, la RSD évite la modification de la fonction d'erreur à minimiser : on peut toujours minimiser une erreur quadratique, comme dans le cas du PMC classique. La fonction d'erreur modifiée semble augmenter fortement le risque de piégeage dans un minimum local, ainsi qu'il est mentionné dans [11].

Notre objectif dans cette section est de montrer que le modèle de réseau à Représentation Scalaire Distribuée est capable de traiter un tel problème de réduction de dimensionnalité, sans nécessiter de connaissance a priori sur la forme de $\mathcal{S}$. Afin de faciliter la visualisation des résultats, nous décrivons des expérimentations menées en dimension 2, mais le modèle est bien entendu capable de traiter des dimensions supérieures.

Les données traitées sont des données synthétiques : il s'agit de vecteurs $\vec{v} = [a, b]^T$, aléatoirement choisis sur un quart de cercle (de rayon 1), selon une loi uniforme en fonction de l'angle. Ces données ont d'abord été traitées avec un réseau de neurones classique en « identity mapping » (2 + 1 + 2 neurones), qui fournit la même solution que l'analyse en composantes principales, à savoir la droite approximant au mieux le quart de cercle (au sens des moindres carrés).

Nous avons ensuite mis en œuvre le modèle RSD (*fig.* 11), avec une fonction génératrice de la RSD gaussienne. L'erreur quadratique est :

$$e_Q = \frac{1}{2} \{(A - a)^2 + (B - a)^2\} .$$

Le tracé des coordonnées de sortie (A, B) fournit une excellente approximation du quart de cercle (*fig.* 12). Comme dans ce réseau, toute l'information a dû passer par x_1, on a bien réalisé une réduction de dimensionnalité.

Le tableau suivant résume les résultats obtenus en fin d'apprentissage (on a noté entre parenthèses le nombre de neurones par couche, hormis les seuils) :

Réseau	e_{QM}
classique (2 + 1 + 2)	0.00389
RSD (2 + 1 + 2)	0.00002

Pour le réseau RSD mis en œuvre, on a :

$$h(u) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{u^2}{2\sigma^2}}$$

et

$$F(u) = \frac{7}{2} (th(u) + 1)$$
$$\sigma = 1$$
$$n_1 = 0$$
$$n_2 = 7 .$$

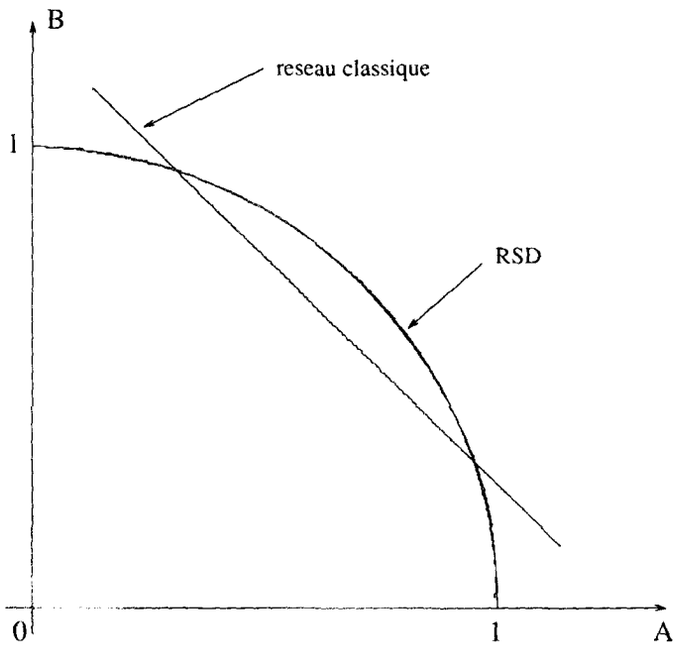

Figure 12. — Réduction de dimensionnalité.

Le choix du facteur $\frac{7}{2}$ conduit à une représentation de la sortie du neurone sur 8 échantillons, puisque le pas d'échantillonnage est systématiquement égal à 1. Comme le montrent les résultats obtenus, cette « résolution » est largement suffisante pour obtenir une bonne approximation du quart de cercle.

Expérimentalement, on constate une mauvaise convergence de l'apprentissage lorsque l'écart type σ est choisi trop faible (inférieur à 0.5). Des problèmes de minima locaux ont été constatés pour ces valeurs. Le choix de σ est donc un compromis :

- Si σ est trop fort (supérieur à 3), les poids sont fortement lissés (voir expression de C_{ij}), et on restreint les capacités du réseau.
- Si σ est trop faible (inférieur à 0,5), le risque de blocage dans un minimum local devient non négligeable.

Une solution pour éviter les 2 problèmes à la fois est de réduire σ en cours d'apprentissage. Pour les expérimentations, nous avons préféré choisir une valeur intermédiaire de l'écart type ($\sigma = 1$). Cette valeur permet d'obtenir un très bon résultat final, sans créer de problèmes de convergence de l'apprentissage.

Signalons également que la convergence est légèrement moins bonne lorsque la Gaussienne est remplacée par un

triangle. Nous n'avons pas recherché les causes théoriques de ce phénomène, mais il pourrait éventuellement être explicable par la variation brutale de la dérivée dans le cas d'une fonction « triangle ».

4.2. PRÉDICTION EN COMPRESSION D'IMAGES

4.2.1. Contexte et idée générale

La compression d'images est un problème qui connaît actuellement une attention soutenue. Pour une revue des techniques habituelles de compression d'images, on pourra consulter par exemple [2] et [4].

Nous avons présenté dans une autre publication [1] une méthode de compression d'images mettant en œuvre l'algorithme de Kohonen [5]. L'algorithme de Kohonen est un algorithme d'auto-organisation qui s'inspire d'un modèle de réseaux de neurones à connexions latérales. L'application de cet algorithme à la compression d'images a également été étudiée dans [6]. Notons dès à présent, car nous allons revenir sur ce point, que ces approches utilisent les capacités de Quantification Vectorielle de l'algorithme de Kohonen, mais n'exploitent pas du tout ses propriétés de conservation de la topologie. L'algorithme de Kohonen est utilisé pour créer un dictionnaire de blocs de référence (par exemple des blocs de 3×3 pixels). La compression consiste à remplacer chaque bloc de l'image par l'index du bloc du dictionnaire qui s'en rapproche le plus. A la reconstruction, on remplace chaque index par le bloc correspondant dans le dictionnaire.

A titre d'illustration, une image de télévision a été comprimée grâce à cette méthode. Un dictionnaire de 256 blocs de référence a été créé en utilisant l'algorithme de Kohonen. Ces 256 blocs sont indexés de 0 à 255. La compression a consisté à remplacer chaque bloc de 3×3 pixels de l'image source par l'index du bloc du dictionnaire qui le représente le mieux (au sens de l'erreur quadratique). Si on représente ces index sous forme de luminance, on obtient une nouvelle image, dite « image des index » (*fig.* 13), dont la taille est réduite d'un facteur 3 dans chaque dimension par rapport à l'image originale. La taille de l'image des index est 224×176.

Figure 13. — Image des index (dictionnaire créé par Kohonen).

Grâce aux propriétés topologiques de l'algorithme de Kohonen, cette image possède toujours une certaine cohérence interne, qu'il serait intéressant d'exploiter pour gagner en taux de compression. Ceci est dû au fait que dans le dictionnaire créé par l'algorithme de Kohonen, si deux blocs de référence ont des index voisins, alors leurs contenus sont voisins. Notons qu'il existe d'autres algorithmes pour créer un dictionnaire, par exemple l'algorithme LBG [9]. L'algorithme LBG est un algorithme itératif de quantification vectorielle [3]. L'algorithme LBG ne conserve pas la topologie. En conséquence, l'image des index obtenue en utilisant un dictionnaire créé par LBG ne possède pas de cohérence interne (*fig.* 14).

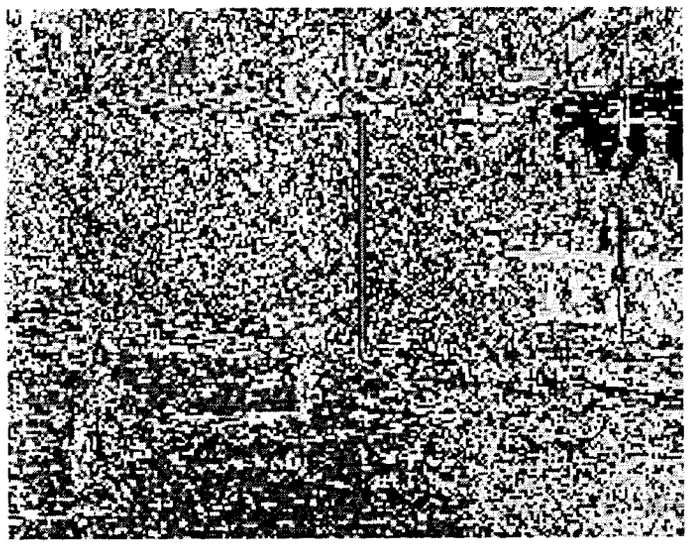

Figure 14. — Image des index (dictionnaire créé par LBG).

L'idée que nous proposons pour améliorer les taux de compression consiste à comprimer à nouveau l'image des index, par exemple en mettant en œuvre un prédicteur. Si la prédiction fonctionne bien (ce qui devrait être possible avec l'image de la figure 13, car elle est relativement cohérente), l'erreur de prédiction sera souvent faible, et son entropie sera donc faible. On peut alors la coder en Huffman, et représenter l'image des index par l'erreur de prédiction codée en Huffman, d'où un gain en taux de compression. Il s'agit d'une compression parfaite de l'image des index, car connaissant l'erreur de prédiction (plus la première ligne de cette image pour l'initialisation) et les coefficients du prédicteur, il est possible de reconstruire parfaitement cette image. Il nous a semblé en effet préférable de ne pas introduire de distorsion due à une compression imparfaite de l'image des index, car cette distorsion risquerait de faire apparaître des défauts importants lors de la reconstruction de l'image source.

4.2.2. Mise en œuvre du modèle RSD comme prédicteur

Le prédicteur reçoit en entrée le voisinage causal $[C_1\ C_2\ C_3\ C_4]^T$ d'un index, et doit fournir en sortie la meilleure estimation de cet index (C).

$$\begin{matrix} C_1 & C_2 & C_3 \\ C_4 & C? & \end{matrix}$$

Le prédicteur proposé est un réseau de neurones qui reçoit en entrée le vecteur $[C_1\ C_2\ C_3\ C_4]^T$ et doit fournir en sortie la meilleure prédiction de l'index C. Nous comparons ci-dessous les performances de différents réseaux : un réseau classique à 2 couches, un réseau classique à 4 couches, et un réseau RSD à 3 couches. Le réseau classique à 2 couches n'est rien d'autre qu'un prédicteur linéaire. Ceci permet d'avoir une référence, car la prédiction linéaire est le mode de prédiction le plus largement utilisé. Pour le réseau classique à 4 couches, les neurones des 2 couches intermédiaires possèdent une non-linéarité en tangente hyperbolique, et le neurone de sortie est linéaire.

Pour le réseau RSD mis en œuvre, on a :

$$h(u) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{u^2}{2\sigma^2}}$$

et

$$F(u) = \frac{15}{2}\,(th(u) + 1)$$

$$\sigma = 2.5$$

$$n_1 = 0$$

$$n_2 = 15\ .$$

Sauf pour le neurone de sortie qui est entièrement linéaire et sans distribution (puisque l'on doit obtenir une valeur scalaire en sortie). Le choix du facteur $\frac{15}{2}$ conduit à une représentation de la sortie du neurone sur 16 échantillons, puisque le pas d'échantillonnage est systématiquement égal à 1.

En entrée du réseau RSD, les valeurs des index sont ramenées entre n_1 et n_2 grâce à une division par une constante adéquate. Le tableau suivant résume les résultats obtenus (erreur quadratique moyenne de prédiction sur les index) avec différents types de réseaux (entre parenthèses, on a indiqué le nombre de neurones sur chaque couche, hormis les seuils). Ces résultats ont été obtenus sur l'image de la figure 13.

Réseau	e_{QM}
classique (4 + 1)	386.6
classique (4 + 12 + 8 + 1)	379.3
RSD (4 + 2 + 1)	**299.5**

En représentant l'image des index par l'erreur de prédiction codée en Huffman (et en tenant compte des octets supplémentaires occupés par les coefficients du prédicteur, la table des codes de Huffman, et la première ligne d'initialisation), on obtient un gain de 17 % sur le taux de compression par rapport à l'algorithme de compression sans prédicteur. Le nombre d'éléments binaires par pixel passe de 0.94 bpp (bits par pixel) à 0.78 bpp. On rappelle que l'image source était codée à 8 bpp (256 niveaux de gris).

D'autres tests effectués avec des images typiquement utilisées comme référence dans le domaine de la compression montrent un gain dû au prédicteur variant de 15 % à 25 % selon le contenu de l'image.

En résumé, nous avons donc proposé un complément des méthodes de compression d'images par l'algorithme de Kohonen [1] [6], qui consiste à effectuer une compression parfaite de l'image des index grâce à la mise en œuvre d'un prédicteur RSD. On peut ainsi gagner en taux de compression, sans modifier l'image des index, et donc sans modifier l'image reconstruite après compression. le gain mentionné de 15 % à 25 % est donc un gain en taux de compression, à qualité d'image strictement égale. La qualité de l'image reconstruite est entièrement déterminée par la qualité du dictionnaire créé par l'algorithme de Kohonen, et les divers exemples examinés dans [6] et [1] montrent une bonne qualité d'image. Les performances du prédicteur n'influent que sur le taux de compression et pas du tout sur la qualité d'image.

Notons enfin que des tentatives de prédiction directement sur des images sources (et non pas des images d'index) n'ont montré aucun gain du réseau RSD ni du réseau classique à 4 couches par rapport à un simple prédicteur linéaire. La mise en œuvre d'un prédicteur RSD ne se justifie donc que lorsque l'opération de prédiction est difficile. Une image d'index est toujours moins cohérente qu'une image source (même si l'algorithme de Kohonen préserve assez bien la topologie), d'où une prédiction effectivement plus difficile.

5. Conclusion

Nous avons proposé un nouveau modèle de neurone et nous avons montré sa compatibilité avec la structure du Perceptron Multi-Couches. Ce modèle permet d'accroître les capacités du réseau à traiter des problèmes non linéaires. Nous avons montré sur deux applications typiques (la réduction de dimensionnalité et la prédiction) l'intérêt de ce modèle par rapport au modèle de neurone classique.

De nombreuses publications ont vanté les performances du Perceptron Multi-Couches lorsqu'il est utilisé comme classifieur. Par contre, on note peu d'applications du Perceptron Multi-Couches comme prédicteur. Le modèle de neurone proposé semble capable de combler, au moins partiellement, cette faiblesse. Les performances d'un réseau utilisant ce type de neurone peuvent être expliquées par le fait que les connexions se comportent alors comme des fonctions adaptatives, et non plus comme de simples gains adaptatifs.

Dans le domaine de la réduction de dimensionnalité, problème qui se rencontre fréquemment en traitement du signal, la RSD pourrait être une réponse aux limitations de l'Analyse en Composantes Principales. Nous avons montré expérimentalement ses capacités lorsque les non-linéarités deviennent trop importantes pour que l'ACP puisse être employée de façon satisfaisante.

Les travaux futurs concernant ce modèle pourraient comporter :

- L'étude de l'influence de la RSD sur la densité de minima locaux, lorsque ce modèle est mis en œuvre dans un Perceptron Multi-Couches entraîné par l'algorithme de rétropropagation du gradient. Le piégeage fréquent dans un minimum local pour de faibles valeurs de l'écart-type de la fonction génératrice de la distribution nous conduisent à penser que la RSD induit probablement, au moins pour certaines valeurs des paramètres, une augmentation de cette densité.
- L'étude de stratégies pour réduire le risque de piégeage dans un minimum local. Nous avons par exemple suggéré la possibilité de faire varier l'écart-type de la fonction génératrice de la distribution en cours d'apprentissage afin de déformer dynamiquement la surface de coût.
- La mise en œuvre du modèle RSD dans le cadre de structures autres que celle du Perceptron Muti-Couches. La RSD n'est en effet pas liée au Perceptron Multi-Couches. En général, sa mise en œuvre dans le cadre d'une nouvelle structure nécessitera l'élaboration d'un algorithme d'apprentissage spécifique.
- L'application du modèle a des problèmes de réduction de dimensionnalité avec des données réelles. L'exemple qui a été donné peut être considéré comme un cas d'école car les données traitées sont des données synthétiques 2D (ceci afin de faciliter la visualisation des résultats). Mais il s'agit toutefois d'un problème difficile du fait de la nécessité d'approximer une courbe sans connaissance a priori sur sa forme.

Manuscrit reçu le 4 novembre 1991.

BIBLIOGRAPHIE

[1] G. Burel, I. Pottier, « Vector Quantization of images using Kohonen algorithm ; Theory and Implementation », *Revue technique Thomson CSF*, Vol. 23, N° 1, mars 1991.

[2] R. Forchheimer, T. Kronander, « Image coding from waveforms to animation », *IEEE Trans. on ASSP*, Vol. 37, N° 12, December 1989.

[3] R. M. Gray, « Vector Quantization », *IEEE ASSP Magazine*, April 1984.

[4] A. K. Jain, « Image data Compression : A Review », *Proceedings of the IEEE*, Vol. 69, N° 3, March 1981.

[5] T. Kohonen, « Self-Organization and Associative Memory », Springer-Verlag, 1984.

[6] E. Le Bail, A. Mitiche, « Quantification Vectorielle d'images par le réseau neuronal de Kohonen », *Traitement du Signal*, Vol. 6, N° 6, 1989.

[7] R. L. Lippmann, « Neural nets for computing », *IEEE Conference on Acoustic and Speech Processing*, 1988.

[8] R. P. Lippmann, « Pattern classification using neural networks », *IEEE Communications Magazine*, November 1989.

[9] Y. Linde, A. Buzo, R. M. Gray, « An algorithm for Vector Quantizer design », *IEEE Trans. on Communications*, Vol. 28, N° 1, January 1990.

[10] D. E. Rumelhart, G. E. Hinton, R. J. Williams, « Learning internal representations by error backpropagation », Parallel Distributed Processing, D. E. Rumelhart and J. L. Mc clelland, Chap. 8, Bradford book, MIT Press, 1986.

[11] Eric Saund, « Dimensionality-Reduction using Connectionist Networks », *IEEE-PAMI*, Vol. 11, N° 3, March 1989.