UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

OPTIMISATION DES RÉSEAUX DE NEURONES DE GRANDE CAPACITÉ: ÉTUDE EXPERIMENTALE DE LEUR INEFFICACITÉ ET EXPLORATION DE SOLUTIONS

présenté par: PIÉRAUT Francis

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. TORRES-MORENO Juan-Manuel, Ph.D., président
M. BRAULT Jean-Jules, Ph.D., membre et directeur de recherche
M. BENGIO Yoshua, Ph.D., membre et codirecteur de recherche
M. LABIB Richard, Ph.D., membre

UNIVERSITÉ DE MONTRÉAL

OPTIMISATION DES RÉSEAUX DE NEURONES DE GRANDE CAPACITÉ: ÉTUDE EXPERIMENTALE DE LEUR INEFFICACITÉ ET EXPLORATION DE SOLUTIONS

FRANCIS PIÉRAUT
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
AVRIL 2003

à Caroline Courtemanche

REMERCIEMENTS

Je tiens tout d'abord à remercier mon directeur et mon co-directeur :
Jean-Jules Brault, professeur à l'École Polytechnique de Montréal, pour m'avoir orienté et conseillé tout au long de cette maîtrise, pour sa lecture attentive et ses commentaires avisés concernant mon mémoire. Yoshua Bengio, professeur à l'Université de Montréal, pour m'avoir accueilli dans son laboratoire[1], pour avoir supervisé mes travaux, pour ses commentaires avisés concernant mes travaux. Il m'a donné accès à une quantité appréciable de ressources informatiques. Par son support financier, il m'a permis de me dévouer totalement à mes travaux de recherche.

Je tiens également à remercier et à souligner la collaboration de Gilles Caporossi aux travaux et aux débats inspirants. Merci pour ton soutien et ton enthousiasme.

Merci aussi à Frédéric Morin et Réjean Ducharme pour leurs critiques et toutes les discussions concernant les travaux réalisés dans le cadre de ce mémoire.

Merci à tous les membres du LISA pour avoir su créer une atmosphère invitante et plaisante dans les pavillons du département d'informatique et de recherche opérationnel de l'Université de Montréal. Certain d'entre eux font parti maintenant de mon quotidien. Merci aussi à Cindy Villeneuve-Asselin et Caroline Courtemanche pour leur revision de mes textes.

[1] LISA

RÉSUMÉ

Le processus d'apprentissage d'un réseau de neurones devient laborieux et inefficace lorsqu'on utilise un grand nombre de neurones cachés (grande capacité). L'étude de ce comportement, l'identification des causes et la quantification expérimentale de l'impact de certaines solutions explorées sur le temps d'apprentissage sont les objectifs du présent mémoire.

Le *«data-mining»* est l'application principale visée par ce projet. Les bases de données associées à ce type d'application sont caractérisées par une quantité importante d'exemples et d'informations disponibles, d'où l'utilisation de réseaux de neurones ayant un grand nombre de paramètres libres. Du point de vue pratique, si on ne peut les entraîner correctement et assez rapidement, les réseaux de neurones ont un intérêt limité dans ce type d'application. Donc, une amélioration significative de la vitesse d'apprentissage ou une meilleur exploitation de la capacité pourrait être très utile.

Dans de nombreux articles[3,4,8], on affirme que l'efficacité de l'apprentissage diminue dramatiquement à mesure que le nombre de neurones cachés augmente, sans toutefois le documenter. Dans certaines conditions, nous avons documenté ce comportement. Plusieurs chercheurs ont identifié certaines des raisons expliquant cette inefficacité, telles *le problème du déplacement de la cible*[2] ainsi que *le problème d'atténuation et de dilution du gradient* lorsqu'on le propage dans les couches de neurones inférieures.

Nous présentons d'autres raisons pouvant expliquer ce comportement, soit *le problème des gradients contradictoires*, *le problème de l'inexistence d'un mécanisme de spécialisation* et *le problème de symétrie*.

Plusieurs solutions pouvant réduire ou éliminer certains de ces problèmes ont été explorées : différentes variantes de réseaux incrémentaux, différents algorithmes de sélection de paramètres à optimiser, différentes techniques de prédiction de paramètres et l'utilisation d'une architecture découplée. Les performances de ces techniques seront présentées et comparées à celles des réseaux de neurones conventionnels sur un ensemble de données de reconnaissance de caractères et sur une base de données créée spécialement pour analyser le comportement de certains algorithmes.

Lors de nos expériences, en utilisant une architecture de réseau de neurones découplée (partiellement interconnectée), nous avons réussi à accélérer la vitesse d'apprentissage lorsque le nombre de neurones cachés était supérieur, ce qui est contraire à notre observation pour une architecture totalement interconnectée. Les réseaux incrémentaux par couches cachées nous ont permis d'atteindre une meilleure exploitation de la capacité, mais les réseaux incrémentaux par ajouts de neurones au niveau de la

[2]Traduction de «moving target problem»

couche cachée n'ont donné aucun résultat concluant. Les meilleurs résultats obtenus à partir des techniques de prédiction des paramètres ne nous ont permis aucun gain significatif sur la vitesse d'apprentissage. Pour comparer les performances des solutions proposées, la librairie de réseaux de neurones fLayers[3] a été développée.
Ce mémoire présente à partir de résultats expérimentaux le problème d'inefficacité de l'apprentissage des réseaux de grande capacité, certaines des raisons entraînant cette inefficacité, les approches que nous avons explorées et les résultats expérimentaux de certaines solutions.

[3]f pour Francis et Layer pour le nom de la structure la plus importante de cette librairie

ABSTRACT

The training process of a neural network become hard and inefficient when there is an important number of hidden units (big capacity). The study of this behavior, the identification of problematic causes and the experimental quantification of the impact of some solutions on training time are the objectives of this master thesis. *«Data-mining»* problems is the target application type. Data bases associated with this type of applications are characterized by a significant quantity of examples and information available, hence the use of neural networks with a great number of free parameters. From a practical point of view, if we cannot train them correctly, neural networks have a limited interest for this type of application. Therefore, a significant improvement of the training speed could be very useful. In many papers[3,4,8], it is claim that the training effectiveness decreases dramatically as the number of hidden neurons increases without however documenting it. Under certain conditions, we documented this behavior. Several researchers identified some of the reasons who can explain this inefficiency, such as *the moving target problem* and *the attenuation and dilution of the error signal* as it is propagates backward through the layers of the network. We present other reasons who can explain this behavior, *the opposite gradients problems*, *the non-existence of a specialization mechanism* and the *symmetry problem*. Several solutions who can reduce or eliminate some of these problems were explored : various alternatives of incremental networks, various algorithms of parameters selection to be optimized, various techniques of prediction of parameters and the use of an uncoupled architecture. The performances of these techniques will be presented and compared with those of a static fully connected network on a character recognition dataset and on a dataset created especially to analyze the behavior of some algorithms. During our experiments, by using an uncoupled network architecture (partially inter-connected), we succeeded to speedup the training time when the numbers of hidden neurons was higher, which contradicts our observation for a fully connected architecture. Incremental networks by adding hidden layers enabled us to reach a better use of the capacity, but incremental networks by adding neurons on the level of the hidden layer did not give any conclusive results. The sporadic parameters prediction with quadratic extrapolation allowed us an training speedup, to the maximum, by a factor of 2. To compare the suggested solutions performances, the C++ library (fLayers[4]) of neural networks was developed. This master thesis present experimental results who shows the inefficiency of the training process of big neural networks, some of the reasons who can explain this inefficiency, explored approaches and experimental results of some solutions.

[4] f for Francis and Layer represent the most important structure of this library

TABLE DES MATIÈRES

REMERCIEMENTS . . . . . **v**

RÉSUMÉ . . . . . **vi**

ABSTRACT . . . . . **viii**

TABLE DES MATIÈRES . . . . . **ix**

LISTE DES FIGURES . . . . . **xiii**

SYMBOLES . . . . . **xv**

ACRONYMES . . . . . **xvii**

LEXIQUE . . . . . **xviii**

LISTE DES TABLEAUX . . . . . **xx**

LISTE DES ANNEXES . . . . . **xxi**

CHAPITRE 1 :INTRODUCTION **1**
1.1 Motivation et contexte de la recherche 1
1.2 Aperçu du mémoire 2

CHAPITRE 2 :RÉSEAUX DE NEURONES **3**
2.1 Architecture standard 3
2.1.1 Équations pour le calcul du gradient 3
2.2 Choix d'initialisation 5
2.3 Algorithme du gradient stochastique 6
2.4 Fonction de coût 6
2.5 Interprétation en terme de plans séparateurs 7
2.6 Comportement attendu en fonction de la capacité 9

CHAPITRE 3 :PRÉSENTATION DU PROBLÈME **13**
3.1 La vitesse d'apprentissage diminue à mesure que la capacité du réseau augmente 13
3.2 Une augmentation de la capacité ne permet qu'une réduction très faible du nombre d'erreurs 16

3.3 Pourquoi l'apprentissage au moyen de la rétro-propagation est-il si lent ? 18
3.3.1 Choix de la valeur de la cible en classification 18
3.3.2 Utiliser un pas pour chaque paramètre 19
3.3.3 Problème de la taille des pas 19
3.3.4 Problème du déplacement de la cible 20
3.3.5 Problème des gradients contradictoires en classification 20
3.3.6 Problème de l'inexistence d'un mécanisme de spécialisation . . 22
3.4 Pourquoi la vitesse d'apprentissage se détériore t-elle à mesure qu'augmente la capacité d'un réseau ? 23
3.4.1 Atténuation et dilution du gradient 23
3.5 Pourquoi une augmentation de la capacité ne permet-elle qu'une réduction très faible du nombre d'erreurs ? 23
3.5.1 Problème de l'inexistence d'un mécanisme de spécialisation des paramètres 24
3.5.2 Problème de symétrie 24
3.6 Est-il possible qu'une capacité accrue accélère l'apprentissage ? 24
3.7 Hypothèse expliquant l'inefficacité des réseaux de grande capacité . . 25

CHAPITRE 4 : APPROCHES ET SOLUTIONS 26
4.1 Approches 26
4.1.1 Diviser pour régner 27
4.1.2 Optimiser une partie des paramètres 28
4.1.3 Prédiction des valeurs des paramètres 28
4.1.4 Choix de l'architecture 28
4.2 Solutions 29
4.3 Réseaux incrémentaux 29
4.3.1 Définition 29
4.3.2 Intérêts 30
4.3.3 Approches pour ajouter des neurones 30
4.3.4 Courbure de la fonction de coût 30
4.3.5 Problématique de l'initialisation des poids 31
4.3.6 Choix d'initialisation des nouveaux paramètres 32
4.3.7 Calcul du gradient sur les nouveaux paramètres 33
4.3.8 Validation 33
4.3.9 Algorithme 34
4.3.10 Considération d'efficacité 34
4.3.11 Impacts sur les différents problèmes 37
4.4 Optimisation d'une partie des paramètres 38
4.4.1 Optimisation des paramètres ayant les gradients les plus élevés 38
4.4.2 Optimisation circulaire 39

4.4.3 Optimisation des paramètres affectant le plus l'erreur de classification . . . 39
4.4.4 Impacts sur les différents problèmes . . . 39
4.5 Prédiction des valeurs de paramètres . . . 40
4.5.1 Précision de l'estimation . . . 41
4.5.2 Considération d'efficacité . . . 41
4.5.3 Approches pour extrapoler . . . 43
4.5.4 Extrapolation quadratique . . . 43
4.5.5 Extrapolation cubique . . . 43
4.5.6 Régression linéaire . . . 44
4.6 Architecture découplée . . . 45
4.6.1 Problème relié à une architecture découplée . . . 45
4.7 Impacts des solutions . . . 47

CHAPITRE 5 :RÉSULTATS . . . 48
5.1 Cadre expérimental . . . 48
5.2 Réseaux incrémentaux - ajout de neurones (plans séparateurs) . . . 48
5.2.1 Résultats : RIPS . . . 48
5.2.2 Hypothèse basée sur les points critiques . . . 49
5.2.3 Comportement sur un exemple simple . . . 50
5.2.4 Résultats . . . 51
5.2.5 Optimisation de tous les paramètres . . . 53
5.2.6 Optimisation uniquement des nouveaux paramètres . . . 54
5.3 Résultats : RICC . . . 54
5.3.1 Est-ce efficace de continuer à rétro-propager dans les couches inférieures ? . . . 59
5.4 Résultats : Optimisation d'une partie des paramètres . . . 60
5.4.1 Résultats : ROC . . . 60
5.4.2 Résultats : ROGE . . . 62
5.5 Résultats : Comportement lorsqu'on accroît la capacité pour un réseau découplé . . . 64
5.6 Résultats : Extrapolation versus une augmentation du pas . . . 67
5.7 Synthèse des résultats expérimentaux . . . 67

CHAPITRE 6 :CONCLUSION . . . 69
6.1 Observations . . . 70
6.2 Contributions expérimentales . . . 70
6.3 Travaux futurs . . . 70

BIBLIOGRAPHIE **72**
II.1 Config. 1 : Architecture standard (100 neurones cachés, MSE) 75
II.2 Config. 2 : Architecture standard (100 neurones cachés, LogSoftMax) 75
II.3 Config. 3 : Architecture découplée (9 neurones cachés/sortie, LogSoftMax) 76

LISTE DES FIGURES

2.1 Architecture standard (totalement interconnecté) 4
2.2 Tracé de la sigmoïde (éq. 2.10) et de sa dérivée (éq. 2.11) 6
2.3 Algorithme du gradient stochastique 7
2.4 Exemple du comportement de l'erreur de classification et du coût en fonction du temps 8
2.5 Sensibilité en fonction de la sortie (MSE (éq. 2.13)) 8
2.6 Sensibilité en fonction de la sortie (LogSoftMax (éq. 2.14)) 9
2.7 Perceptron et exemple de frontière de décision 10
2.8 Exemple d'un percepton dans un réseau de neurones 11
2.9 Comportement typique et souhaité en fonction de la capacité (atteinte d'un taux d'erreur inférieur lorsque la capacité est supérieure) 12

3.1 Ralentissement de l'apprentissage durant l'entraînement en fonction de l'augmentation de la capacité (base de donnée «letters») 14
3.2 Ratio erreur(200 neurones cachés, 100 neurones cachés, t) 15
3.3 Comportement observé du ratio erreur 15
3.4 Comportement typique en fonction de la capacité 16
3.5 Agrandissement de la figure 3.4 17
3.6 Problème des gradients contradictoires d'un réseau totalement interconnecté vs un réseau découplé 21
3.7 Espaces de solution et exemple de solution 25

4.1 Ajout d'un plan séparateur au niveau de la couche cachée 30
4.2 Ajout d'une couche cachée 31
4.3 Perceptron - Configuration «Cascade Correlation» 31
4.4 Courbure de la fonction de coût 32
4.5 Optimisation des nouveaux paramètres 34
4.6 Optimisation pour la propagation lorsque certains des paramètres sont fixes 35
4.7 Algorithme du gradient stochastique modifié 36
4.8 Algorithme incrémental 36
4.9 Algorithme incrémental efficace 36
4.10 Exemples d'évolution de 6 poids synaptiques choisis au hasard (MSE et LogSofMax) 40
4.11 Exemples d'évolution de 6 biais synaptiques choisis au hasard (MSE et LogSofMax) 41

4.12 Architecture découplée (partiellement interconnectée) 45

5.1 Résultat Réseau incrémental (erreur) 49
5.2 Résultat Réseau incrémental (coût) 50
5.3 MLDB 51
5.4 Surfaces de décision et région de classification 52
5.5 2 plans séparateurs - Surfaces de décision et régions de classification . 52
5.6 3 plans séparateurs - Surfaces de décision 53
5.7 Résultats 2 neurones cachés 54
5.8 Résultats 3 neurones cachés 55
5.9 Résultats 5 neurones cachés ; 1ère configuration 56
5.10 Résultats 5 neurones cachés ; 2ème configuration 57
5.11 Réseau incrémental - Optimisation de tous les paramètres 58
5.12 Ajout de couches cachées 58
5.13 Comparaison avec et sans l'optimisation de la couche inférieure 59
5.14 Optimisation circulaire 61
5.15 3 neurones cachés : Optimisation des paramètres ayant les gradients les plus élevés 62
5.16 3 neurones cachés : Optimisation standard 63
5.17 Architecture découplée : accéleration de l'apprentissage en fonction de l'augmentation de la capacité 64
5.18 Architecture découplée : comportement de l'apprentissage en fonction de l'augmentation de la capacité 65
5.19 Architecture découplée vs Architecture standard 66
5.20 Extrapolation quadratique sporadique 67
5.21 Augmentation sporadique du pas 68

III.1 Influence d'une perturbation constante 78
III.2 Perturbation 78

IV.1 Influence d'une perturbation sporadique 80

V.1 Optimisation standard 81
V.2 Optimisation circulaire 82
V.3 Optimisation : Max(dérivée de la fonction d'activation) 82
V.4 Optimisation : Max(sortie) 82
V.5 Optimisation : Max(*sensibilité*) 83
V.6 Optimisation : Max(Gradient) 83

SYMBOLES

z_k Représente la valeur de sortie du k ième neurone

δ Sensibilité

δ_k Sensibilité du k ième neurone de sortie

δ_j Sensibilité du j ième neurone caché

w Lettre utilisée pour représenter un poids, w pour «weight»

w_{jk} Poids synaptique reliant le neurone caché j au neurone de sortie k

w_{ij} Poids synaptique reliant le neurone d'entrée i au neurone caché j

∇ Gradient (voir lexique)

x_i Valeur d'entrée du i ième neurone d'entrée

y_j Sortie du j ième neurone caché

n Valeur du pas du gradient "learning rate" $(0 < n < 1)$

n_h Nombre de neurones cachés

e Lettre utilisée pour représenter le nombre d'époques

J Représente la fonction de coût

h Nombre de neurones cachés, h pour "hidden units"

net_k Valeur de sortie du neurone k ; $net_k = g'_z(z_k)$

net_j Valeur de sortie du neurone j ; $net_j = g'_j(y_j)$

g_z Fonction d'activation du neurone de sortie z

g_y Fonction d'activation du neurone caché y

Θ Valeur représentant le critère d'arrêt

z_k Représente la valeur de sortie du k ième neurone

δ Sensibilité

δ_k Sensibilité du k ième neurone de sortie

δ_j Sensibilité du j ième neurone caché

w Lettre utilisée pour représenter un poids, w pour «weight»

w_{jk} Poids synaptique reliant le neurone caché j au neurone de sortie k

w_{ij} Poids synaptique reliant le neurone d'entrée i au neurone caché j

∇ Gradient (voir lexique)

x_i Valeur d'entrée du i ième neurone d'entrée

y_j Sortie du j ième neurone caché

n Valeur du pas du gradient "learning rate" $(0 < n < 1)$

n_h Nombre de neurones cachés

e Lettre utilisée pour représenter le nombre d'époques

J Représente la fonction de coût

h Nombre de neurones cachés, h pour "hidden units"

net_k Valeur de sortie du neurone k ; $net_k = g'_z(z_k)$

net_j Valeur de sortie du neurone j ; $net_j = g'_j(y_j)$

g_z Fonction d'activation du neurone de sortie z

g_y Fonction d'activation du neurone caché y

Θ Valeur représentant le critère d'arrêt

ACRONYMES

bprop «Back propagation» (rétro-propagation)

CPU Central Processing Unit (Unité centrale de traitement)

fprop «Forward propagation» (Propagation des entrées aux sorties du réseau de neurones)

MLDB «Base de Donnée Minimum Local», nom donnée à la base de données que nous avons créé (voir figure 5.3)

LISA Laboratoire des systèmes adaptatifs (Université de Montréal)

LogSoftMax Logarithme du SoftMax

MSE «Mean Square Error»=$(t - z_i)^2$ (différence de l'erreur au carré)

SoftMax $y_i = \frac{e^{z_i}}{\sum_{t=1}^{c} e^{z_t}}$

RICC Réseau Incrémental par ajout de Couches Cachées

RIPS Réseau Incrémental par ajout de Plans Séparateurs (Ajout de neurones au niveau de la couche cachée)

ROGE Réseau avec Optimisation des groupes de connections pour lesquels les Gradients sont les plus Élevés

ROC Réseau avec Optimisation Circulaire

tanh Tangente hyperbolique

LEXIQUE

Algorithme d'apprentissage Nom représentant l'ensemble des algorithmes permettant d'inférer une fonction à partir d'exemples.

Algorithme parallélisable Algorithme dont plusieurs sections peuvent être exécutées simultanément.

Architecture standard Un réseau dont l'architecture est standard signifie qu'il est totalement interconnecté.

Biais Valeur numérique que l'on ajoute à la sortie d'un neurone suite à l'application de la fonction d'activation.

Capacité Représente le potentiel d'un réseau de neurones à modéliser une fonction complexe. La capacité est reliée au nombre de paramètres libre d'un réseau de neurones.

Cible Valeur à prédire.

Classe Catégorie à prédire. En classification, la prédiction consiste à décider à quelle catégorie est associée une observation.

Classification Prédiction d'une caractéristique discrète, d'une classe.

Coût Valeur numérique représentant l'importance de l'erreur.

«Data-mining» Mot anglais signifiant «exploitation des données». Se dit de techniques logicielles permettant de faire apparaître de nouvelles relations utiles entre des données extraites des gigantesques bases de données gérées par certaines grandes entreprises.

Époque Lors du processus d'apprentissage, une époque est complétée lorsque tous les exemples ont été utilisés pour entraîner le modèle. En général, plusieurs époques sont nécessaires pour entraîner convenablement un réseau de neurones.

Espace de solution Ensemble des solutions permises par les différentes combinaisons de valeurs des paramètres d'un réseau de neurones.

Fonction d'activation Fonction que l'on applique à la sortie d'un neurone.

Généralisation Capacité d'un modèle à faire de bonnes prédictions sur des exemples n'ayant pas servi à l'entraînement du modèle.

Gradient Le gradient représente la modification infinitésimale à appliquer à un paramètre pour réduire le coût.

Gradient batch On utilise la somme des gradients de tous les exemples de l'ensemble d'entraînement pour modifier les paramètres.

Gradient stochastique On utilise le gradient d'un exemple à la fois pour modifier les paramètres.

Grappe d'ordinateurs Ensemble d'ordinateurs connectés entre eux.

Heuristique Règle qui n'est pas basée sur des fondements théoriques mais sur une idée raisonnable.

Matrice hessienne Matrice de la dérivée seconde :
$$\begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_{i=1} \partial x_{j=1}} & \cdots & \frac{\partial^2 f(x)}{\partial x_{i=1} \partial x_{j=n}} \\ \cdots & \cdots & \cdots \\ \frac{\partial^2 f(x)}{\partial x_{i=n} \partial x_{j=1}} & \cdots & \frac{\partial^2 f(x)}{\partial x_{i=n} \partial x_{j=n}} \end{pmatrix}$$

Observation Série de caractéristiques associées à un exemple ou un élément à classifier.

Optimisation Dans le contexte des réseaux de neurones, l'optimisation est la minimisation de la fonction de coût.

Paramètre(s) Dans le contexte des réseaux de neurones, les paramètres sont tous les biais et tous les poids synaptiques du réseau.

Pas Facteur multiplicatif que l'on utilise pour calculer le gradient («learning rate» en anglais).

Poids ou poids synaptiques Valeur numérique associée à une connection entre deux neurones.

Point critique Caractéristique d'une position de la surface de coût où la dérivée première est nulle.

Réseau standard Réseau de neurones totalement interconnecté (voir figure 2.1).

Sensibilité La sensibilité d'un neurone k est la dérivée partielle du coût par rapport à la sortie du neurone k ($\partial J / \partial net_k$).

Sigmoïde Nom d'une fonction utilisée comme fonction d'activation (voir figure 2.2).

Surapprentissage Phénomène qui se produit quand le processus d'apprentissage entraîne une réduction de la généralisation causé par une modélisation trop exacte des données d'entraînement.

Itération Une itération consiste à effectuer la propagation des valeurs d'entrée d'un réseau de neurones et de rétro-propager l'erreur.

Réseau de neurones Un réseau de neurones (artificiel) est un outil dont l'architecture est inspirée des neurones du cerveau. Il ressemble au cerveau par deux aspects : il est capable d'acquérir des connaissances par un processus d'apprentissage, l'information stockée est utilisable pour effectuer de la prédiction.

Rétro-propagation Nom de l'algorithme de calcul du gradient utilisé pour effectuer l'apprentissage d'un réseau de neurones. Il permet de rétro-propager le coût dans les couches de neurones inférieures.

LISTE DES TABLEAUX

4.1 Solutions 29
4.2 Impacts des solutions sur les différents problèmes (voir page xxii pour la définition des acronymes) 47

5.1 Impacts réels des solutions (voir page xxii pour la définition des acronymes) 68

I.1 Nombre d'opérations à exécuter 74
I.2 Comparaison N vs 2N neurones cachés 74

LISTE DES ANNEXES

Annexe I :	**Comparaison du temps CPU : N versus 2N neurones**	**74**
Annexe II :	**Profiling**	**75**
Annexe III :	**Perturbation**	**77**
Annexe IV :	**Perturbation sporadique**	**79**
Annexe V :	**Optimisation sélective**	**81**

CHAPITRE 1

INTRODUCTION

Dans de nombreux domaines, nous sommes intéressés à faire de la prédiction. Les réseaux de neurones artificiels permettent de réaliser cet objectif. L'apprentissage consiste à extraire les relations d'un ensemble d'observations et à les utiliser pour faire de la prédiction sur de nouveaux cas.

1.1 Motivation et contexte de la recherche

Le nombre de paramètres d'un réseau de neurones est relié à sa capacité ou son potentiel de représenter une relation complexe. Plus la capacité est grande, plus l'espace des solutions est grand[1]. En pratique, on constate que le processus d'apprentissage des réseaux de neurones de grande capacité est extrêmement lent et inefficace[8,9,17]. Pour certains problèmes, le nombre important de paramètres à utiliser est nécessaire et ceci est dû au nombre de caractéristiques disponibles et/ou à la complexité de la fonction à apprendre. Les problèmes de *«data-mining»* [3,4] entrent dans cette catégorie. D'autres types de problèmes nécessitent également une grande capacité. À titre d'exemple, au LISA, nous sommes intéressés à utiliser des réseaux de neurones pour représenter un modèle de language. Cela consiste à modéliser la probabilité d'un mot sachant les mots précédents. Le réseau permettant une telle modélisation doit posséder nécessairement un nombre important de neurones d'entrée et de sortie et peut prendre de plusieurs heures à plusieurs jours pour effectuer une seule époque. Le présent mémoire expose certaines raisons expliquant ce comportement, les solutions envisagées et leurs impacts en temps de calcul sur des exemples réels. Nous attribuons l'inefficacité des réseaux de grande capacités à des problèmes d'optimisation. Ayant constaté que le *surapprentissage* de ce type de réseau est difficilement atteignable, les considérations de généralisation sont négligées. Pour chacune des solutions que nous avons explorées, nous considérerons l'accélération de l'apprentissage sur l'ensemble d'entraînement et l'amélioration du taux d'apprentissage final. De façon générale, l'évolution de l'erreur d'apprentissage sera présentée en fonction du temps de calcul et non en fonction du nombre d'époques.

[1] Voir notes de cours de Yoshua Bengio http ://www.iro.umontreal.ca/bengioy/ift6266/

1.2 Aperçu du mémoire

Le chapitre 2 présente les éléments nécessaires à la compréhension de ce qu'est un réseau de neurones. Le chapitre 3 expose le problème d'inefficacité des réseaux de grande capacité et les problèmes d'optimisation sous-jacents. Le chapitre 4 traite des approches et des solutions explorées. Le chapitre 5 fait état des résultats expérimentaux et le chapitre 6 présente la conclusion.

CHAPITRE 2

RÉSEAUX DE NEURONES

Dans ce chapitre, nous présentons les notions de base des réseaux de neurones nécessaires à la compréhension des chapitres subséquents. Les différents points traités seront l'architecture standard d'un réseau de neurones, l'algorithme de la descente du gradient stochastique, les équations pour le calcul des gradients pour les différents paramètres, le problème du choix d'initialisation des paramètres, certaines considérations à propos de la fonction de coût, l'interprétation en terme de plans séparateurs et le comportement attendu en fonction de la capacité du réseau. Ce chapitre est fortement inspiré du chapitre sur les réseaux de neurones du livre «Patter Classification»[5] et du livre «Neural Networks for Parttern Recognition»[1].

2.1 Architecture standard

La figure 2.1 présente l'architecture d'un réseau de neurones standard à une couche cachée utilisée pour un problème de classification. Chaque cercle est associé à un neurone et chaque connexion à un poids synaptique. Le nombre de classes correspond au nombre de sorties. En régression linéaire, nous n'aurions eu qu'une sortie dont la fonction d'activation serait linéaire. Ce réseau a Z sorties, H neurones cachés et D entrées. En théorie, un réseau de neurones d'une couche cachée peut modéliser n'importe quelle relation non-linéaire[1]. La complexité des fonctions ou l'espace des solutions qui peut être représenté dépend du nombre de paramètres libres du réseau. En augmentant le nombre de neurones cachés, il est possible de modéliser des fonctions plus complexes.

2.1.1 Équations pour le calcul du gradient

Les équations pour calculer les gradients pour les différents paramètres d'un réseau de neurones à une couche cachée sont données par les équations 2.5 et 2.6, où J est la fonction de coût, g_z la fonction d'activation de la sortie et g_y celle des neurones cachés :

$$y_j = g_y(\sum_{i=1}^{d} w_{ij} \cdot x_i) \tag{2.1}$$

$$z_k = g_z(\sum_{j=1}^{h} w_{kj} \cdot y_j) \tag{2.2}$$

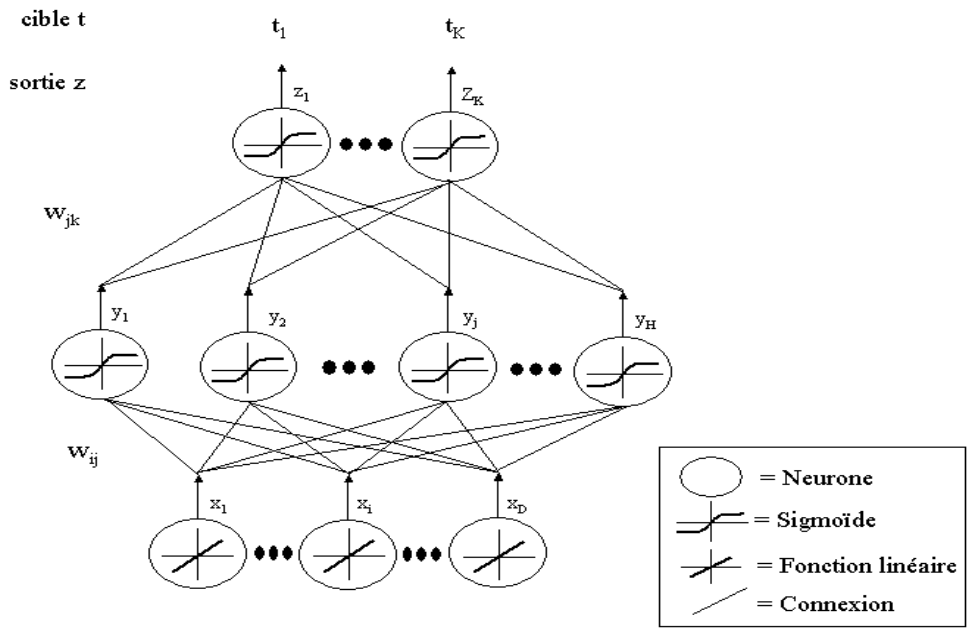

FIG. 2.1 – Architecture standard (totalement interconnecté)

$$\delta_k = (\frac{\partial J}{\partial z_k}) \cdot g'_z(z_k) \tag{2.3}$$

$$\delta_j = [\sum_{k=1}^{c} w_{kj} \cdot \delta_k] \cdot g'_y(y_j) \tag{2.4}$$

$$\nabla w_{ij} = \delta_j \cdot x_i \tag{2.5}$$

$$\nabla w_{kj} = \delta_k \cdot y_j \tag{2.6}$$

2.2 Choix d'initialisation

Le choix initial des poids d'un réseau influence significativement le processus d'entraînement. Les poids doivent être choisis au hasard, mais il faut s'assurer que les sigmoïdes sont activées dans la région linéaire. Dans le cas où les poids ont des valeurs numériques élevées, les sigmoïdes vont saturer et engendrer des gradients faibles qui ralentiront l'apprentissage initial. Si les valeurs sont trop petites, l'apprentissage sera aussi très lent. Des valeurs de poids moyennes, amenant la sigmoïde dans la région légèrement non-linéaire, entraînent les gradients à être suffisamment grands pour permettre l'apprentissage et assurer que le réseau apprenne les relations linéaires avant les parties non-linéaires de la fonction à apprendre[15]. Si la fonction d'activation est une sigmoïde ou une tangente hyperbolique et qu'elle est activée dans la région linéaire, les valeurs de la dérivée se trouveront dans la zone de valeurs les plus élevées et par conséquent, le gradient sera élevé. Accomplir une bonne initialisation nécessite la coordination de la normalisation de l'ensemble d'entraînement, du choix de la sigmoïde et du choix d'initialisation des poids. La distribution des sorties de chaque neurone doit idéalement avoir une variance égale à 1 et une moyenne égale à 0 [15]. Cela est accompli au niveau de la couche d'entrée en normalisant tous les exemples d'entraînement. Cette normalisation s'effectue en soustrayant leurs moyennes respectives puis en les divisant par leurs écarts types. Pour obtenir une variance près de 1 au niveau de la première couche cachée, il ne suffit que d'appliquer une tanh dans le cas où l'on est assuré que ses entrées ont toutes un écart-type de 1. Si l'on assume que les entrées sont non-corrélées et qu'elles ont toutes un écart-type de 1, alors la variance de la somme des poids est donnée par l'équation 2.7 (où m est le nombre d'entrées du neurone). Pour s'assurer que l'écart-type soit approximativement 1, les poids (équation 2.9) doivent être tirés aléatoirement d'une distribution de moyenne 0 et de variance donnée par l'équation 2.8.

$$\sigma_y = (\sum_j w_{ij}^2)^{1/2} \tag{2.7}$$

$$\sigma_w = m^{-1/2} \tag{2.8}$$

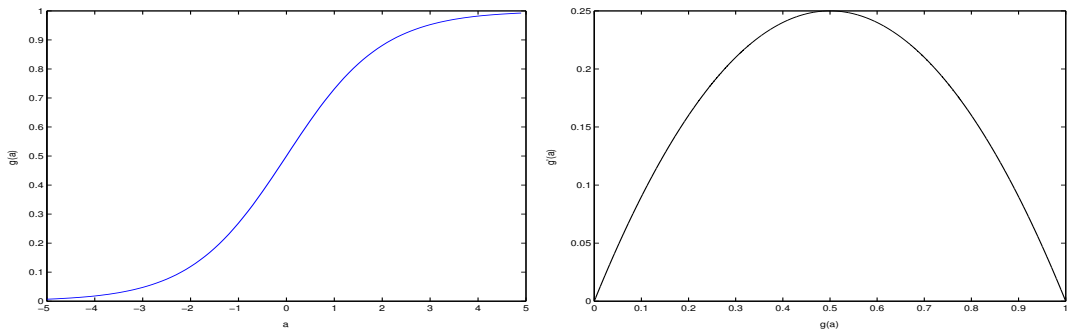

FIG. 2.2 – Tracé de la sigmoïde (éq. 2.10) et de sa dérivée (éq. 2.11)

$$w_{kj} \sim U(0, \sigma_w) \tag{2.9}$$

La figure 2.2 présente la dérivée de la sigmoïde en fonction de sa valeur. Il est à noter que la dérivée se situe dans la section des valeurs maximales lorsque ses valeurs sont dans la région linéaire.

$$g(a) = \frac{1}{1 + \exp(-a)} \tag{2.10}$$

$$g'(a) = g(a) \cdot (1 - g(a)) \tag{2.11}$$

2.3 Algorithme du gradient stochastique

Dans le contexte des réseaux de neurones, l'apprentissage se fait de façon itérative. À chaque itération, nous modifions les poids des synapses et les biais du réseau au moyen d'une technique de descente de gradient. On définit un critère à minimiser que l'on appelle le coût. Pour des raisons de performance, nous ne nous intéresserons qu'à la descente de gradient stochastique, une technique où l'on applique le gradient après chaque exemple. L'algorithme d'apprentissage consiste à propager les valeurs des caractéristiques d'un exemple des entrées jusqu'aux sorties, d'appliquer la fonction de coût et de le rétro-propager.

La descente de gradient stochastique permet de converger plus rapidement que la descente de gradient «batch» et c'est pour cette raison que nous nous sommes limités à cette technique[15].

2.4 Fonction de coût

Tous les gradients dépendent de la fonction de coût. Les deux fonctions de coût les plus fréquemment utilisées sont la différence de l'erreur quadratique (MSE) et le

Initialisation n, Θ, e=0, m=0
Faire e=e+1 (epoques)
  Faire m=m+1 (pour tous les exemples)
    x=exemple choisi au hasard
    propager les valeurs x_1 - x_d
    calculer les δ_k selon la fonction de coût
    calculer les $\nabla w_{jk} = \delta_k \cdot y_j$
    calculer les $\nabla \delta_j ; \delta_j = [\sum_{k=1}^{c} w_{kj} \cdot \delta_k] \cdot g'_y(y_j)$
    calculer les $\nabla w_{ij} = \delta_j \cdot x_i$
    $w_{ij} \leftarrow w_{ij} - n \cdot \nabla w_{ij}$
    $w_{jk} \leftarrow w_{jk} - n \cdot \nabla w_{jk}$
  Tant que $\|\nabla J\| < \Theta$
Tant que m < nombre d'exemples

FIG. 2.3 – Algorithme du gradient stochastique

logarithme probabiliste (LogSoftMax). Lorsqu'on est dans le contexte de la classification, nous sommes intéressés à minimiser l'erreur de classification. Sachant que cette erreur n'est pas dérivable, nous sommes contraint de minimiser un critère différent. En général, nous avons utilisé le LogSoftMax. En pratique, tel que présenté par l'exemple de la figure 2.4, l'erreur de classification et le coût se comportent de façon très similaire. Pour transformer la sortie du réseau en probabilité, nous faisons un softmax (équation 2.12). δ_k, la *sensibilité* du neurone de sortie k, égale $(t - z_t) \cdot g'_z(z_k)$ lorsque la fonction de coût est le MSE mais est égale à $(t - z_t)$ lorsque la fonction de coût est le LogSoftMax. Les figures 2.5 et 2.6 illustrent les valeurs de δ_k selon la fonction de coût choisie lorsque la cible est égale à 1. En comparant les deux figures, nous constatons la répercussion du choix de la fonction de coût sur les gradients. En pratique, le LogSoftMax est une fonction beaucoup plus discriminante que la MSE et nous la favorisons lorsque les données sont peu bruitées.

$$y_i = \frac{e^{z_i}}{\sum_{t=1}^{c} e^{z_t}} \tag{2.12}$$

$$\delta_k = (t - z_t) \cdot g'_z(z_k) \tag{2.13}$$

$$\delta_k = (t - z_t) \tag{2.14}$$

2.5 Interprétation en terme de plans séparateurs

Un perceptron permet de trouver une frontière de décision linéaire séparant deux classes tel que présenté à la figure 2.7. La frontière entre la région 1 (R1) et la région

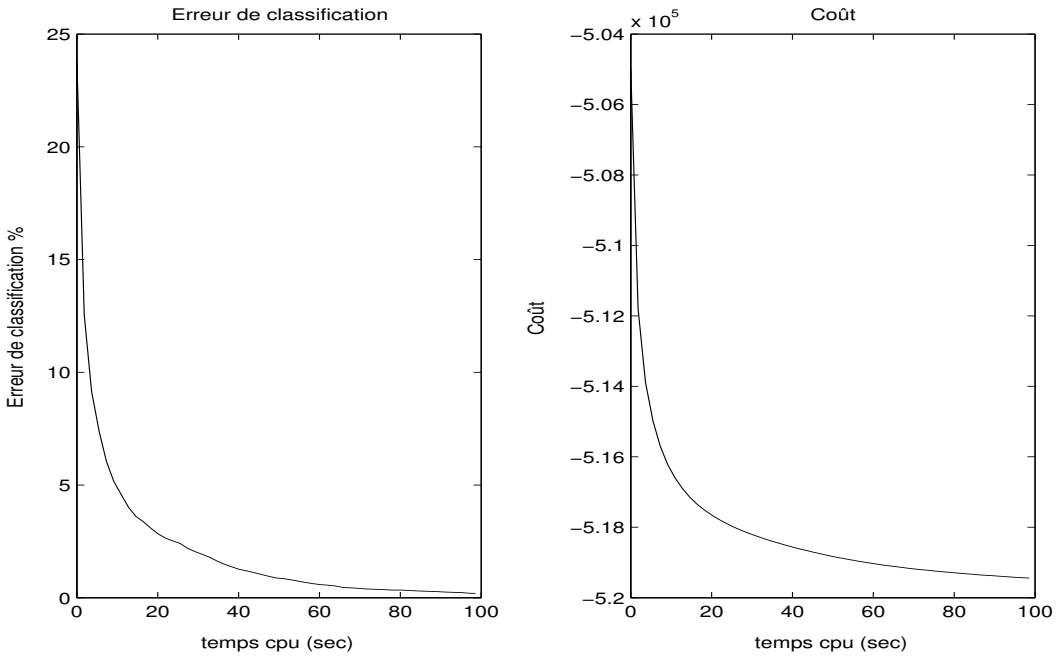

FIG. 2.4 – Exemple du comportement de l'erreur de classification et du coût en fonction du temps

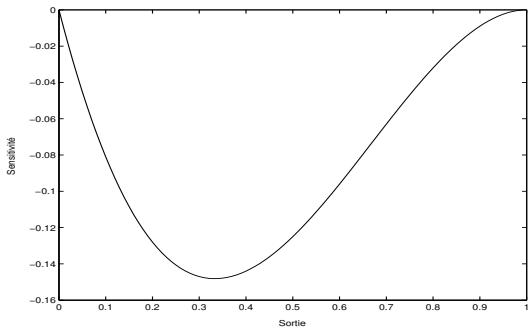

FIG. 2.5 – Sensibilité en fonction de la sortie (MSE (éq. 2.13))

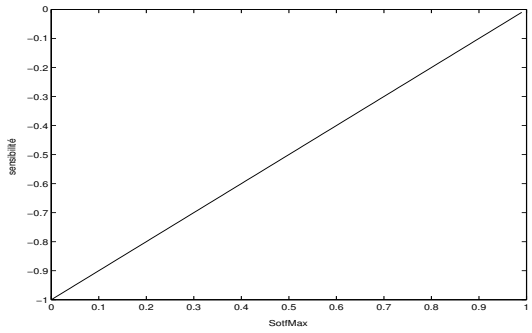

FIG. 2.6 – Sensibilité en fonction de la sortie (LogSoftMax (éq. 2.14))

2 (R2) représente une surface de décision. Chaque neurone caché d'un réseau à une couche cachée est en fait un simple perceptron (voir figure 2.8). La couche supérieure du réseau nous permet de trouver les combinaisons des positions par rapport aux différents plans nous permettant de faire le classement.

2.6 Comportement attendu en fonction de la capacité

La figure 2.9 représente le comportement typique et souhaité de l'évolution de l'erreur en fonction du temps lorsqu'on entraîne sur un même problème des réseaux de différente capacité. Une capacité plus élevée permet d'avoir des modèles plus complexes et par conséquent, la possibilité d'atteindre de plus faibles taux d'erreur. Malencontreusement, ce comportement n'est pas respecté lorsque la capacité est très élevée. De façon générale, l'apprentissage est très rapide au début, mais ralentit très rapidement et devient de plus en plus difficile.

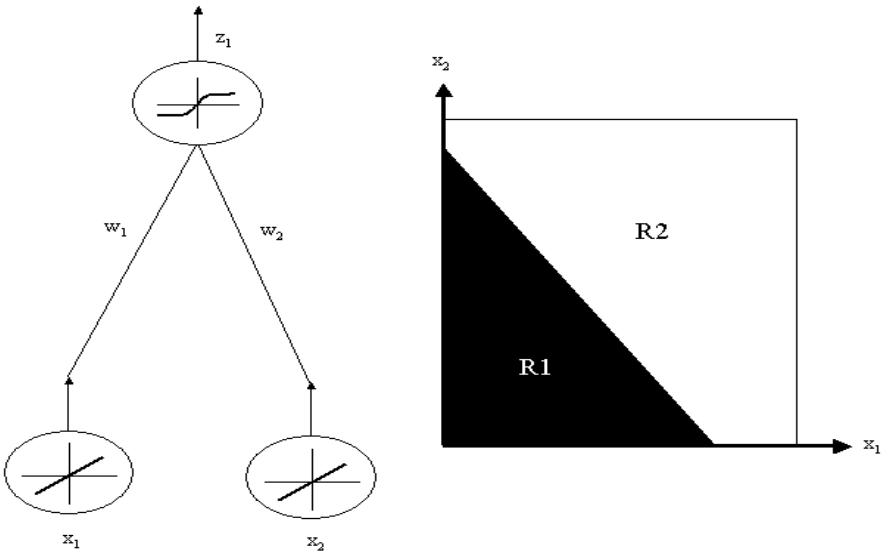

FIG. 2.7 – Perceptron et exemple de frontière de décision

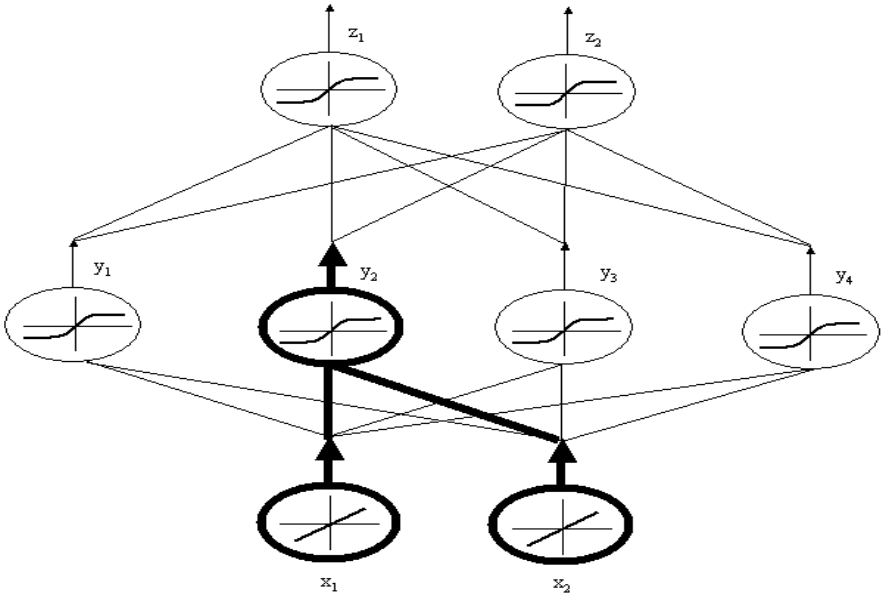

FIG. 2.8 – Exemple d'un percepton dans un réseau de neurones

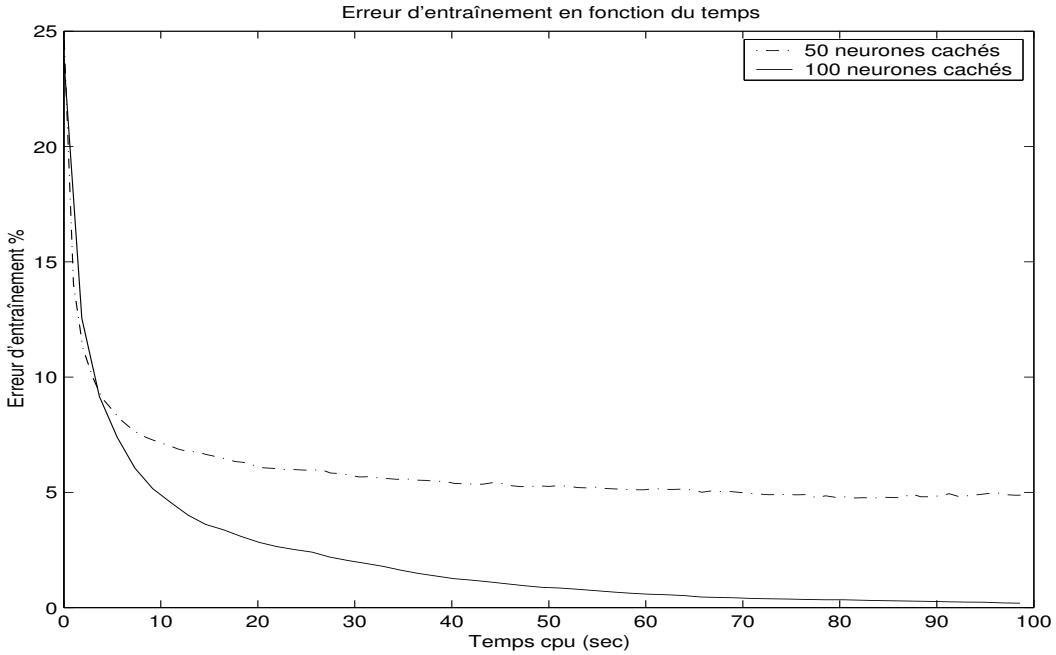

FIG. 2.9 – Comportement typique et souhaité en fonction de la capacité (atteinte d'un taux d'erreur inférieur lorsque la capacité est supérieure)

CHAPITRE 3

PRÉSENTATION DU PROBLÈME

Le problème de l'inefficacité d'apprentissage d'un réseau de neurones ayant un grand nombre de neurones cachés peut se voir de deux façons :

- La vitesse de l'apprentissage diminue à mesure que la capacité du réseau augmente (§3.1)
- Une augmentation de la capacité ne permet qu'une réduction très faible du nombre d'erreurs (§3.2)

Il est important de souligner que la première constatation dépend du temps et que la seconde n'en dépend pas. Ces problèmes seront présentés à partir de résultats expérimentaux obtenus en entraînant nos réseaux de neurones sur la base de donnée de reconnaissance de caractères «Letters» (§5.1). Par la suite, nous présenterons les considérations pour une utilisation efficace de l'algorithme de rétro-propagation et ses problèmes inhérents (§3.3). Puis, nous présenterons certaines raisons pour lesquelles l'apprentissage se détériore à mesure que la capacité augmente (§3.4) et certaines raisons expliquant la difficulté d'exploitation d'une capacité accrue (§3.5). Le présent chapitre sera conclu sur notre interrogation concernant la possibilité qu'une augmentation de la capacité puisse accélérer l'apprentissage ainsi que notre hypothèse expliquant l'inefficacité des réseaux de grande capacité.

3.1 La vitesse d'apprentissage diminue à mesure que la capacité du réseau augmente

Lors de nos expériences, nous avons constaté que plus la capacité d'un réseau est élevée, plus le réseau prendra du temps pour atteindre des taux d'erreur équivalant à un réseau de plus faible capacité. Cette observation n'est valide, en tout temps, qu'au début de l'apprentissage(voir figure 2.9). Lorsque la capacité d'un réseau est insuffisante pour représenter la fonction à apprendre, il est évident qu'après un certain temps, un réseau de plus grande capacité atteigne des taux d'erreur inférieurs plus rapidement.

Par contre, au delà d'une certaine capacité, cette observation est toujours valide (figure 3.1). Pour caractériser ce comportement, nous avons défini le *ratio erreur*. Le ratio erreur (équation 3.1) est le ratio de l'erreur d'apprentissage obtenu en entraînant un réseau de neurones de capacité C[1] par rapport à l'erreur d'apprentissage obtenu

[1] C pour capacité de Comparaison

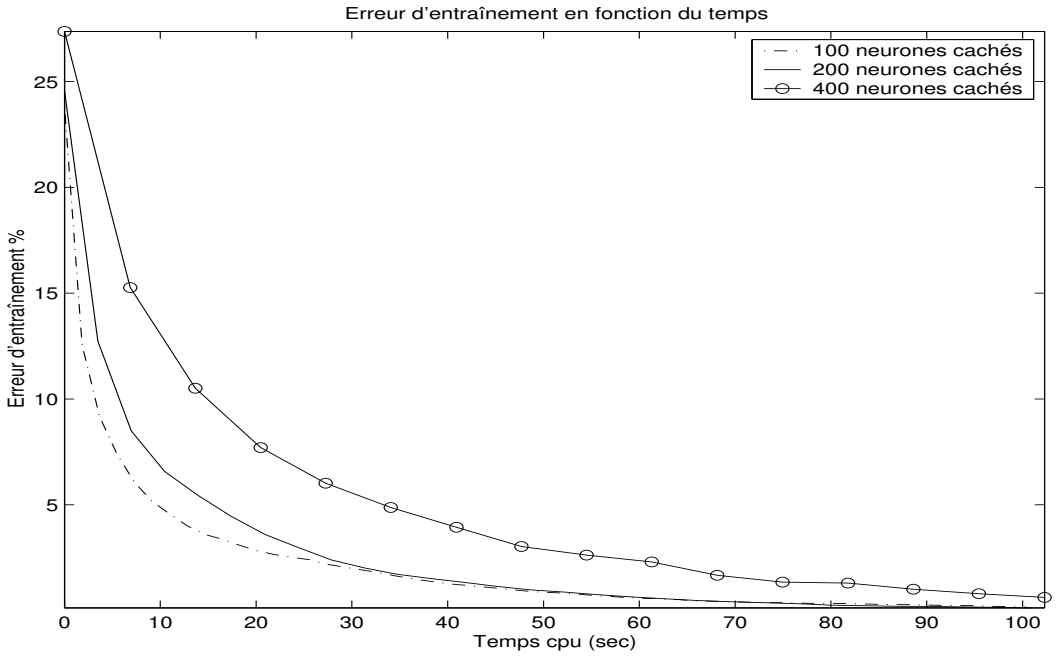

FIG. 3.1 – Ralentissement de l'apprentissage durant l'entraînement en fonction de l'augmentation de la capacité (base de donnée «letters»)

en entraînant un réseau de capacité R[2] après un temps de calcul identique (t).

$$Ratio\ erreur(C, R, t) = \frac{Erreur_C(t)}{Erreur_R(t)} \tag{3.1}$$

Où $Erreur_N(t)$ est égale à l'erreur d'apprentissage du réseau ayant une capacité N après t secondes.

La figure 3.2 présente une approximation du ratio des temps du réseau de 200 neurones cachés par rapport à celui de 100 neurones pour atteindre les mêmes taux d'erreur. Ce ratio débute à 1 et croît au début de l'apprentissage. Par la suite, il diminue et peut passer sous le seuil de 1 si la capacité du réseau de référence est insuffisante pour représenter la fonction à apprendre. La figure 3.3 présente le comportement idéalisé de l'évolution du ratio en fonction du nombre de paramètres (N) et en fonction du temps.

[2]R pour capacité de Référence

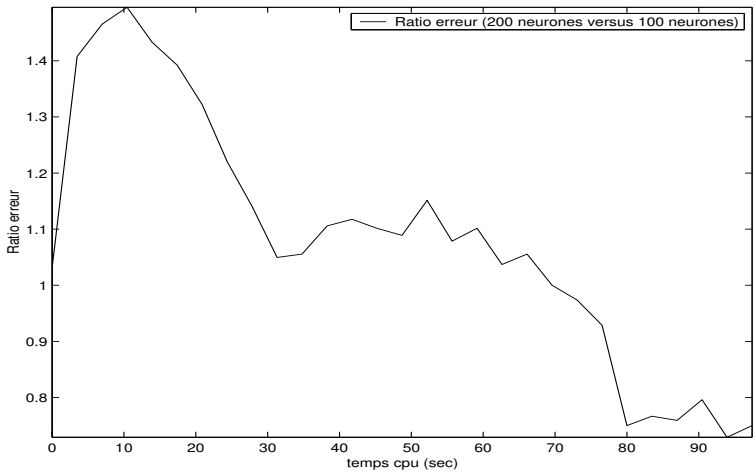

FIG. 3.2 – Ratio erreur(200 neurones cachés, 100 neurones cachés, t)

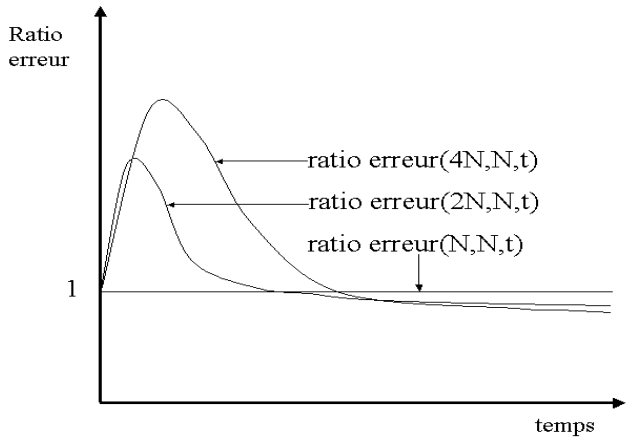

FIG. 3.3 – Comportement observé du ratio erreur

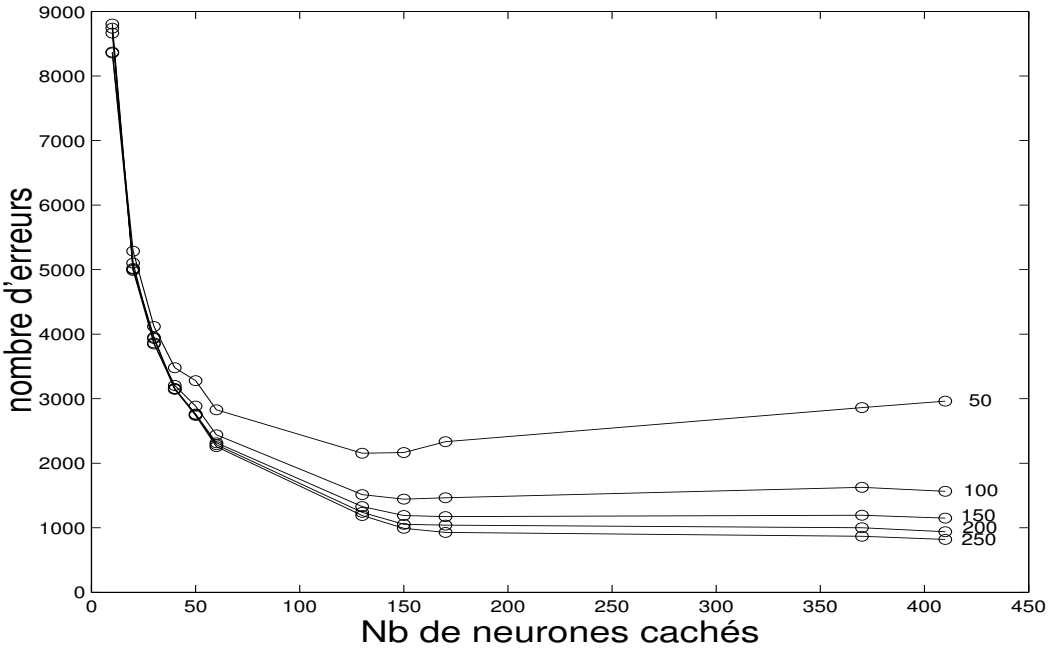

FIG. 3.4 – Comportement typique en fonction de la capacité

3.2 Une augmentation de la capacité ne permet qu'une réduction très faible du nombre d'erreurs

En pratique, nous avons constaté que malgré une augmentation significative de la capacité d'un réseau de neurones, le nombre d'exemples de l'ensemble d'entraînement bien appris n'augmente que très faiblement. Cette amélioration de l'apprentissage est très décevante si l'on tient compte du temps de calcul supplémentaire qu'un réseau de plus grande capacité nécessite pour effectuer chaque époque (voir annexe §I).

Les figures 3.4 et 3.5 nous permettent de constater la difficulté de l'exploitation de la capacité supplémentaire. Les cercles représentent les résultats expérimentaux obtenus sur la base de données «letters»(§5.1) et les lignes représentent les interpolations. En abscisse, nous présentons le nombre de neurones cachés utilisés, soit : 10, 20, 30, 40, 50, 60, 130, 150, 170, 370 et 410. En ordonnée, nous présentons le nombre d'erreurs. Le nombre d'exemples d'entraînement est de 18000. Le nombre d'époques augmente de 50 lorsque nous passons à la ligne inférieure.

En observant ce graphique, on constate qu'une augmentation importante du nombre de neurones n'améliore que très peu le nombre d'exemples bien appris. Il est important de mentionner que nous nous intéressons à l'apprentissage et négligeons l'aspect de généralisation. neurones pour atteindre les mêmes taux d'erreur.

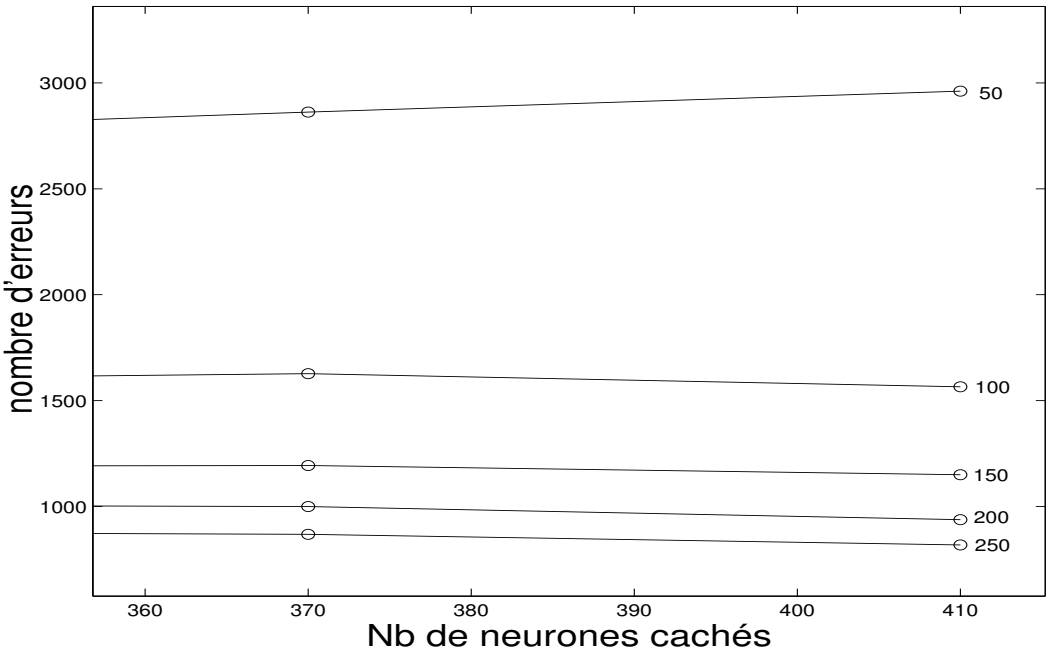

FIG. 3.5 – Agrandissement de la figure 3.4

3.3 Pourquoi l'apprentissage au moyen de la rétro-propagation est-il si lent ?

La limitation la plus importante de la rétro-propagation est la lenteur des pas d'apprentissage à partir d'exemples. Même sur des bancs d'essais simples, il faut plusieurs centaines d'époques pour extraire l'information des exemples. Une époque consiste à traiter tous les exemples de l'ensemble d'entraînement. Les deux problèmes majeurs identifiés par Fahlman et Lebiere sont le *problème de la taille des pas* (§3.3.3) et le *problème du déplacement de la cible* [8](§3.3.4).
Nous avons identifié trois autres problèmes que nous avons appelés respectivement le *problème des gradients contradictoires* (§3.3.5), *le problème de l'inexistence d'un mécanisme de spécialisation* (§3.3.6 et §3.5.1) et le *problème de de symétrie*(§3.5.2).
Dans l'article de Yann LeCun et autres[15], on parle des heuristiques nécessaires pour que la rétro-propagation soit efficace. Voici les points importants :

- Choix de la valeur de la cible[3] en classification avec MSE(§2.2)
- Initialisation des poids (§3.3.1)
- Utilisation d'un pas pour chaque paramètre (§3.3.2)

3.3.1 Choix de la valeur de la cible en classification

En classification lorsqu'on utilise le MSE comme fonction de coût, les valeurs des cibles sont typiquement binaires (e.g.-1,+1). Le bon sens suggère de choisir comme valeur les asymptotes des fonctions d'activation. Toutefois, ce choix a certains inconvénients. Le premier est l'instabilité. Le processus d'entraînement va pousser les sorties du réseau à s'approcher des valeurs des cibles, tout en sachant que cela n'est atteignable qu'asymptotiquement. Comme répercussion, les poids sont poussés vers des valeurs de plus en plus grandes lorsque la dérivée de la fonction d'activation est près de zéro. Les poids de valeurs élevées augmentent le gradient, mais celui-ci est aussi affecté par la multiplication par la dérivée de la fonction d'activation, une valeur exponentiellement petite, produisant un gradient près de zéro. Ceci entraîne les poids à se bloquer à la valeur courante. L'autre problème est que lorsque les sorties saturent, le réseau ne donne aucune indication du niveau de confiance du résultat. Lorsqu'un exemple tombe près de la surface de décision, la classification est incertaine. Idéalement, cela devrait être reflété par des valeurs de sorties importantes pour les 2 classes et non pas près des asymptotes respectives. Toutefois, les poids de valeurs importantes tendent à pousser les sorties vers les extrémum et ce, nonobstant l'incertitude. Ces mêmes poids rendent impossible la distinction entre des exemples

[3]Le terme anglais est «target»

typiques[4] et non-typiques[5]. Une des solutions à ces problèmes est d'utiliser une valeur de cible incluse dans l'intervalle de la fonction d'activation, plutôt que de choisir les valeurs des asymptotes. Il faut faire attention que le choix ne restreigne pas la sortie dans la région linéaire. Un choix judiceux consiste à utiliser le point où la seconde dérivée est nulle. Cela permet de garder l'avantage de la non-linéarité sans saturer la fonction d'activation.

3.3.2 Utiliser un pas pour chaque paramètre

L'utilisation d'un pas pour chaque paramètre permet en théorie de converger plus rapidement. Le choix optimal de ces valeurs peut en principe être fait à partir de l'information de la dérivée seconde (§3.3.3). L'idée est d'assurer une vitesse de convergence uniforme pour chaque poids. Tout dépendant de la courbure de la surface de la fonction de coût, certains poids nécessitent de grands pas et d'autres de plus petits afin de permettre une vitesse de convergence convenable. Malheureusement, un algorithme permettant un gain significatif de cette manière reste à démontrer.

3.3.3 Problème de la taille des pas

Ce problème survient parce que la rétro-propagation n'utilise que l'information de la dérivée première reliée à chaque paramètre. En calculant ces dérivées, nous pouvons effectuer une descente de gradient dans l'espace des poids pour réduire l'erreur à chaque pas. En effectuant un nombre de pas infini en suivant le gradient qui est recalculé après chaque pas, nous finissons par atteindre un minimum local de la fonction d'erreur. En pratique, nous ne sommes pas intéressés à itérer indéfiniment. Pour effectuer un apprentissage plus rapide, nous souhaitons utiliser le plus grand pas possible. Si nous utilisons un pas trop grand, le réseau ne va pas converger vers une solution acceptable. Pour choisir un pas acceptable, nous avons besoin de l'information des dérivées supérieures afin d'avoir des indications sur la courbure de la fonction de coût. Cette information n'est pas disponible à partir de la rétro-propagation standard. Pour simplifier, on peut supposer que tous les poids sont indépendants (non-corrélés). Sachant que la fonction d'erreur de chaque paramètre est différente, il est préférable de choisir un pas pour chacun d'eux. Cela aurait l'avantage d'éliminer le problème de l'apprentissage non-uniforme et permettrait une convergence approximativement à la même vitesse pour chaque poids.
Si les paramètres sont non-corrélés, le *hessien* est nécessairement diagonal alors la manière de trouver le pas optimal est de calculer la dérivée seconde. Dans le contexte

[4]Exemple tiré de l'ensemble d'entraînement ou très similaire
[5]Exemple très différent de tous les exemples de l'ensemble d'entraînement

où la fonction de coût est quadratique et que l'on calcule aussi la dérivée seconde, nous pouvons trouver le pas optimal pour chaque paramètre et converger en une seule époque (Newton algorithm[5]). En pratique, cette solution est inapplicable aux réseaux de grande capacité, car elle nécessite l'inversion d'une matrice N[6] par N ayant une complexité $O(N^3)$ par itération. Sachant qu'il n'y a aucune garantie que la surface d'erreur soit quadratique, la convergence ne peut être assurée[15]. Il existe d'autres techniques, telles que Quickprop[5], une technique moins coûteuse, et le gradient conjugué[5]. La technique de Quickprop consiste à conserver l'information de deux dérivées successives et de trouver les coefficients d'une parabole passant par ces deux points. Par la suite, nous trouvons le point minimum de la parabole. Pour le gradient conjugué, on effectue la première descente de gradient dans la même direction tant que l'erreur diminue[7]. Puis, on effectue la descente de gradient dans une direction «conjuguée» tant que l'erreur diminue. Cette technique ne peut être appliquée qu'en mode «batch».

Les techniques du second ordre nécessitant le calcul du hessien sont beaucoup trop coûteuses en raison de la quantité de paramètres des réseaux que nous voulons entraîner. Tel que mentionné à la section 2.3, nous nous sommes limités au gradient stochastique.

3.3.4 Problème du déplacement de la cible

Lorsqu'on utilise la technique de rétro-propagation, le calcul du gradient sur un paramètre ne tient pas compte des changements des autres paramètres. Le problème est que les poids essaient d'apprendre un problème qui change constamment en raison des autres poids. Cela explique le temps important avant que les poids se spécialisent définitivement sur certaines caractéristiques (voir les figures 4.10 et 4.11). Une manifestation du *problème du déplacement de la cible* est ce qu'on appelle l'effet de troupeau. Supposons que nous ayons deux classes, chaque unité décide de façon indépendante sur quel problème elle va réduire l'erreur. En pratique, tous les paramètres vont être modifiés pour réduire l'erreur. Ce phénomène prend de l'importance à mesure qu'augmente le nombre de paramètres.

3.3.5 Problème des gradients contradictoires en classification

Ce problème survient en classification. Lors du calcul des facteurs de *sensibilité* ($\delta_j = [\sum_{k=1}^{c} w_{kj} \cdot \delta_k] \cdot f'(net_j)$) associés à chaque neurone caché, les apports des différents neurones de sortie peuvent être du même signe ou être contradictoires (signe

[6]Nombre de paramètres

[7]Le terme anglais pour désigner cette technique est le «line search»

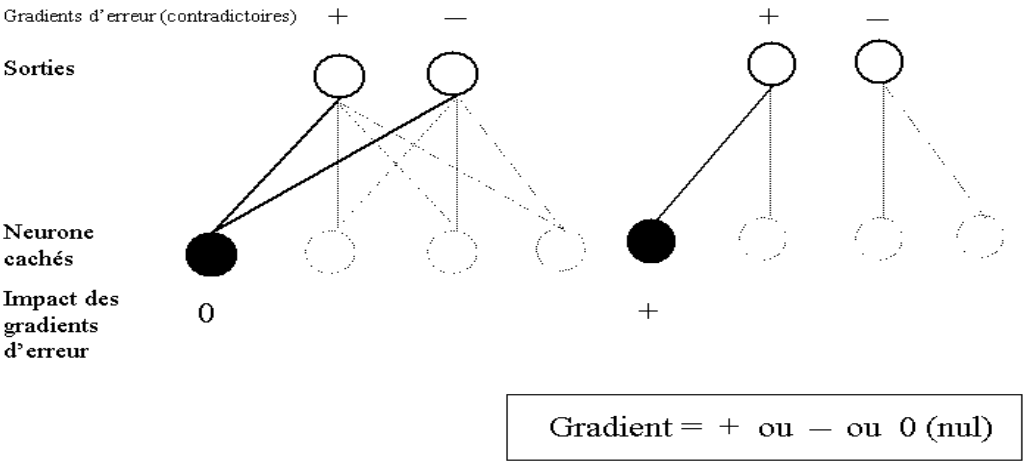

FIG. 3.6 – Problème des gradients contradictoires d'un réseau totalement interconnecté vs un réseau découplé

opposé). Lorsque les valeurs sont contradictoires, la magnitude du gradient sera réduite ou à l'extrême nulle. Lorsqu'elles sont du même signe, la magnitude du gradient sera augmentée. Cette augmentation ou diminution se compare à une augmentation ou une diminution du pas pour tous les paramètres dont le gradient dépend de ce δ. Donc, plus il y a de classes, plus cet effet est potentiellement important. Sachant que nous n'utilisons que l'information de la dérivée première, faire un pas trop court ou trop long dans la direction du gradient n'est pas souhaité et ne peut qu'entraîner un ralentissement de l'apprentissage.
Si l'on prend la perspective des plans séparateurs (§2.5), on pourrait imaginer que ceux-ci soient poussés à se déplacer dans des directions opposées pour satisfaire une diminution du coût de classification associé à des classes distinctes (voir figure 3.6).
Habituellement, nous connectons tous les neurones cachés aux neurones de sortie afin de partager l'information des neurones cachés. Ce partage d'information entraîne la création d'un modèle plus général. Sachant que nous nous intéressons aux problèmes de grande capacité, le *surapprentissage* n'est pas vraiment important, car il est difficilement atteignable. Donc, l'utilisation d'une architecture entraînant la construction d'un modèle plus général n'est pas très importante. En utilisant des neurones spécifiquement pour chaque classe (architecture découplée), nous utilisons un modèle

moins général, mais nous éliminons le *problème des gradients contradictoires*. Cette approche a été expérimentée et les résultats sont présentés à la section 5.5. En résumé, le *problème des gradients contradictoires* perturbe le processus d'optimisation des neurones cachés et donc ralentit l'apprentissage.
Il est important de mentionner que pour un nombre de paramètres identiques, une architecture totalement interconnectée permet de représenter un nombre de solutions supérieures à une architecture découplée en raison du partage d'informations au niveau des neurones cachés.

3.3.6 Problème de l'inexistence d'un mécanisme de spécialisation

Dans le contexte des réseaux de neurones, un mécanisme de spécialisation consiste à n'optimiser qu'une partie des paramètres selon un certain critère.
Tel que mentionné à la section 2.3 L'algorithme d'apprentissage consiste à propager les valeurs des caractéristiques d'un exemple des entrées jusqu'aux sorties, d'appliquer la fonction de coût et de rétro-propager l'erreur. N'ayant aucun mécanisme de spécialisation des paramètres, il est nécessaire de refaire la propagation à travers tous les paramètres. Dans le cas contraire, pour la propagation dans les poids qui ne seront pas optimisés, il est possible de la faire qu'une fois et de conserver les résultats (§4.3.10). En conservant ces résultats, nous pouvons optimiser le mécanisme de propagation. De plus, la rétro-propagation n'a besoin d'être faite que sur les paramètres choisis. Bref, avec un mécanisme de spécialisation, la propagation et la rétro-propagation sont beaucoup moins coûteuses en nombre d'opérations et, contrairement à l'algorithme standard, les paramètres se spécialisent.

3.4 Pourquoi la vitesse d'apprentissage se détériore t-elle à mesure qu'augmente la capacité d'un réseau ?

Tel que présenté dans la section 3.1, la vitesse maximale d'apprentissage d'un réseau diminue à mesure que la capacité est augmentée. Plusieurs chercheurs ont identifié certaines raisons expliquant ce comportement, telles que la plus grande importance du *problème du déplacement de la cible* et *le problème de l'atténuation et de la dilution du gradient* lorsqu'on le propage dans les couches de neurones inférieures. Le *problème du déplacement de la cible* et le *problème des gradients contradictoires* sont des problèmes qui prennent de l'importance à mesure que la capacité augmente. *Le problème des gradients contradictoires* est aggravé à cause de l'augmentation du déplacement de la cible.

3.4.1 Atténuation et dilution du gradient

Tel que mentionné dans la section 2.2, les valeurs initiales des poids entre la couche cachée et la couche de sortie sont choisis en fonction du nombre de neurones cachés (équation 3.2). Plus ce nombre est élevé, plus le gradient sera dilué au niveau des gradients associés aux paramètres situés entre la couche d'entrée et la couche de neurones cachés.

$$w_{kj} = U \sim (0, \sigma_w), \sigma_w = h^{-1/2} \tag{3.2}$$

où h est le nombre de neurones cachés.
Cela s'explique ainsi : plus le nombre de neurones cachés est élevé, plus la variance (σ_w) est petite. Cela implique qu'à l'initialisation, tous les w_{kj} seront en moyenne plus petits. Donc, les δ_j seront en moyenne plus petits lorsque le nombre de neurones est plus élevé, entraînant une diminution en moyenne de tous les gradients entre la couche d'entrée et la couche de sortie (voir les équations 3.3 et 3.4).

$$\delta_j = [\sum_{k=1}^{c} w_{kj} \cdot \delta_k] \cdot f'(net_j) \tag{3.3}$$

$$\nabla w_{ij} = n \cdot \delta_j \cdot x_i \tag{3.4}$$

3.5 Pourquoi une augmentation de la capacité ne permet-elle qu'une réduction très faible du nombre d'erreurs ?

En pratique, au delà d'un certain seuil, nous constatons qu'un réseau de capacité accrue exploite difficilement cette capacité supplémentaire(§3.2). Nous croyons que deux raisons expliquent plus particulièrement ce comportement, soit *l'inexistence d'un*

mécanisme permettant une spécialisation des paramètres et *le problème de symétrie.* Ces deux raisons seront présentées séparément aux deux prochaines sous-sections.

3.5.1 Problème de l'inexistence d'un mécanisme de spécialisation des paramètres

Contrairement aux techniques de «boosting»[20] et aux mixtures d'experts[2], l'apprentissage standard d'un réseau de neurones n'a aucun mécanisme pour spécialiser les paramètres sur certaines caractéristiques ou certaines régions de caractéristiques. Ce mécanisme de spécialisation permet d'appliquer la stratégie de *diviser pour régner.* les avantages de cette stratégie sont détaillés à la section 4.1.1. N'exploitant pas cette technique, le problème est plus difficile et l'apprentissage ne peut qu'être plus lent et moins efficace.

3.5.2 Problème de symétrie

Sachant qu'avant le début de l'apprentissage, tous les paramètres d'un réseau de neurones sont initialisés au hasard et que les gradients sont calculés de façon indépendante d'un neurone à l'autre, il est possible d'imaginer que plusieurs neurones fassent un apprentissage similaire. Cette similarité est en fait inutile et entraîne une mauvaise exploitation de la capacité réel du réseau de neurones. L'augmentation de capacité durant l'apprentissage ou l'optimisation seule d'une partie des paramètres permet de réduire ce problème.

3.6 Est-il possible qu'une capacité accrue accélère l'apprentissage ?

Contrairement à ce qui est observé en pratique, nous croyons qu'une capacité accrue peut accélérer l'apprentissage. Du point de vue de l'espace des solutions, un espace plus grand contient plusieurs solutions équivalentes à une solution d'un espace inférieur. Il est facile d'imaginer qu'une solution hautement non-linéaire d'un espace réduit puisse être équivalente à une solution quelque peu non-linéaire d'un espace supérieur. Sachant que le processus itératif d'apprentissage devient très lent lorsqu'on tente d'apprendre les non-linéarités, il est possible qu'il prenne moins d'époques pour atteindre une solution au moyen d'un réseau de plus grande capacité. Malgré le fait qu'une itération prenne plus de temps pour un réseau de plus grande capacité, il est tout à fait possible d'imaginer qu'un réseau de plus grande capacité puisse atteindre une solution plus rapidement en temps de calcul. La figure 3.7 présente le concept exposé.

Dans la figure de gauche, on représente l'espace des solutions d'un réseau de N et de (N+K) neurones. Dans la figure de droite, on représente au moyen de flèches

FIG. 3.7 – Espaces de solution et exemple de solution

l'évolution de l'état initial après chaque époque pour atteindre une solution. Par exemple, dans l'espace de solutions d'un réseau de N neurones, il faut 5 époques pour atteindre une solution au problème. Dans l'espace des solutions d'un réseau de N+K neurones, on y trouve 5 solutions équivalentes supplémentaires. Dans cet espace de plus grande taille, il est possible qu'une des solutions puisse être atteinte en moins d'époques, soit 3 époques au lieu de 5 époques tel que présenté dans l'exemple de la figure 3.7. Bref, contrairement à ce qui est observé en pratique, nous croyons que l'apprentissage devrait être plus rapide lorsqu'un réseau de neurones a initialement une capacité supérieure.

3.7 Hypothèse expliquant l'inefficacité des réseaux de grande capacité

Pour simplifier l'énoncé de notre hypothèse expliquant l'inefficacité des réseaux de grande capacité, nous avons regroupé sous l'expression de *problèmes de la rétro-propagation* : *le problème de déplacement de la cible*, *le problème des gradients contradictoires*, *le problème de la taille des pas*, *le problème d'atténuation et de dilution du gradient*, *le problème d'inexistence d'un mécanisme de spécialisation* et *le problème de symétrie*. Le *problème de la taille des pas* n'a pas été inclus à cette liste car il est implicite à l'algorithme de rétro-propagation standard. Nous croyons que l'inefficacité des réseaux de grande capacité est due à l'accroissement des *problèmes de la rétro-propagation* plus la capacité du réseau est grande. Nous croyons que l'atténuation ou l'élimination d'un ou plusieurs de ces problèmes devrait permettre d'accélérer l'apprentissage d'un réseau de grande capacité et de mieux exploiter la capacité.

CHAPITRE 4

APPROCHES ET SOLUTIONS

Premièrement, ce chapitre présente les approches que nous avons étudiées et évaluées pour accélérer l'apprentissage des réseaux de grande capacité. Deuxièmement, nous présentons les solutions[1] particulières que nous avons explorées et leurs impacts(§4.7) sur *les problèmes de la rétro-propagation* soit : le *problème du déplacement de la cible*, *le problème des gradients contradictoires*, *le problème de l'atténuation et dilution du gradient*, *le problème de symétrie*, *le problème de l'inexistence d'un mécanisme de spécialisation.*
Les impacts seront traités dans la perspective où l'on compare l'importance du problème de la solution par rapport à un réseau standard. À l'état final, les deux réseaux doivent avoir la même capacité. Sachant que nos solutions, à l'exception des techniques reliées à l'optimisation d'une partie des paramètres, modifient l'espace de solution durant l'entraînement, nous n'avons aucune certitude que l'atténuation ou l'élimination de certains *problèmes de la rétro-propagation* puisse réellement accélérer le processus d'apprentissage ou de permettre une meilleure exploitation de la capacité. En ce qui concerne les techniques reliées à l'optimisation d'une partie des paramètres, nous n'avons aucune certitude que l'atténuation de certains des *problèmes de la rétro-propagation* puisse accélérer l'apprentissage car le nombre d'époques pour atteindre les mêmes taux d'erreur pourrait être supérieur. Par contre, cette technique devrait permettre une meilleure exploitation de la capacité.
Pour conclure ce chapitre, nous présenterons une synthèse des impacts des différentes solutions sur *les problèmes de la rétro-propagation.*

4.1 Approches

Les quatre approches identifiées pour s'attaquer au problème d'inefficacité d'apprentissage des réseaux de neurones de grande capacité, sont :

1. Diviser pour régner (§4.1.1)
2. Optimiser une partie des paramètres à la fois (§4.1.2)
3. Prédiction des valeurs des paramètres (§4.1.3)
4. Choix de l'architecture (§4.1.4)

[1]Une solution est une technique pour réaliser une approche

Avant de présenter une description détaillée des quatre approches dans des sous-sections distinctes, nous parlerons de l'importance des capacités de parallélisme des approches.

Considération de parallélisme

L'apprentissage d'un réseau de neurones standard au moyen du gradient stochastique est assez complexe à paralléliser. Cette technique implique des transferts d'information et de la synchronisation. Il est clair qu'une solution, dont la version parallèle serait moins contraignante, aurait de gros avantages dans le contexte où nous avons accès à des grappes d'ordinateurs[17]. Cette solution permettrait d'accélérer d'avantage l'apprentissage. Sachant que les grappes d'ordinateurs sont de plus en plus répandus et que nous avons accès à plusieurs d'entre elles, nous avons ajouté une indication sur les possibilités de parallélisme des différentes approches au tableau 4.2.

4.1.1 Diviser pour régner

La technique la plus intuitive est de fragmenter le problème d'apprentissage en sous-problèmes moins complexes. Cette fragmentation permet de spécialiser les différents paramètres. En général, il est plus difficile d'apprendre globalement un problème que les sous-problèmes le constituant. Cette stratégie fondamentale de l'algorithmique porte le nom de «diviser pour régner». Cette stratégie réduit le nombre de paramètres à optimiser ainsi que le temps requis pour faire une itération.

Plusieurs approches utilisent directement ou indirectement cette stratégie, dont les mixtures[2], les techniques de «boosting»[20], les systèmes hiérarchiques[10] et les systèmes incrémentaux. Avec une mixture d'experts, chaque expert se spécialise sur une sous-région de l'espace des données.

En ce qui concerne le «boosting», l'approche consiste à entraîner un réseau de faible capacité et à utiliser les données qui n'ont pas bien été apprises par un autre réseau. Une récursivité est appliquée tant qu'on le désire. Les résultats des différents réseaux spécialisés sont pondérés en fonction de leur erreur respective. Contrairement aux mixtures d'experts, les techniques de boosting ne sont pas parallélisables. Par contre, le nombre d'exemples utilisés pour entraîner les nouveaux modèles diminue.

En ce qui concerne les réseaux incrémentaux, l'approche consiste à commencer avec un réseau de faible capacité, puis à y ajouter des paramètres lorsque l'erreur d'apprentissage ne s'améliore plus. Plusieurs articles nous poussent à croire qu'il pourrait être avantageux d'utiliser des réseaux incrémentaux[4, 6, 12, 13, 18, 19, 22, 23]. L'article «Trading generalisation for learning efficiency ?»[21] analyse l'aspect de généralisation des réseaux constructifs. À l'heure actuelle, le «boosting» et les mixtures sont deux techniques qui sont assez bien étudiées. Par contre, plusieurs variantes de réseaux de neurones incrémentaux n'ont jamais été expérimentées ou leurs comportements n'ont jamais été présentés.

Les dernières recherches les plus concluantes sur les réseaux incrémentaux avec optimisation d'une partie des paramètres ont été réalisées en 1991 sous le nom de «cascade-correlation» [8]. En général, cette dernière approche permet un apprentissage rapide. Les deux idées principales de cet algorithme sont l'architecture en cascade et une maximisation de la corrélation entre les nouveaux neurones et l'erreur résiduelle. L'architecture en cascade permet d'ajouter récursivement un neurone lorsque le poids du neurone courant ne change plus. L'algorithme d'apprentissage permet de maximiser la corrélation entre les nouveaux neurones et l'erreur résiduelle.

4.1.2 Optimiser une partie des paramètres

Cette approche est apparentée aux techniques d'élagage[2] [7, 13]. Lorsqu'on élague un réseau de neurones, on élimine les connexions inutiles ou peu influentes pour réduire le nombre de calculs à chaque itération. Les intérêts principaux de cette technique consistent à introduire un mécanisme de spécialisation des paramètres ainsi qu'à réduire le nombre de calculs pour la descente de gradient en ne l'effectuant que sur certains paramètres. Par exemple, pour réduire le temps d'apprentissage, il pourrait être intéressant d'optimiser les paramètres reliés aux neurones les plus influents (§4.4.1).

4.1.3 Prédiction des valeurs des paramètres

Tel que mentionné à la section 2.3, les paramètres[3] du réseau de neurones sont modifiés durant le processus itératif de l'apprentissage. Il pourrait être intéressant d'utiliser l'historique des valeurs de différents paramètres pour prédire leur évolution. Cette technique pourrait permettre de réduire le temps de calcul en réduisant le nombre d'époques. Cette technique n'a aucune répercussion sur *les problèmes de la rétro-propagation.*

4.1.4 Choix de l'architecture

Cette approche a été inspirée du *problème des gradients contradictoires.* Plus le nombre de sorties d'un réseau est élevé, plus le calcul des gradients au niveau des neurones de sortie peut être perturbé et donc, plus l'apprentissage peut devenir difficile. Une solution à ce problème consiste à modifier l'architecture du réseau en ne permettant qu'une connexion entre les neurones cachés et les neurones de sorties. Donc, l'architecture peut avoir une incidence sur *les problèmes de la rétro-propagation.*

[2] Élimination des connexions inutiles, («pruning» en anglais

[3] Les paramètres sont les poids synaptiques et les biais

4.2 Solutions

Le tableau 4.1 nous énumère les différentes solutions que nous allons expérimenter et les approches qui leur sont associées.

Approches	Solutions
Diviser pour régner	Réseaux incrémentaux
Opt. partie params	Optimisation circulaire et gradients élevés
Prédiction	Régréssion linéaire, extrapolation quadratique et cubique
Différente architecture	Architecture découplée

TAB. 4.1 – Solutions

4.3 Réseaux incrémentaux

Tel que mentionné dans la section 4.1.1, certaines variantes de réseaux incrémentaux n'ont pas encore été explorées. Les solutions que nous explorerons sont :

- L'optimisation uniquement des paramètres associés aux neurones ajoutés durant l'entraînement.
- L'optimisation des paramètres associés aux neurones ajoutés durant l'entraînement et ceux de la couche de sortie.
- L'optimisation de tous les paramètres.

Les résultats expérimentaux se trouvent à la section 5.2. Tout d'abord, le présent chapitre présente la définition d'un réseau de neurones incrémental, ses intérêts et les différentes approches pour ajouter des neurones. Par la suite, nous verrons une heuristique sur la courbure de la fonction de coût selon la position des paramètres et la problématique qui y est reliée. La troisième partie traitera de l'initialisation des nouveaux paramètres et du calcul du gradient sur les nouveaux paramètres. Pour finir, nous présenterons les résultats concluants d'une première expérience, une considération d'efficacité et un des algorithmes expérimentés pour ajouter automatiquement de la capacité.

4.3.1 Définition

Un réseau de neurones incrémental est un réseau auquel nous ajoutons des neurones durant l'entraînement. En général, nous n'optimisons que les nouveaux paramètres, mais nous nous sommes aussi intéressés au comportement lorsque nous optimisons tous les paramètres, les anciens et les nouveaux.

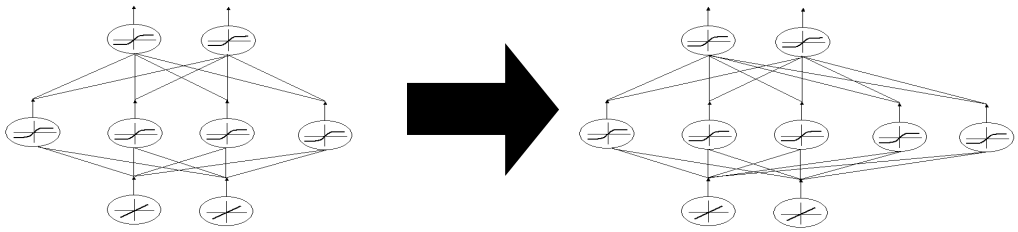

FIG. 4.1 – Ajout d'un plan séparateur au niveau de la couche cachée

4.3.2 Intérêts

Un des avantages de l'approche incrémentale est de trouver automatiquement la capacité idéale sachant que le choix optimal du nombre de neurones reste un problème ouvert[14,16,18,23]. Dans le contexte des réseaux de grande capacité, la possibilité d'une réduction significative du temps nécessaire à l'exécution d'une itération et la possibilité de spécialiser les paramètres sont les motivations principales.

4.3.3 Approches pour ajouter des neurones

Il existe différentes approches permettant d'augmenter la capacité d'un réseau de neurones au cours de l'apprentissage et ce, pour réaliser un réseau incrémental. Une première approche consiste à ajouter de nouveaux plans séparateurs au moyen de nouveaux neurones cachés (figure 4.1). L'acronyme que nous avons donné à un réseau incrémental qui utilise cette approche est RIPS [4]. La seconde approche consiste à ajouter des couches de nouveaux neurones entre la dernière couche cachée et la couche de sortie (figure 4.2). L'acronyme que nous avons donné à un réseau incrémental qui utilise cette approche et RICC. La dernière approche consiste à ajouter un neurone que l'on connecte entre les entrées et la couche de sortie (figure 4.3). Au cours des époques subséquentes, le nouveau neurone est considéré comme une entrée. Cette approche est inspirée de l'apprentissage par «cascade-correlation» [8]. Cette approche n'a pas été explorée.

4.3.4 Courbure de la fonction de coût

Il est important de garder à l'esprit que la courbure[15] de la fonction de coût ou la magnitude des gradients dépend de la position des paramètres dans le réseau de

[4] Réseau Incrémental Plans Séparateurs

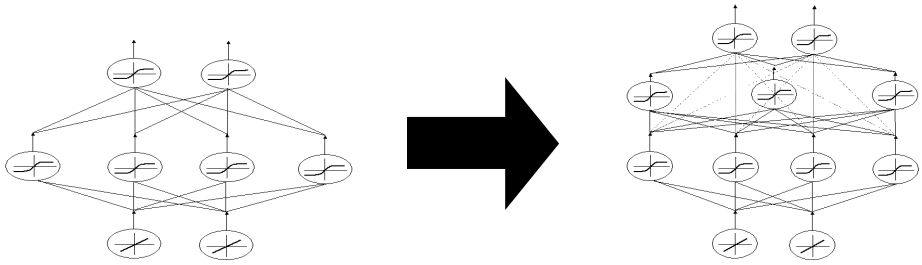

FIG. 4.2 – Ajout d'une couche cachée

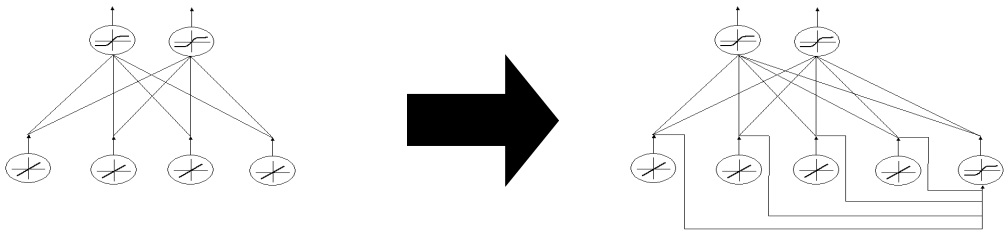

FIG. 4.3 – Perceptron - Configuration «Cascade Correlation»

neurones. Cette courbure diminue plus on se rapproche des entrées du réseau tel que présenté à la figure 4.4 où J représente la fonction de coût et w un poids synaptique [15]. Cette observation peut s'expliquer parce que l'influence des paramètres inférieurs est réduite lorsque les neurones sont activés dans les régions non-linéaire. De plus, l'importance de l'influence dépend des paramètres supérieures. En pratique, il est préférable de mettre un pas plus élevé pour les paramètres associés aux couches inférieures. Cela permet d'accélérer l'apprentissage, mais ne change en rien la courbure. Lorsque la courbure est faible, il devient difficile d'optimiser les paramètres associés aux couches inférieures car les gradients sont très faibles. Cette observation apporte un intérêt pour comparer l'évolution de l'apprentissage d'un réseau de plusieurs couches cachées avec un réseau incrémental où l'on y ajoute des couches cachées. En ajoutant les couches cachées durant l'apprentissage, le problème de la courbure pourrait être moins important pour des réseaux incrémentaux.

4.3.5 Problématique de l'initialisation des poids

Le choix de l'initialisation des nouveaux paramètres peut perturber la fonction de coût et ce n'est pas souhaité. Dans le cas où l'architecture d'un réseau est fixe, tous les poids sont initialisés de façon aléatoire avant l'entraînement(§2.2). Dans le contexte

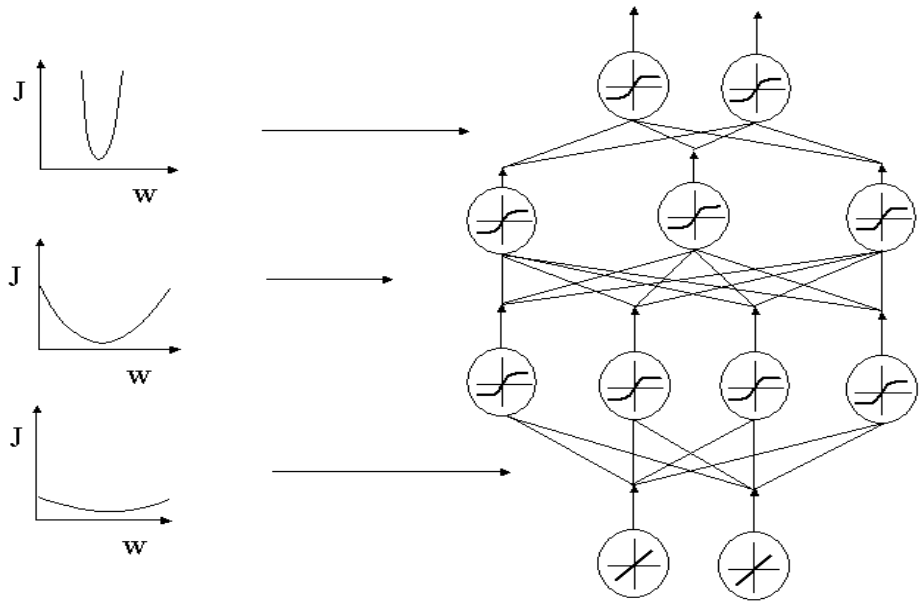

FIG. 4.4 – Courbure de la fonction de coût

des réseaux incrémentaux, l'ajout de capacité est effectué lorsque celle-ci est jugée insuffisante pour apprendre le problème. L'initialisation complètement au hasard des nouveaux poids ajoutés au cours de l'entraînement introduit l'ajout d'une erreur supplémentaire qui nuirait à l'objectif. L'erreur ainsi introduite créerait une perturbation de la fonction de coût. Donc, il serait inefficace et contradictoire que l'ajout de nouvelle capacité entraîne par la même occasion l'introduction d'une erreur supplémentaire. Une telle initialisation modifierait le coût et entraînerait une perturbation du réseau initial. Par exemple, si nous faisons une descente de gradient sur tous les paramètres, l'erreur supplémentaire engendrée par les nouveaux paramètres affecterait tous les paramètres du réseau et ne pourrait qu'entraîner des comportements étranges sur l'évolution de la fonction de coût.

4.3.6 Choix d'initialisation des nouveaux paramètres

La solution évidente consiste à initialiser au hasard tous les nouveaux paramètres à l'exception des poids et biais associés aux connexions des neurones de sortie qui eux seront initialisés à zéro. L'initialisation à zéro de ces paramètres assure que les nouveaux neurones n'entraînent pas une modification du coût (perturbation du coût). L'initialisation au hasard des poids d'entrée nous assure que le calcul du gradient sur

tous les nouveaux paramètres sera différent de zéro. Les poids d'entrées seront initialisés au moyen d'une valeur aléatoire tirée d'une distribution uniforme de moyenne 0 et de variance 1 (§2.2).

4.3.7 Calcul du gradient sur les nouveaux paramètres

Considérant que le calcul des gradients sur les paramètres associés à un neurone caché est indépendant des autres neurones cachés, il est donc possible d'ajouter de nouveaux neurones en utilisant les mêmes équations pour calculer les gradients. Voici les équations pour calculer les gradients dans un réseau de neurones à une couche cachée :

$$\delta_k = (\frac{\partial J}{\partial z_k}) \cdot f'(net_k) \tag{4.1}$$

$$\delta_j = [\sum_{k=1}^{c} w_{kj} \cdot \delta_k] \cdot f'(net_j) \tag{4.2}$$

$$\nabla w_{ij} = \delta_j \cdot x_i \tag{4.3}$$

$$\nabla w_{kj} = \delta_k \cdot y_j \tag{4.4}$$

Dans le contexte où une erreur persiste, si nous initialisons les paramètres des nouveaux neurones tel que mentionné à la section précédente, les gradients sur les nouveaux paramètres (équations 4.3 4.4) seront nécessairement différents de zéro. Il nous est donc théoriquement possible de faire une descente de gradient sur tous les paramètres (gradient différent de 0) ou seulement sur les nouveaux paramètres. Dans la section suivante (4.3.8), nous vérifierons expérimentalement que le coût continue bel et bien à diminuer lorsque nous ne faisons que la descente de gradient sur les nouveaux paramètres.
Dans le cas où nous serions dans un minimum local et que nous optimiserions tous les paramètres, une perturbation(§4.3.5) du coût pourrait être désirable, mais cette direction de recherche n'a pas été explorée profondément. Dans le cas où nous n'optimiserions que les nouveaux paramètres, il serait préférable que la perturbation soit nulle afin que la nouvelle capacité serve le plus possible à réduire l'erreur résiduelle et non l'erreur qu'elle a introduite. De plus, dans ce contexte, la perturbation n'aurait aucun impact sur les gradients des anciens paramètres parce que la descente de gradient n'est pas faite sur ceux-ci.

4.3.8 Validation

La figure 4.5 montre qu'il est possible d'ajouter des neurones durant l'entraînement et de continuer à réduire l'erreur, sans l'avoir perturbée, lorsque nous ne faisons que

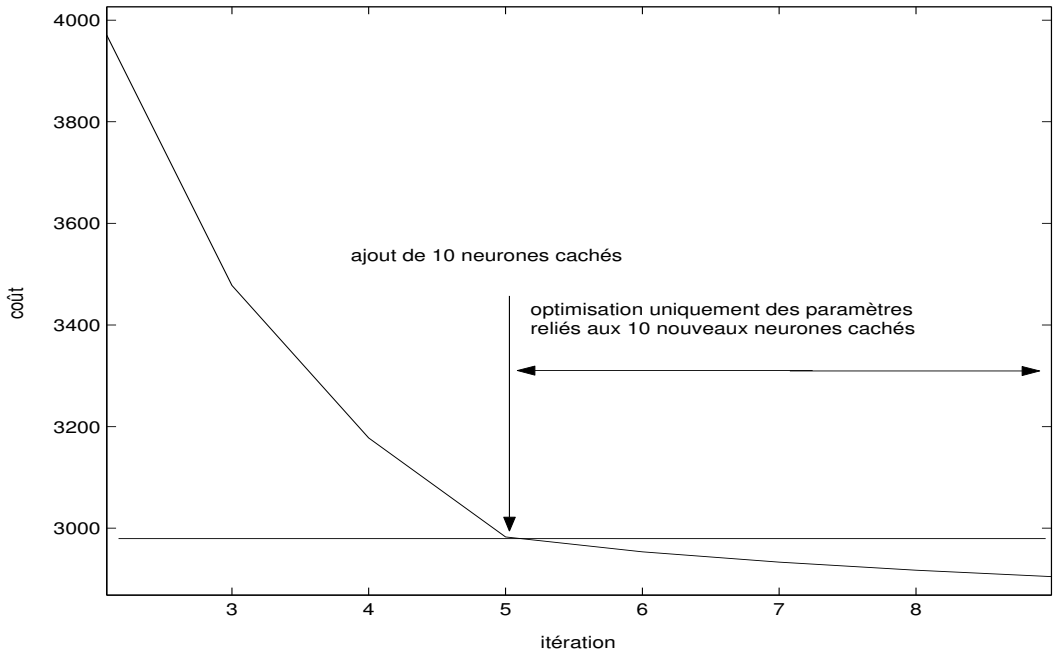

FIG. 4.5 – Optimisation des nouveaux paramètres

la rétro-propagation sur les paramètres associés à ces nouveaux neurones. À l'époque 5, nous ajoutons 10 neurones et désactivons l'optimisation des anciens neurones.

4.3.9 Algorithme

Dans le but d'automatiser l'ajout de nouvelle capacité, il nous faut définir un mécanisme nous permettant de décider de l'ajout de nouveaux neurones selon un certain critère. Notre approche consiste à modifier le critère d'arrêt de l'apprentissage de l'algorithme du gradient stochastique (algorithme modifié : figure 4.7 vs algorithme standard : figure 2.3). Un des algorithmes que nous avons implanté consiste à ajouter de la nouvelle capacité lorsque la réduction du coût n'a pas diminué au minimum d'un certain facteur durant l'époque subséquente.

4.3.10 Considération d'efficacité

Considérant un ensemble d'entraînement de taille fixe et une descente de gradient sur les paramètres associés aux nouveaux neurones cachés, il est inefficace de propager plus d'une fois des valeurs dans les neurones dont les poids ne sont pas optimisés. Pour faire une descente de gradient, il faut initialement propager les valeurs d'entrée

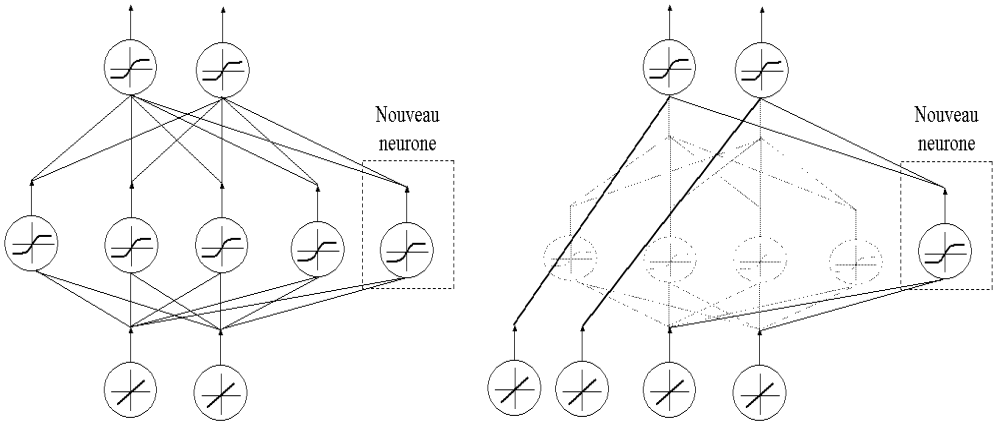

FIG. 4.6 – Optimisation pour la propagation lorsque certains des paramètres sont fixes

du réseau. Sachant que nous itérons normalement plusieurs fois sur l'ensemble d'entraînement, il est beaucoup plus efficace de propager les entrées une seule fois dans les neurones de poids non optimisés[5] et de conserver ces résultats. L'implantation de cette approche est primordiale pour faire des comparaisons en terme de performance en temps de calcul. Tel que montré dans l'annexe 1, la propagation occupe une proportion très importante du temps de l'apprentissage (voir annexe II). Pour un telle configuration, l'algorithme incrémental efficace est présenté à la figure 4.9.

[5]Les poids non optimisés sont nécessairement fixes

Initialisation n, Θ, ρ, i=0, e=0, m=0
Faire e=e+1 (pour toutes les epoques)
  Faire m=m+1 (pour tous les exemples)
    x=exemple choisi au hasard
    propager les valeurs x_1 a x_d
    calculer les δ_k selon la fonction de coût
    calculer les $\nabla w_{jk} = \delta_k \cdot y_j$
    calculer les $\delta_j; \delta_j = [\sum_{k=1}^{c} w_{kj} \cdot \delta_k] \cdot g'_y(y_j)$
    calculer les $\nabla w_{ij} = \delta_j \cdot x_i$
    $w_{ij} \leftarrow w_{ij} - n \cdot \nabla w_{ij}$
    $w_{jk} \leftarrow w_{jk} - n \cdot \nabla w_{jk}$
Tant que e < nombre d'epoques ou $\frac{\|\nabla J_{e-1}\| - \|\nabla J_e\|}{\|\nabla J_{e-1}\|} < \Theta\%$
return e

FIG. 4.7 – Algorithme du gradient stochastique modifié

Initialisation i (nombre maximum d'increment), n (nombre de neurone cache a ajouter)
Faire i=i+1
  executer algo (figure 4.7)

FIG. 4.8 – Algorithme incrémental

Initialisation i (nombre maximum d'increment), n (nombre de neurones caches a ajouter)
Faire e=e+1
  executer algo (4.7)
  creer une nouvelle base de donnees
  deconnecter les anciens neurones
  creer c nouvelles entrees
  connecter les nouvelles entrees aux sorties

FIG. 4.9 – Algorithme incrémental efficace

4.3.11 Impacts sur les différents problèmes

Seuls les réseaux incrémentaux par ajout de neurones au niveau de la couche cachée et les réseaux incrémentaux par ajout de couches cachées ont été explorées. Pour ces deux techniques, certains impacts sur *les problèmes de la rétro-propagation* sont différents et seront donc traités séparément.
En débutant l'apprentissage par un réseau plus petit, il y a nécessairement une réduction du *problème de déplacement de la cible* et une réduction *du problème d'atténuation et de dilution du gradient*. Cette approche n'a aucune influence sur le *problème des gradients contradictoires* et crée un *mécanisme de spécialisation*. Cette solution réduit *le problème de symétrie* car des paramètres sont ajoutés durant l'apprentissage. Cette approche ne facilite pas la *parallélisation* de l'apprentissage.

4.4 Optimisation d'une partie des paramètres

Nous avons exploré diverses méthodes où l'on optimise qu'une partie des paramètres sans avoir préalablement calculé les gradients pour tous les paramètres. Sachant que dans notre librairie de réseau de neurones «fLayers», les paramètres sont regroupés par connexion[6], il est plus simple d'activer ou de désactiver la rétro-propagation sur ceux-ci. Les 3 techniques explorées sont les suivantes :

- Optimisation des groupes de connections pour lesquels les gradients sont les plus élevés (ROGE)
- Optimisation circulaire (ROC)
- Optimisation des paramètres associés aux neurones de sortie affectant le plus l'erreur de classification (ROEC)

Les résultats expérimentaux se trouvent à la section (5.4). Sachant que ces trois approches sont basées sur le même principe, les impacts sur *les problèmes de la rétro-propagation* seront semblables et seront donc traités dans la même section. Ces trois techniques seront détaillées séparément dans les 3 prochaines sous-sections.

4.4.1 Optimisation des paramètres ayant les gradients les plus élevés

De façon analytique, la seule manière d'identifier les connexions sur lesquels les gradients seront les plus élevés, sans avoir préalablement calculé tous les gradients, consiste à identifier la sortie la plus élevée des neurones cachés et le facteur de sensibilité le plus élevé du neurone caché. La sortie la plus élevée du neurone caché permet d'identifier la connexion entre le neurone caché et les sorties qui influence le plus le coût. Le facteur de sensibilité le plus élevé du neurone caché permet d'identifier la connexion entre les entrées et un neurone caché qui influence le plus le coût. Le prochain paragraphe explique à partir des équations de la rétro-propagation les raisons sous jacentes au mécanisme de sélection des connexions tel que présenté ci-haut.
Les gradients dépendent des facteurs de *sensibilité*, des entrées et des sorties de neurones cachés ($\nabla w_{ij} = n \cdot \delta_j \cdot x_i, \nabla w_{kj} = n \cdot \delta_k \cdot y_j$). Nous ne considérons pas la grandeur du pas, car c'est une constante. Le premier critère qui nous vient à l'esprit est d'optimiser les paramètres associés au neurone ayant le facteur de *sensibilité* le plus élevé. Ce choix permet d'optimiser les paramètres ij pour lesquels les gradients seront le plus élevé, mais ce n'est pas le cas pour les paramètres kj. Le choix optimal pour les poids kj est celui associé au neurone ayant la sortie la plus élevée (y_i).

[6]Exemple de connexion : synapses reliant les entrées au neurone caché i

4.4.2 Optimisation circulaire

L'optimisation circulaire consiste à rétro-propager l'erreur seulement sur les paramètres d'un neurone caché à la fois et à choisir le neurone adjacent pour la prochaine époque. Ce choix est arbitraire et a été motivé par sa simplicité d'implantation. Après avoir choisi de façon séquentiel chaque neurone, nous continuons l'optimisation en sélectionnant le premier neurone, d'où le nom d'optimisation circulaire.

4.4.3 Optimisation des paramètres affectant le plus l'erreur de classification

Tel que mentionné à la section 2.4, il n'est pas possible de minimiser l'erreur de classification. Par contre, le MSE et le LogSoftMax sont deux fonctions de coût pour lesquelles la minimisation est possible. Certes, la minimisation d'un de ces deux critères permet indirectement de réduire l'erreur de classification mais est-ce que l'erreur d'entraînement pourrait être réduite plus rapidement si l'on rétro-propager l'erreur sur les paramètres associés aux neurones de sortie ayant le plus d'impact sur l'erreur de classification ? Nous avons tenté de vérifier cette hypothèse. Il est important de mentionner que cette technique réduit le temps de calcul de la rétro-propagation et qu'elle pourrait éliminer une contrainte sur les paramètres qui ne sont plus optimisés. Tout d'abord il faut définir notre stratégie pour sélectionner les paramètres affectant le plus l'erreur de classification. Lorsque la classification est bonne ou mauvaise, nous avons décidé de rétro-propager l'erreur sur les paramètres associés aux connections reliant les neurones cachés et les sorties ayant une valeur de sortie supérieure à une valeur θ et sur toutes les connections reliant les entrées aux neurones cachés. Dans les deux cas, cette technique permet de réduire les ambiguïtés d'une bonne classification. Bien entendu, lorsque la classification est fausse, nous rétro-propageons l'erreur aussi sur la connexion reliant la couche cachée à le neurone de sortie représentant la bonne classe. Les résultats expérimentaux de cette dernière approche ne seront pas présentés ici, car elle n'a été explorée qu'en surface.

4.4.4 Impacts sur les différents problèmes

En n'optimisant qu'une partie des paramètres à chaque itération, on réduit nécessairement *le problème de déplacement de la cible*. Cette approche n'a aucune influence sur *le problème d'atténuation et de dilution du gradient* et sur *le problème des gradients contradictoires* mais crée un *mécanisme de spécialisation*. Cette solution réduit *le problème de symétrie* car certains poids restent fixes. Cette approche ne facilite pas la *parallélisation* de l'apprentissage.

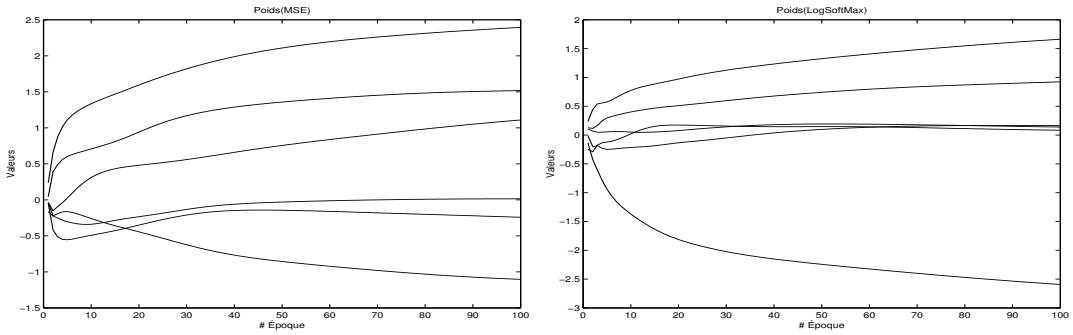

FIG. 4.10 – Exemples d'évolution de 6 poids synaptiques choisis au hasard (MSE et LogSofMax)

4.5 Prédiction des valeurs de paramètres

Lors de nos expériences, nous avons observé l'évolution des valeurs des paramètres. En général, nous constatons que ceux-ci ne varient que très peu et qu'il serait intéressant d'exploiter cette propriété en estimant les valeurs futures des paramètres. Vu la quantité de paramètres, nous avons expérimenté des techniques simples d'estimation au moyen de l'extrapolation et de la régression. L'estimation peut accélérer le processus d'apprentissage en réduisant le nombre d'époques. Les figures (4.10) et (4.11) montrent l'évolution des poids et des biais lorsque nous utilisons le MSE ou le LogSoftMax comme fonction de coût. Ces expériences ont été réalisées sur la base de données «Letters» avec une architecture totalement interconnecté et un pas optimal. Nous avons séparé les biais des poids synaptiques pour montrer que le comportement de leur évolution était semblable, ce qui à prime abord n'est pas évident. Cette observation nous permet donc de justifier l'utilisation d'une même technique d'estimation pour les biais et les poids. À partir des figures (4.10) et (4.11), nous pouvons observer que l'évolution des poids et des biais est similaire pour un pas optimal lorsqu'on utilise le MSE ou LogSoftMax comme fonction de coût. Nous constatons aussi que l'évolution devient de plus en plus linéaire au fur et à mesure de l'apprentissage. Cette dernière observation nous amemera à comparer nos techniques de prédiction à une augmentation sporadique du pas lors de l'apprentissage.
À l'ajustement du pas et du nombre de neurones cachés, cette approche introduit plusieurs nouveaux paramètres d'ajustement dont le choix de l'époques à prédire et le nombre d'époque avant lequel cette estimation peut être appliquée. Ces ajustements supplémentaires réduisent l'attrait de cette technique. Bien entendu, ceux-ci dépendent de l'ensemble d'entraînement et il devient impossible de développer d'heuristiques.

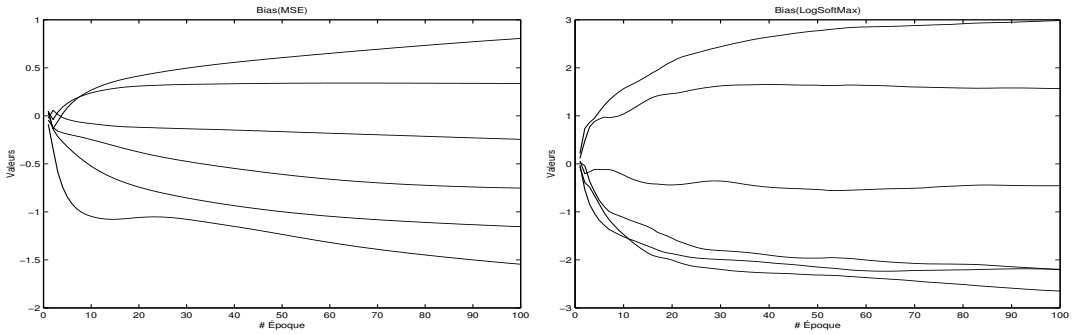

FIG. 4.11 – Exemples d'évolution de 6 biais synaptiques choisis au hasard (MSE et LogSofMax)

4.5.1 Précision de l'estimation

Une estimation imprécise peut être modélisée comme une perturbation de la valeur que l'on aurait obtenue si nous avions réellement effectué les itérations. À titre informatif, l'annexe V présente l'impact d'une perturbation lors de l'apprentissage. Sachant que la perturbation ne permet pas d'accélération de l'apprentissage et qu'elle détériore la solution, nous nous sommes limités à prédire les valeurs entre 5 et 10 époques après l'époque courante.

4.5.2 Considération d'efficacité

Cette approche consiste à estimer la valeur qu'atteindra chacun des paramètres «n» époques plus tard à partir des valeurs précédentes. Pour que cette approche soit pertinente, le temps de calcul de cette estimation plus le temps pour réaliser une époque doit être inférieur au temps requis pour effectuer les «n» époques (équation 4.5). Après l'estimation, nous ré-optimisons tous les paramètres car une estimation imprécise peut entraîner temporairement une dégradation importante de la solution (§V). En pratique, nous avons observé que l'estimation dégrade l'erreur mais qu'après avoir ré-optimisé tous les paramètres après une seule époque, l'erreur descend dramatiquement comme si il fallait réajuster les paramètres après l'estimation.

$$temps_{estimation} + temps_{optimisation}(1) < temps_{optimisation}(n) \tag{4.5}$$

Voici les points importants à considérer pour choisir une technique d'estimation qui pourrait accélérer l'apprentissage :

- La quantité de paramètres à estimer est importante
- Une faible perturbation n'a pas de répercussion significative (Annexe §IV)

Sachant que la quantité de paramètres à estimer est importante, le calcul de l'estimation doit être simple.

4.5.3 Approches pour extrapoler

Nous avons expérimenté trois méthodes pour prédire les valeurs futures des paramètres. Elles consistent à trouver les coefficients de polynômes au moyen de l'historique[7].

- Extrapolation quadratique
- Extrapolation cubique
- Extrapolation par régression linéaire

4.5.4 Extrapolation quadratique

Pour faire une extrapolation quadratique, nous utilisons les trois derniers points de l'historique (équation 4.6). Pour des raisons de simplicité, nous leur attribuons toujours les indices 0, 1 et 2. En trouvant les paramètres de la parabole passant par ces trois points, nous pouvons utiliser l'équation 4.7 afin d'estimer les valeurs pour $x > 2$. En posant $x_0 = 0$,$x_1 = 1$ et $x_2 = 2$ alors $y(0) = c$, $y(1) = a + b + c$ et $y(2) = 4 \cdot a + 2 \cdot b + c$.

$$historique : (x_0, y_0), (x_1, y_1), (x_2, y_2) \tag{4.6}$$

$$y(x) = a \cdot x^2 + b \cdot x + c \tag{4.7}$$

$$c = y_0 \tag{4.8}$$

$$b = (4 \cdot y_1 - y_2 - 3 \cdot y_0)/2 \tag{4.9}$$

$$a = y_1 - b - c \tag{4.10}$$

4.5.5 Extrapolation cubique

Pour faire une extrapolation cubique, nous utilisons les quatre derniers points de l'historique (4.11). En trouvant les paramètres du polynôme de degré 3 passant par ces quatre points, nous pouvons utiliser l'équation (4.12) afin d'estimer les valeurs pour $x > 3$. En posant $x_0 = 0$, $x_1 = 1$, $x_2 = 2$ et $x_3 = 3$ alors $y(0) = d$, $y(1) = a+b+c+d$, $y(2) = 8 \cdot a + 4 \cdot b + 2 \cdot c + d$ et $y(3) = 27 \cdot a + 9 \cdot a + 3 \cdot c + d$.

$$historique : (x_0, y_0), (x_1, y_1), (x_2, y_2, (x_3, y_3) \tag{4.11}$$

$$y(x) = a \cdot x^3 + b \cdot x^2 + c \cdot x + d \tag{4.12}$$

$$d = y_o \tag{4.13}$$

[7]L'historique d'un paramètre est les «n» dernières valeurs séquentielles obtenues après chaque époque

$$c = (y_3 - 4.5 \cdot y_2 + 9 \cdot y_1 - 5.5 \cdot d)/3 \tag{4.14}$$

$$b = (8 \cdot y_1 - 7 \cdot d - 6 \cdot c - y_2)/4 \tag{4.15}$$

$$a = y_1 - d - c - b \tag{4.16}$$

4.5.6 Régression linéaire

La dernière technique d'estimation expérimentée consiste à faire une régression linéaire sur les «n» derniers éléments de l'historique. La regression permet de trouver la droite qui réduit au minimum la somme de l'erreur quadratique de tous les points. En trouvant les coefficients m et b de l'équation 4.17, nous pouvons effectuer des prédictions afin d'estimer les valeurs pour $x > n$.

$$y(x) = m \cdot x + b \tag{4.17}$$

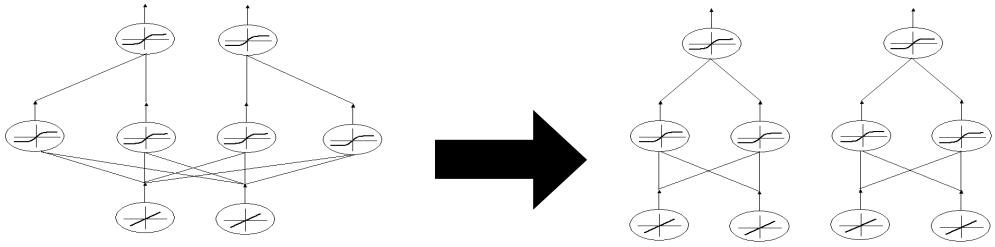

FIG. 4.12 – Architecture découplée (partiellement interconnectée)

4.6 Architecture découplée

Cette solution ne s'applique qu'aux problèmes de classification. Au lieu d'utiliser un réseau totalement interconnecté (figure 2.1), on crée un réseau distinct pour chaque classe (figure 4.12). Avec cette configuration, chaque réseau apprend à distinguer une classe des autres. Cette approche transforme le problème de classification en problèmes binaires indépendants si la fonction de coût est le MSE. Les inconvénients à architecture sont développés à la section 4.6.1. Les réseaux peuvent donc être entraînés séparément ou en parallèle. Lorsque la fonction de coût est le LogSoftMax, les réseaux ne peuvent pas être entraînés séparément mais peuvent tout de même être entraînés en parallèle. De façon générale, en utilisant un réseau par classe, on peut faire apprendre les différents réseaux en parallèle et permettre une accélération de l'apprentissage de l'ordre du nombre de classes.
Le *problème du déplacement de la cible* est réduit, car pour un même problème, on devrait avoir un nombre inférieur de paramètres pouvant influencer chaque sortie comparativement à une architecture totalement interconnectée. De la même façon, on devrait avoir un nombre inférieur de neurones cachés connectés à chacune des sorties et donc *le problème d'atténuation et dilution du gradient* devrait être réduit. Cette architecture élimine *le problème des gradients contradictoires* car les neurones cachés ne sont connectés qu'à une seule sortie. *Le problème de symétrie* n'est pas affecté et l'on crée un *mécanisme de spécialisation* car chaque réseau se spécialisera sur la réduction de l'erreur associée à une seule classe.

4.6.1 Problème relié à une architecture découplée

Certes, l'utilisation d'une architecture découplée élimine *le problème des gradients contradictoires* mais, malheureusement, il introduit un problème. Pour un nombre identique de paramètres, un réseau d'architecture découplée permet de représenter un espace de solutions inférieur à un réseau totalement interconnecté. Cela est causé

par l'inexistence du partage de l'information des neurones cachés. La complexité de la fonction pouvant être représentée par chacune des sorties est donc réduite. Donc, pour une capacité identique, un réseau d'architecture découplée doit avoir beaucoup plus de paramètres libres. Ceci implique nécessairement une augmentation du temps de calcul pour effectuer une itération.

4.7 Impacts des solutions

Chacune des solutions a une influence sur le *problème du déplacement de la cible* (§3.3.4), *le problème des gradients contradictoires* (§3.3.5), *le problème d'inexistence d'un mécanisme de spécialisation* (§3.3.6 et §3.5.1), *le problème de dilution et d'atténuation du gradient* (§3.4.1) et sur *le problème de symétrie* (§3.5.2). L'atténuation ou l'élimination d'un ou plusieurs de ces problèmes devrait permettre d'accélérer l'apprentissage.
Le tableau 4.2 présente une synthèse des impacts des différentes solutions sur ces différents problèmes. Une indication sur les possibilités de parallélisme des différentes solutions a été ajoutée. Tel que mentionné au début de ce chapitre, les impacts seront traités dans la perspective où l'on compare l'importance du problème de la solution par rapport à un réseau standard. À l'état final, les deux réseaux doivent avoir la même capacité. Sachant que nos solutions, à l'exception des techniques reliées à l'optimisation d'une partie des paramètres, modifient l'espace de solution durant l'entraînement, nous n'avons aucune certitude que l'atténuation ou l'élimination de certains problèmes puisse réellement accélérer le processus d'apprentissage ou permettre une meilleure exploitation de la capacité.
Les cases vides indiquent qu'il n'y a aucune influence, «↓» signifie une diminution du

	Dépl. de la cible	Grad. contradic.	spécialisation	Dilution du grad.	Symétrie	parall.
RIPS	↓		√	↓	↓	
RICC	↓		√	↓	↓	
Opt. parties	↓		√			
Extrapol.						
AD	↓	√	√	↓		√

TAB. 4.2 – Impacts des solutions sur les différents problèmes (voir page xxii pour la définition des acronymes)

problème et les «√» indiquent que l'approche élimine le problème. Il est intéressant de noter que la solution d'utiliser une architecture découplée est celle qui a une influence sur le plus grand nombre de problème. Par contre, pour un nombre identique de paramètres, cette architecture à une capacité inférieure (§4.6.1).

CHAPITRE 5

RÉSULTATS

Ce chapitre présentera une section de résultats pour deux configurations de réseaux incrémentaux (RIPS et RICC), pour les réseaux avec optimisation d'une partie des paramètres à la fois, sur le comportement lorsque l'on augmente la capacité d'un réseau dont l'architecture est découplée et sur l'efficacité des techniques d'extrapolation. Ce chapitre sera conclu par une synthèse des résultats expérimentaux obtenus.

5.1 Cadre expérimental

La base de données académique utilisée pour effectuer la majorité de nos expériences est «Letters». Il s'agit d'un problème de reconnaissance de caractères (classification :26 classes) qui contient 20 000 exemples. Cette base de données a les caractéristiques requises pour réaliser le type d'expériences désirées, car elle nécessite une quantité appréciable de paramètres (100 neurones cachés équivalant à 4326 paramètres pour une architecture totalement interconnectée), possède un nombre important de classes et contient un nombre important d'exemples. Les expériences ont été réalisées sur une des grappes d'ordinateurs du LISA constituée de 24 Pentium 4 de 2.4 MHz.

5.2 Réseaux incrémentaux - ajout de neurones (plans séparateurs)

Dans cette section, nous présentons les résultats expérimentaux des réseaux de neurones incrémentaux dont l'incrément est fait en ajoutant des plans séparateurs au niveau de la couche cachée (RIPS). Tout d'abord, nous présenterons des résultats décevants que nous avons obtenus sur la base de données «Letters». Dans un deuxième temps, ne pouvant faire une analyse de ces résultats, nous avons créé une base de données simple sur laquelle une analyse de l'évolution des plans séparateurs est réalisable.

5.2.1 Résultats : RIPS

Dans la série d'expériences suivante, nous comparons un réseau incrémental optimisé (§4.3.10) avec deux réseaux standards. Le réseau incrémental a initialement 50 neurones, et 50 nouveaux neurones y sont ajoutés durant l'entraînement. Lors de l'ajout de nouvelle capacité, la rétro-propagation est désactivée sur les 50 premiers neurones. Tel que suggéré dans la section 4.3.10, nous avons implanté la version optimisée pour tirer avantage des poids qui restent fixes. Ces résultats montrent que

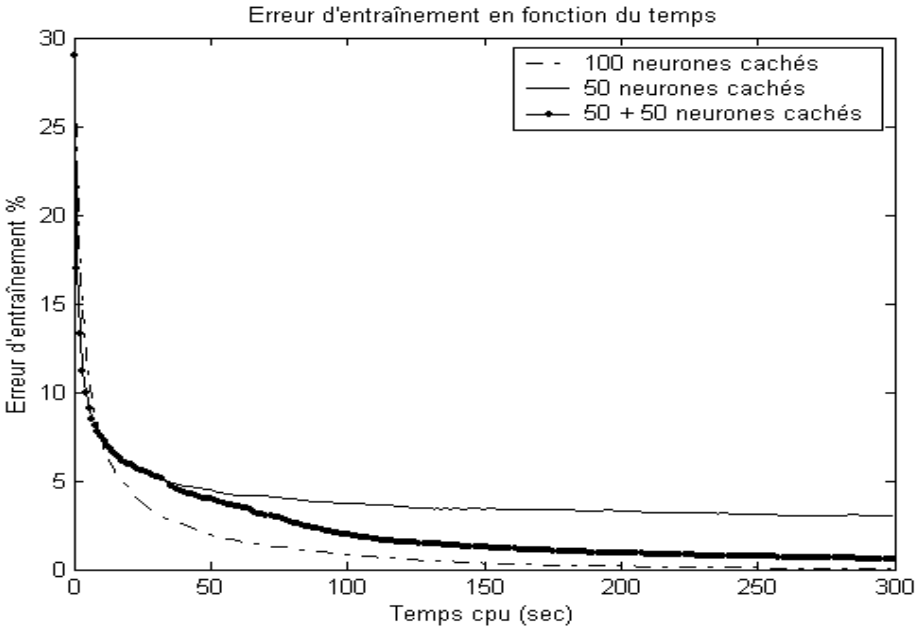

FIG. 5.1 – Résultat Réseau incrémental (erreur)

pour ce problème, il est préférable d'utiliser un plus grand nombre de neurones plutôt que d'utiliser un réseau incrémental optimisé. Il est intéressant de noter que le coût diminue beaucoup moins rapidement après l'ajout des nouveaux neurones.

5.2.2 Hypothèse basée sur les points critiques

Les résultats décevants des RIPS nous ont amené à poser l'hypothèse que l'apprentissage d'un réseau incrémental peut être plus long ou totalement bloqué parce que l'on passe nécessairement par des points critiques de la fonction de coût. Lorsque le point critique est un minimum local, l'apprentissage n'est plus possible. Dans le cas ou le point critique serait un point d'inflexion, la poursuite du processus d'apprentissage ne peut qu'être que très lent. L'utilisation d'un réseau initial plus petit entraîne la solution à converger vers des points critiques par lesquels un réseau de plus grande capacité ne serait sûrement pas passé. Cette hypothèse est inspirée, entre autre, d'un article traitant des minimums locaux et des plateaux [11]. Dans cet article, on prouve qu'un point critique correspondant à un minumum global d'un plus petit modèle peut être un minimum local ou un point critique d'un modèle plus gros.
Pour valider cette hypothèse, nous avons construit un exemple simple sur lequel nous pouvons observer le comportement et analyser les résultats en observant le compor-

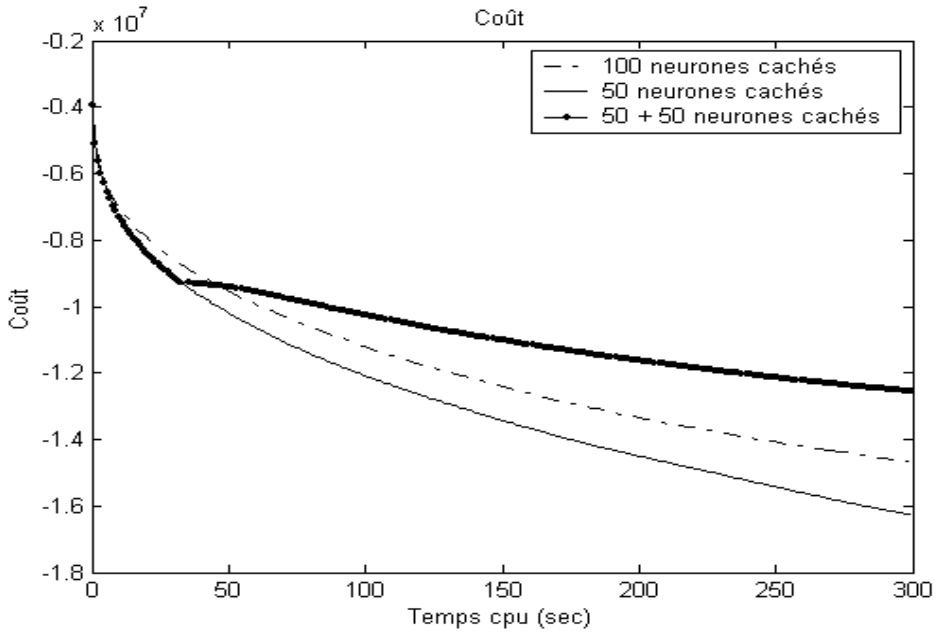

FIG. 5.2 – Résultat Réseau incrémental (coût)

tement des plans séparateurs.

5.2.3 Comportement sur un exemple simple

L'inefficacité des réseaux incrémentaux (RIPS) expérimentés sur »Letters» et l'incapacité d'en faire une analyse nous a conduit à la création d'une base de données permettant de faire une analyse sur l'évolution des plans séparateurs. Nous avons construit une base de données en deux dimensions et un mécanisme permettant de regarder le déplacement des surfaces de décision des neurones cachés. L'hypothèse que nous tentons de valider est qu'un réseau incrémental entraîne la solution à passer nécessairement par des points critiques de la fonction de coût. À partir des prochains résultats, nous pouvons analyser l'évolution des surfaces de décision pour avoir un indice sur la validité de l'hypothèse émise. Nous avons créé de toutes pièces une base de données de 2 classes (X et O) contenant 26 exemples. Nous l'avons nommée «MLDB» pour «Minimum Local Base de Données». Elle a la propriété de permettre une séparation des classes à partir d'une configuration simple de 3 plans séparateurs.
Dans le contexte où nous n'avons que 2 plans séparateurs, il est impossible de faire une séparation parfaite. Donc, un réseau de deux neurones cachés ne devrait pas pouvoir classer parfaitement l'ensemble d'entraînement. Si nous avons la possibilité

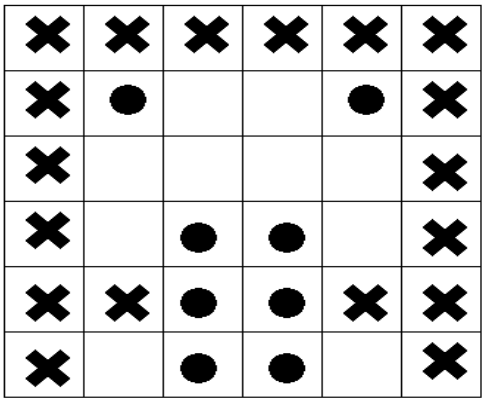

FIG. 5.3 – MLDB

d'utiliser 3 plans séparateurs, une solution simple existe. Un réseau de 3 neurones cachés devrait pouvoir faire un classement sans erreur. Par contre, dans le cas d'un réseau incrémental où certains paramètres ne sont pas fixés, si nous ajoutons à ces 2 plans la possibilité d'utiliser un troisième plan, la position des deux premiers plans séparateurs devront être modifiés pour obtenir la solution avec 3 plans séparateurs. Dans la prochaine section, nous avons observé le déplacement de ces 2 surfaces de décision d'un réseau incrémental de 2 neurones cachés où nous ajoutons un neurone. Dans le contexte où nous avons deux neurones cachés, chaque neurone se spécialisera sur l'apprentissage d'une frontière de décision. Après quelques époques, les frontières ressembleront aux droites telles que présentées dans la figure 5.5. Avec ces deux frontières, le réseau classera correctement tous les points, sauf les deux cercles gris. Si nous avons 3 neurones cachés, les frontières ressembleront aux droites telles que présentées à la figure 5.6. Avec ces trois frontières, le réseau classera correctement tous les points.

5.2.4 Résultats

La figure 5.7 présentent l'évolution des surfaces de décision pour chacun des neurones d'un réseau de deux neurones cachés, de 3 neurones (figure 5.8) et 5 neurones

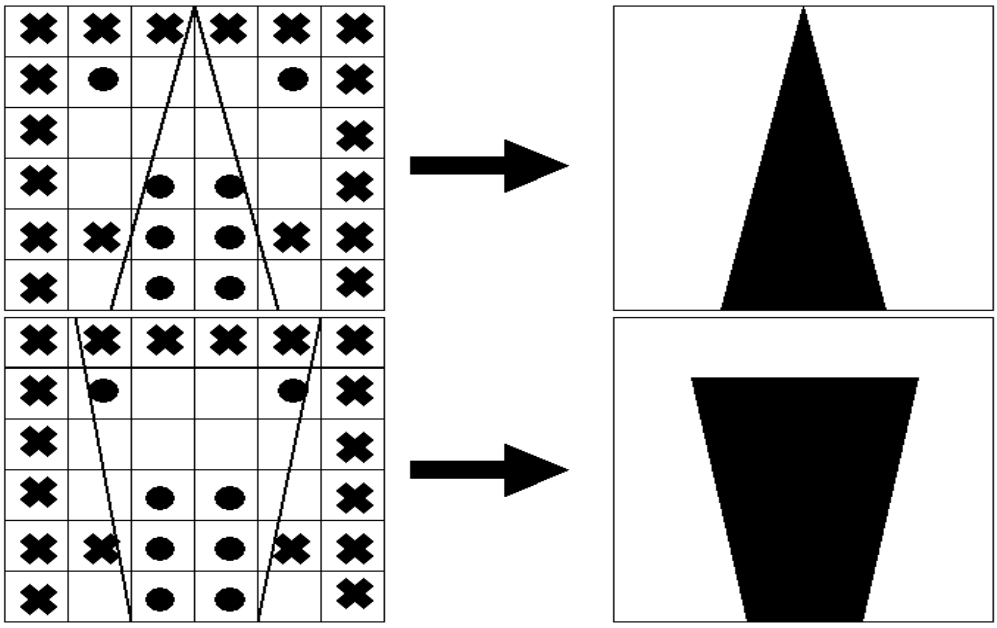

FIG. 5.4 – Surfaces de décision et région de classification

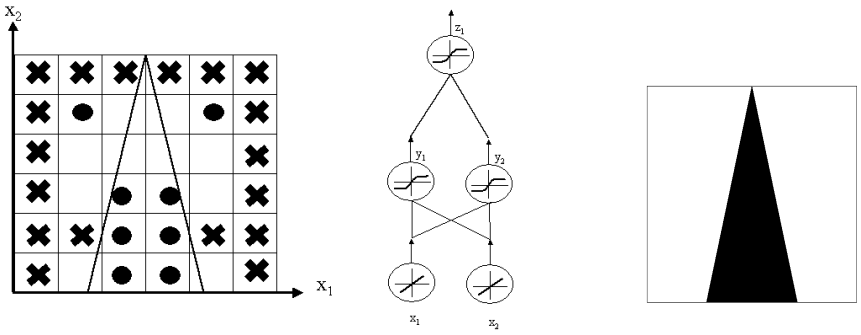

FIG. 5.5 – 2 plans séparateurs - Surfaces de décision et régions de classification

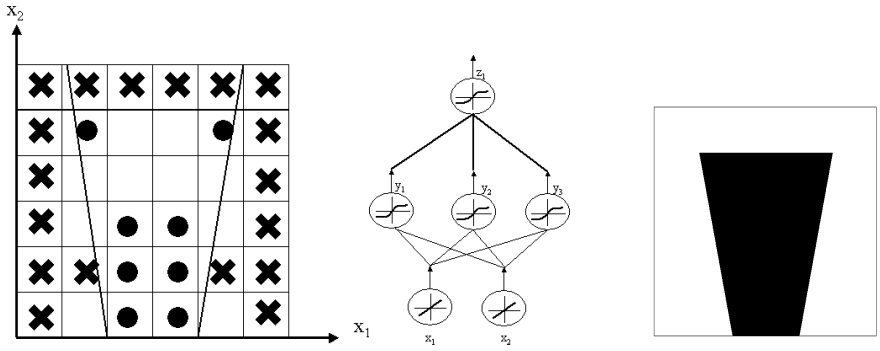

FIG. 5.6 – 3 plans séparateurs - Surfaces de décision

(2 configurations : figure 5.11 et 5.10). Dans l'évolution des 2 configurations de 5 neurones (pas=0.2 vs pas=0.5), il est intéressant de constater que certains neurones sont désactivés automatiquement en n'offrant aucune surface de décision.

5.2.5 Optimisation de tous les paramètres

Dans l'expérience suivante, nous avons entraîné un réseau de 2 neurones cachés jusqu'à ce qu'il obtienne les deux surfaces de décision telles que déjà obtenu à la figure (5.7). Par la suite, nous avons ajouté un neurone caché. En ajoutant un neurone, l'apprendre a été extrêmement long pour atteindre une classification sans erreur de l'ensemble d'entraînement. Sachant qu'un point critique d'un réseau de N-1 neurones est un point critique d'un réseau de N neurones (§5.2.2), nous avons ajouté le nouveau neurone lorsque la fonction de coût est à ce point critique. Il est à noter que durant l'entraînement du réseau ayant 3 neurones cachés initialement, le réseau n'est jamais passé par ce point critique. Cela s'explique par la présence d'une surface de décision supplémentaire qui est présente dès le début de l'apprentissage. Donc, en utilisant un mécanisme incrémental avec optimisation de tous les paramètres, la solution passe nécessairement par des solutions différentes au cours du processus itératif. La solution donnée à partir du réseau incrémental est beaucoup plus complexe (solution hautement non-linéaire). Dans cet exemple, la perturbation de la fonction de coût par l'initialisation des nouveaux poids de sortie à des valeurs différentes de zéro a permis une amélioration négligeable de la vitesse d'apprentissage, soit environ 100 époques de moins. Cette initialisation a engendré une perturbation de 0.5% de la fonction de coût. Nos expériences nous ont montré qu'il est préférable de mettre

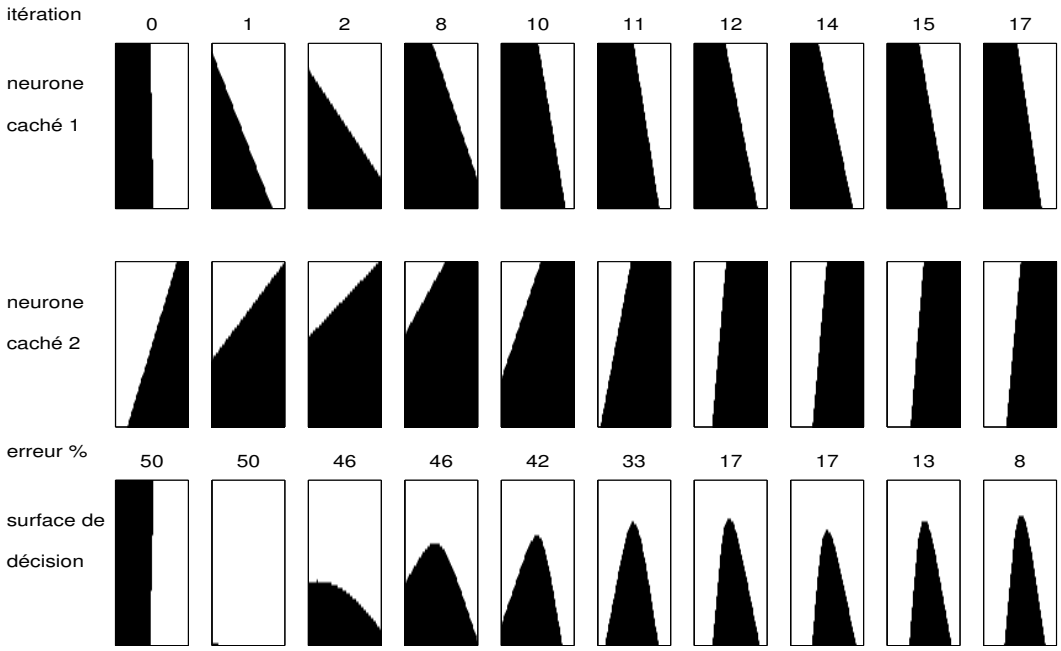

FIG. 5.7 – Résultats 2 neurones cachés

un pas supérieur pour les nouveaux paramètres afin d'accélérer l'apprentissage de ce réseau incrémental.

5.2.6 Optimisation uniquement des nouveaux paramètres

Lorsque nous n'optimisions que les nouveaux paramètres, les nombreuses expérimentations effectuées ne nous ont pas permis d'apprendre à classifier parfaitement l'ensemble d'entraînement. Ce résultat était prévisible (voir figure 5.5) au vu des figures et de notre analyse.

5.3 Résultats : RICC

Dans la série d'expériences présentées à la figure 5.12, nous comparons l'évolution de l'erreur d'apprentissage d'un réseau de deux couches cachées de 100 neurones avec deux réseaux d'une couche cachée auxquels nous avons ajouté une ou deux couches cachées. Pour ces réseaux incrémentaux, nous avons optimisé tous les paramètres et non seulement ceux des nouvelles couches. Nous avons tout d'abord trouvé la meilleure configuration de pas d'apprentissage par couche cachée, soit un pas de 0.2 pour la couche de sortie et de 0.1 pour le reste. Dans la première expérience d'ajout d'une

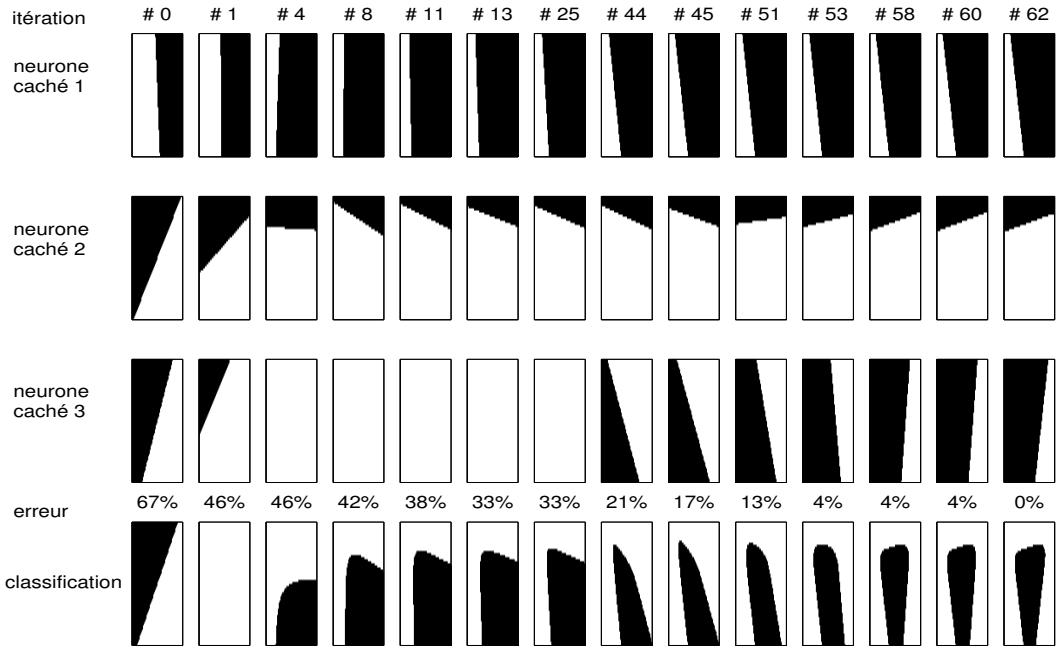

FIG. 5.8 – Résultats 3 neurones cachés

couche cachée, nous en avons ajouté une à l'époque 50 ou approximativement 90 secondes. Pour la seconde expérience, nous en avons ajouté une à l'époque 35 et 70 (approximativement 60 et 25 secondes).

Ces expériences montrent qu'il peut être avantageux d'ajouter des couches cachées durant l'entraînement. Par contre, cette constatation se limite à l'ajout d'une seule couche cachée. Lorsque nous avons ajouté une deuxième couche cachée, nous n'avons observé aucune amélioration significative. L'amélioration obtenue au moyen de l'ajout d'une couche cachée peut s'expliquer par l'élimination temporaire du problème de la courbure de la fonction de coût selon la position des paramètres (§4.3.4). Il est intéressant de constater que cette technique a permis une meilleure exploitation de la capacité du réseau, car l'erreur asymptotique est nettement inférieure.

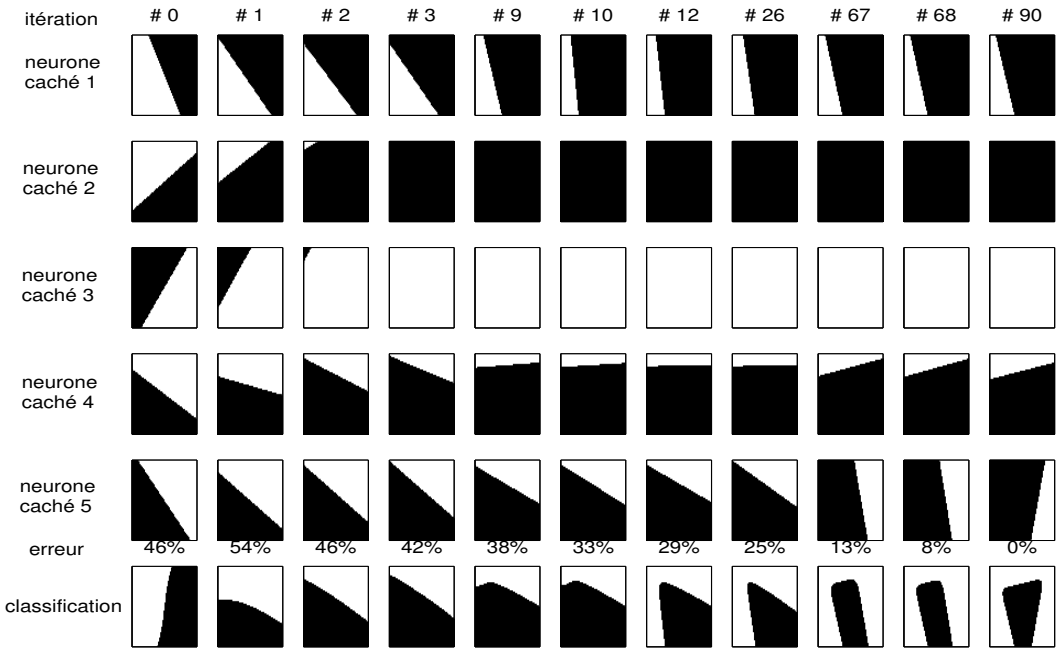

FIG. 5.9 – Résultats 5 neurones cachés ; 1ère configuration

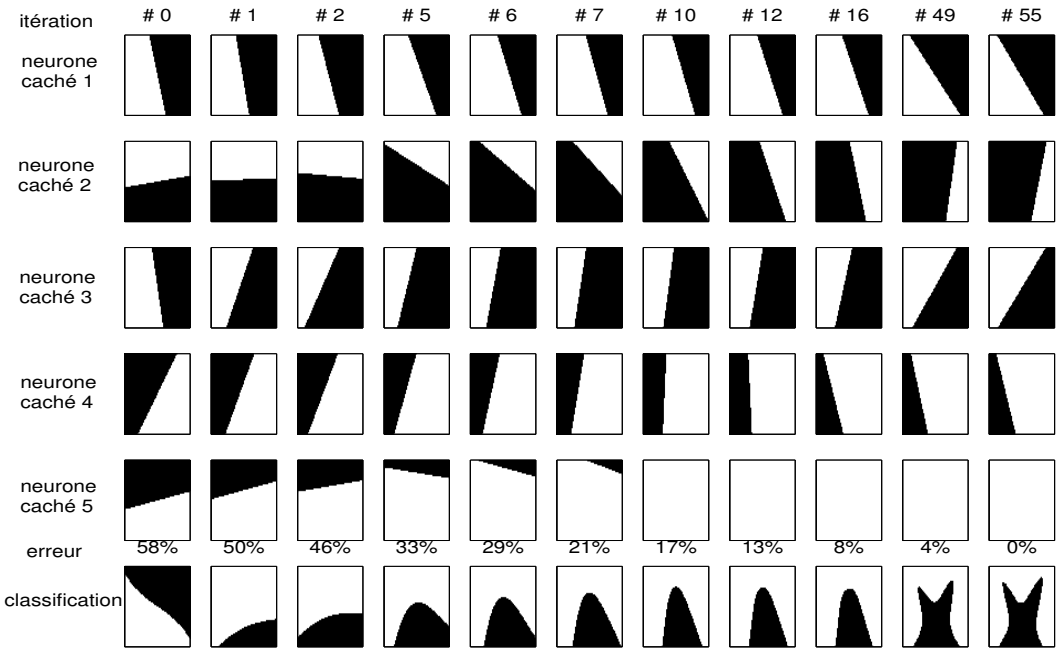

FIG. 5.10 – Résultats 5 neurones cachés ; 2ème configuration

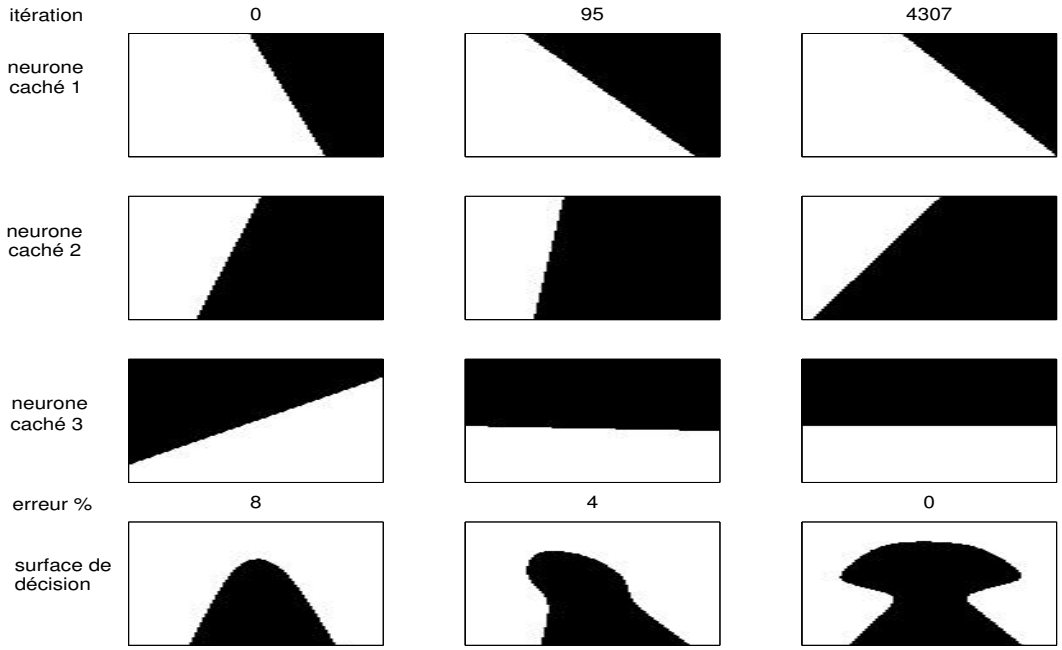

FIG. 5.11 – Réseau incrémental - Optimisation de tous les paramètres

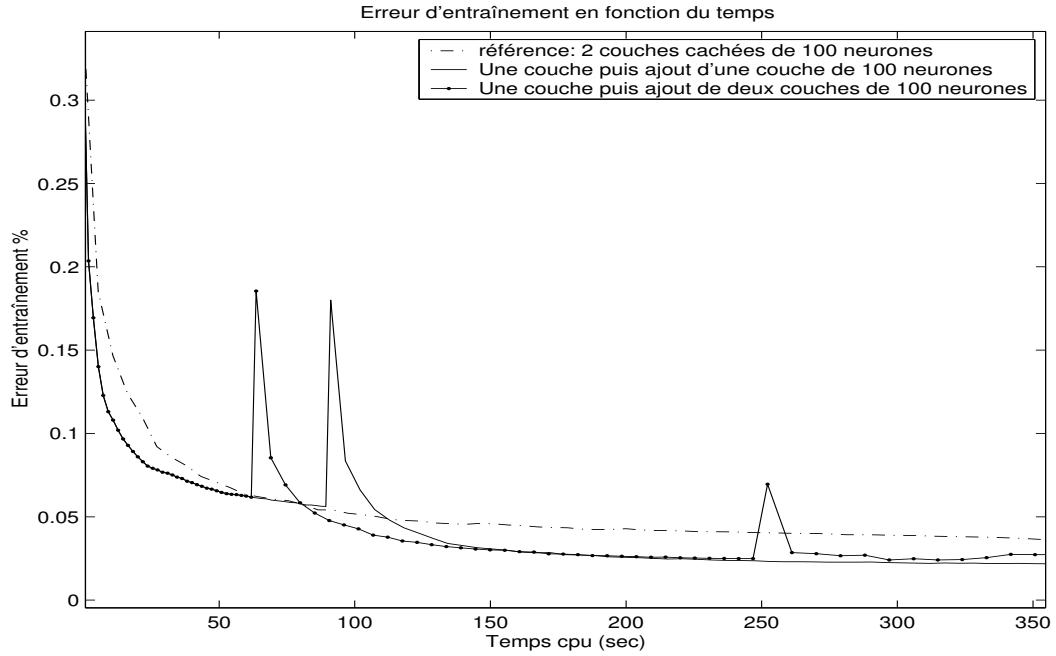

FIG. 5.12 – Ajout de couches cachées

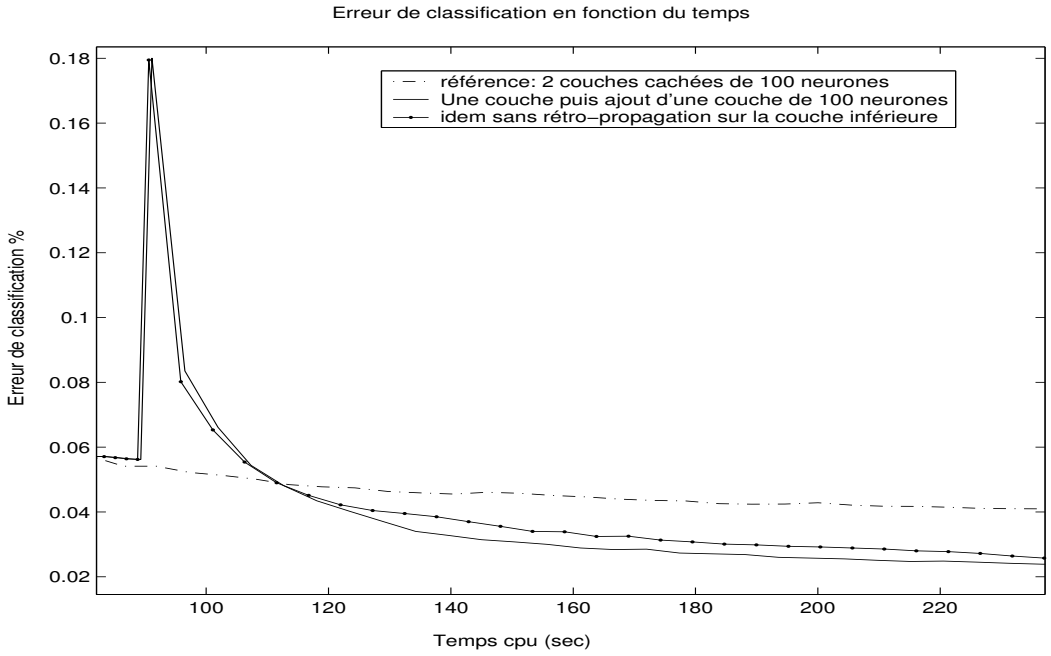

FIG. 5.13 – Comparaison avec et sans l'optimisation de la couche inférieure

5.3.1 Est-ce efficace de continuer à rétro-propager dans les couches inférieures ?

L'expérience présentée à la figure 5.13 a pour but de voir s'il est efficace de rétro-propager dans la couche inférieure après l'ajout d'une couche cachée. Cette expérience indique qu'il est préférable de faire la rétro-propagation sur tous les paramètres.

5.4 Résultats : Optimisation d'une partie des paramètres

Nous n'avons pas expérimenté cette approche sur la base de données «letters» par ce que nous n'avons aucune heuristique pour savoir quelle proportion des paramètres devraient être optimisée à chaque itération et que notre implantation est inefficace. Cette contrainte d'efficacité nous a amené à faire des comparaisons en termes d'époques au lieu du temps de calcul. Sachant que nous utilisons, en général, cent neurones cachés pour cette base de données, nos techniques nous donnent un critère pour sélectionner un nombre de paramètres équivalents à ceux associés à une seule neurone caché, soit 1% des paramètres du réseau.
Pour voir si cette idée était prometteuse, nous avons décidé d'expérimenter cette approche sur la base de données (MLDB[1]).

5.4.1 Résultats : ROC

À la figure 5.14, nous avons fait la descente de gradient sur les paramètres associés à un seul neurone à la fois (§4.4). Ce mécanisme a été activé après avoir optimisé tous les paramètres durant cinq époques. À chaque époque, nous choisissions le neurone suivant. La sélection s'est faite de façon circulaire. Cette expérience montre que cette technique converge vers une solution équivalente à celle que l'on obtient si l'on optimise tous les paramètres (figure 5.8). Il est à noter que l'apprentissage se fait en moins d'époques que lorsque l'on optimise tous les paramètres à la fois et cela avec un pas inférieur (0.2 versus 0.3). Cette expérience nous suggère, mais reste à vérifier, qu'il pourrait être avantageux de n'optimiser que certains paramètres à chaque itération. De plus, cela peut réduire le temps d'apprentissage étant donné que la descente de gradient est moins coûteuse à chaque itération.

[1] Voir figure 5.3 à la page 51

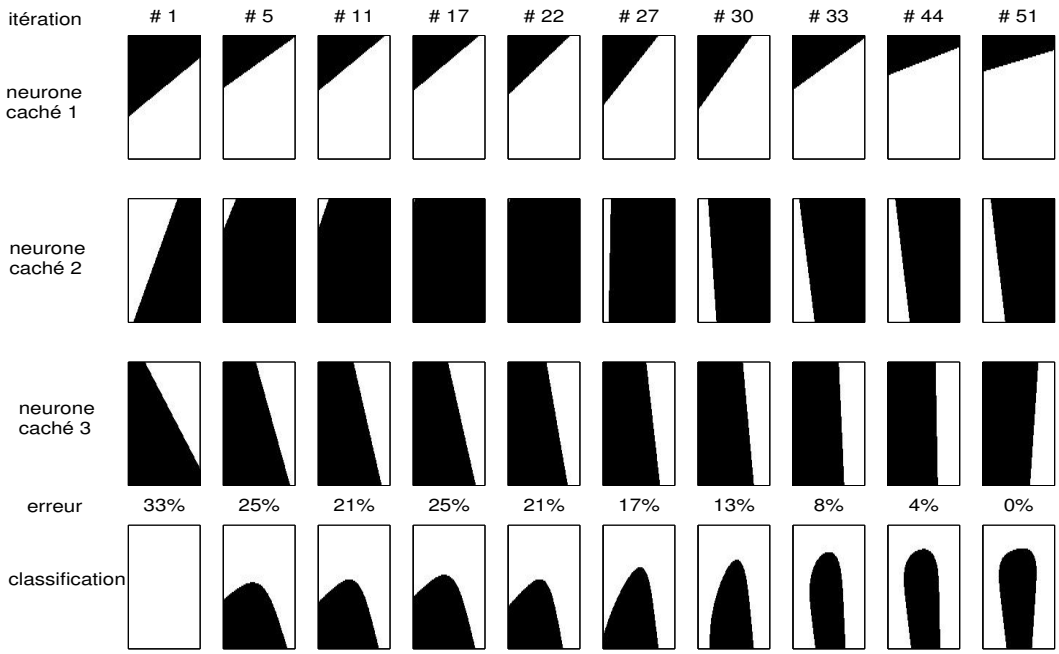

FIG. 5.14 – Optimisation circulaire

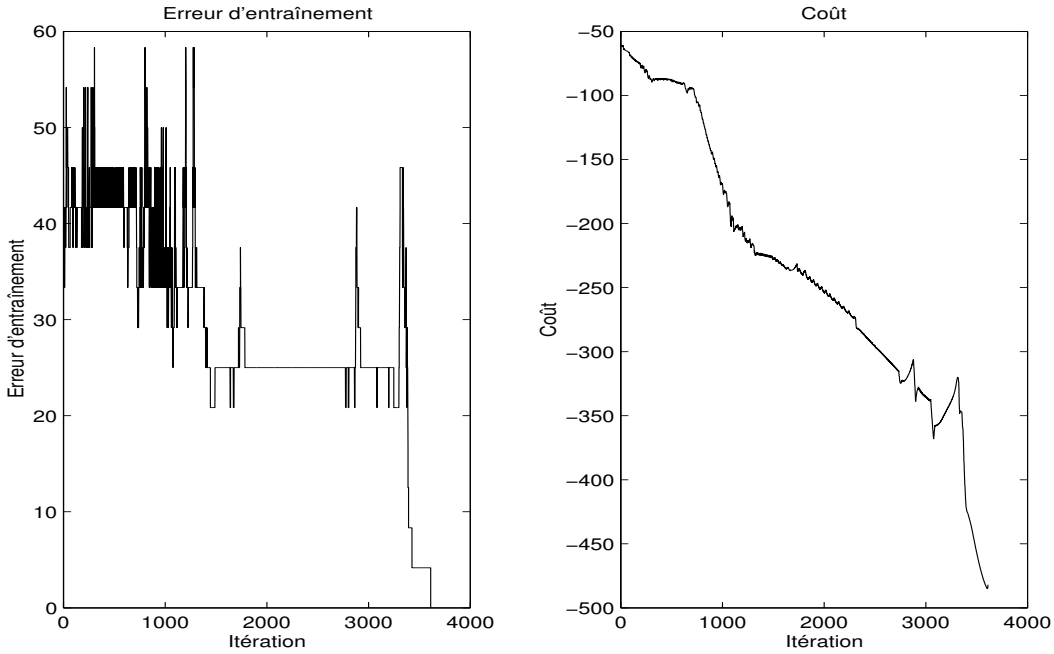

FIG. 5.15 – 3 neurones cachés : Optimisation des paramètres ayant les gradients les plus élevés

5.4.2 Résultats : ROGE

La prochaine figure (5.15) présente les résultats obtenus sur MLDB lorsque nous n'optimisons que les paramètres associés aux groupes de connections pour lesquels les gradients sont les plus élevés (§4.4.1). Il est intéressant de noter que le coût descend beaucoup plus que lorsque l'on optimise tous les paramètres (voir figure 5.16). Par contre, l'erreur de classification a un comportement étrange. Cette dernière constatation nous amène à douter de notre implantation de cette technique.

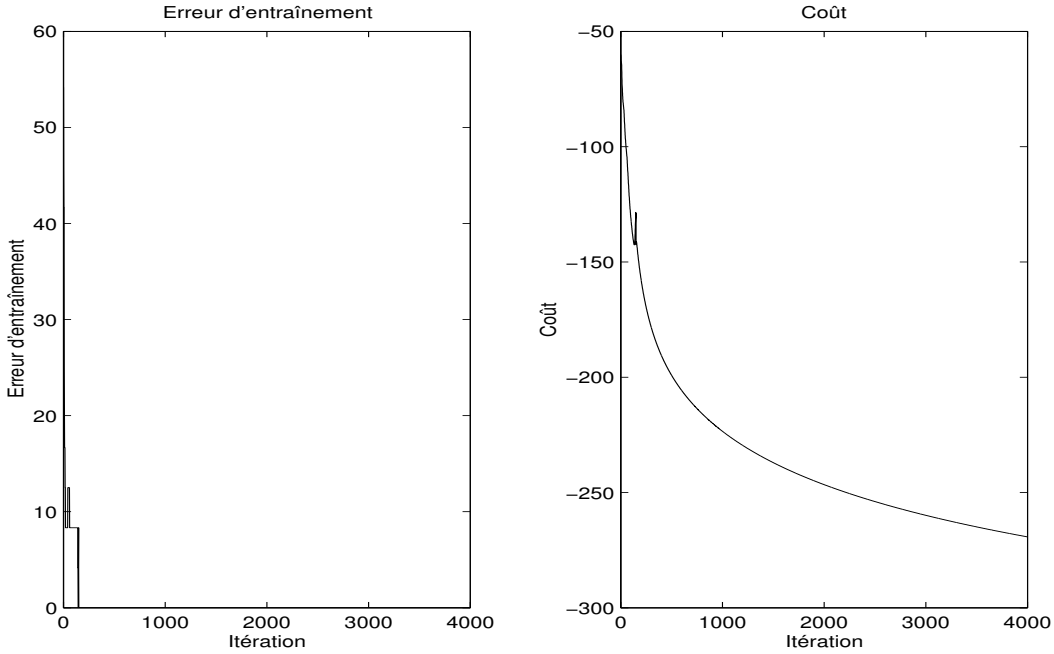

FIG. 5.16 – 3 neurones cachés : Optimisation standard

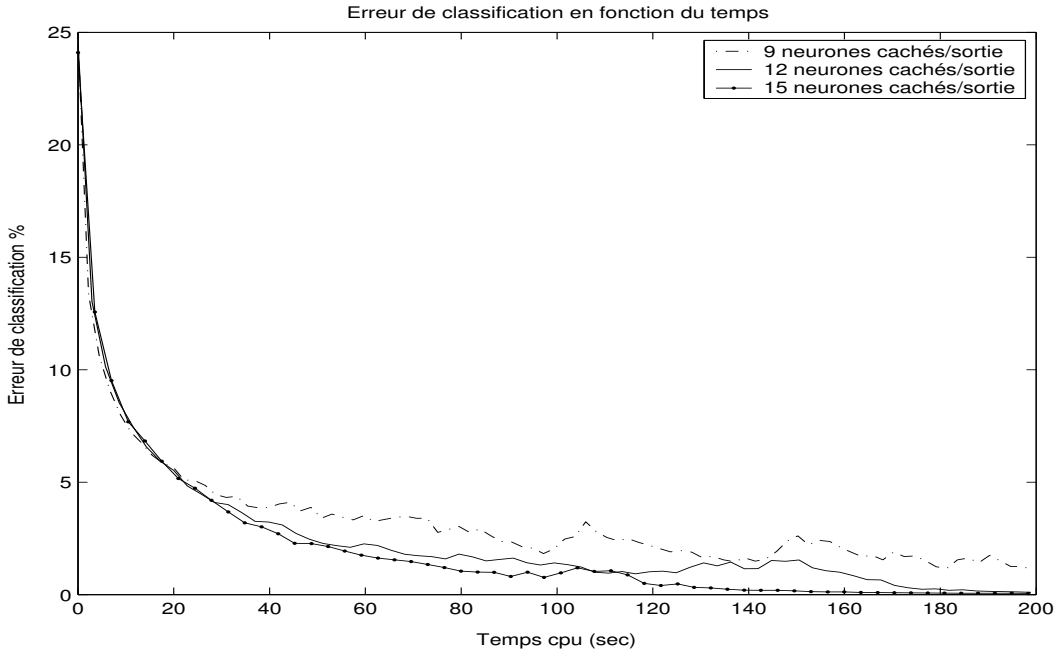

FIG. 5.17 – Architecture découplée : accéleration de l'apprentissage en fonction de l'augmentation de la capacité

5.5 Résultats : Comportement lorsqu'on accroît la capacité pour un réseau découplé

Nous avons observé qu'avec une architecture totalement interconnecté, il y a une diminution de l'efficacité d'apprentissage lorsque la capacité du réseau est augmentée(§3.1). Dans la série d'expériences suivante, nous voulions voir si ce comportement était identique lorsque nous utilisions une architecture découplée. Pour augmenter la capacité, nous connectons de trois à 5 neurones cachés supplémentaires aux sorties. La figure (5.17) présente les résultats que nous avons obtenu. Contrairement à une architecture standard, nous avons pu observer une amélioration de l'efficacité d'apprentissage. Par contre, cette amélioration ne croit plus après un certain nombre de neurones cachés par sortie (voir figure 5.18). La figure 5.19 présente la comparaison des performances de l'architecture découplée pour laquelle nous avons obtenu les meilleures performances et les meilleurs résultats obtenus pour architecture standard. Il est à noter que le réseau d'architecture découplée à environ 1.7 fois le nombre de paramètres du réseau d'architecture standard.

Contrairement à une architecture standard, nous observons une amélioration de l'efficacité d'apprentissage. Nous attribuons cette amélioration aux propriétés sui-

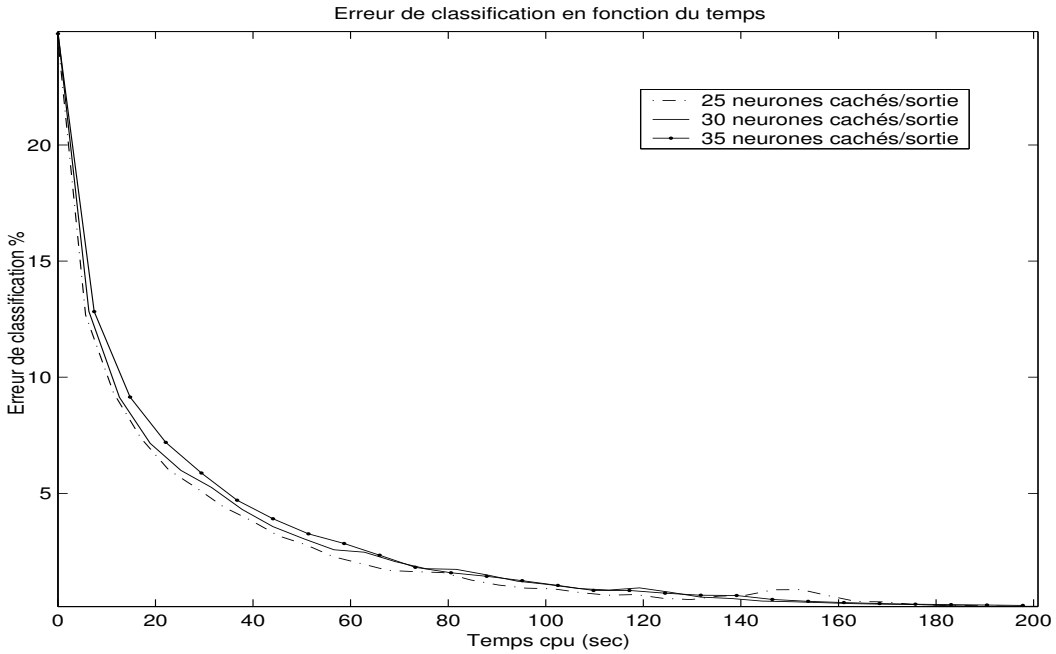

FIG. 5.18 – Architecture découplée : comportement de l'apprentissage en fonction de l'augmentation de la capacité

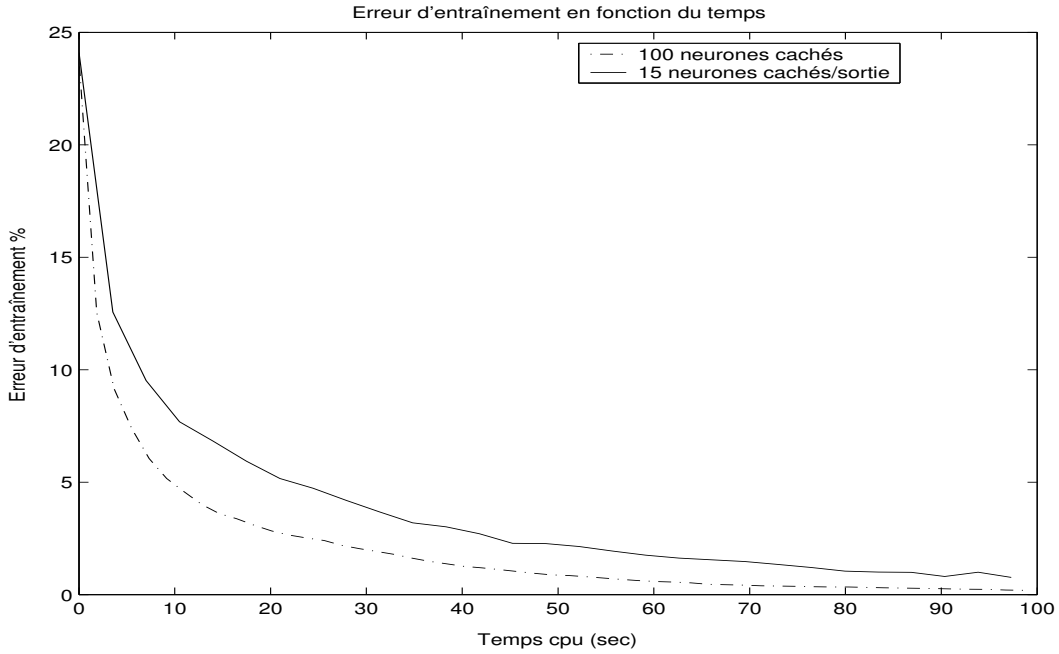

FIG. 5.19 – Architecture découplée vs Architecture standard

vantes de cette architecture :

- Élimination du *problème des gradients contradictoires*
- Diminution du *problème de déplacement de la cible*
- Création d'un mécanisme de spécialisation
- Diminution du *problème d'atténuation et dilution du gradient*

Contrairement à une architecture standard, les neurones cachés ne sont connectés qu'à une seule sortie. Considérant que la complexité de la fonction permettant de distinguer une classe des autres est différente pour chaque classe, une meilleure distribution du nombre de neurones aurait pu être faite. Pour des fins de simplicité, nous avons utilisé un nombre de neurones cachés identique pour chaque classe.

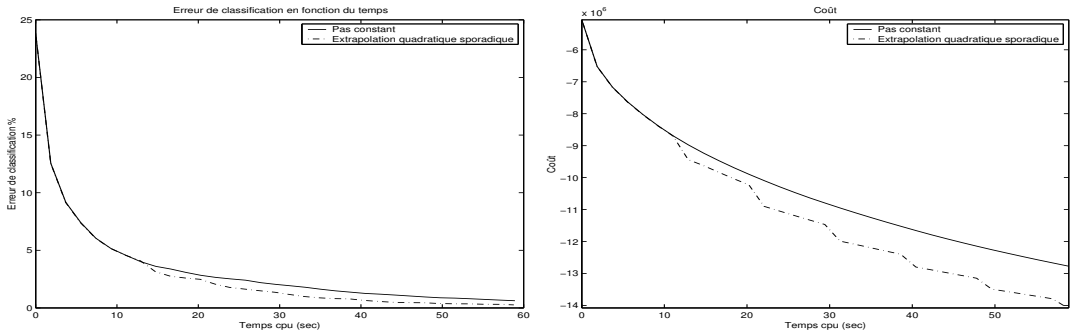

FIG. 5.20 – Extrapolation quadratique sporadique

5.6 Résultats : Extrapolation versus une augmentation du pas

Seul l'estimation EQ[2] sporadique des valeurs futures des paramètres a permis une accélération minime de l'apprentissage (figure 5.20). De plus, nous avons constaté qu'une simple augmentation sporadique du pas donne des résultats similaires (figure 5.21). L'estimation EC[3] sporadique n'a pas permis d'accélérer l'apprentissage mais à plutôt détérioré l'apprentissage et l'estimation ERL[4] sporadique n'a pas vraiment accéléré l'apprentissage.
Toutes ces techniques détérioraient l'apprentissage si elles étaient appliquées trop tôt durant l'apprentissage. Pour l'estimation EQ, nous avons effectué 10 époques avant de débuter les estimations sporadiques qui par la suite étaient au 5 époques. De plus, les estimations permettaient une accélération de l'apprentissage lorsqu'on estimait la valeur de chaque paramètre jusqu'à 7 époques plus tard.

5.7 Synthèse des résultats expérimentaux

Voici un tableau qui fait une synthèse des résultats obtenus en ce qui a trait à l'accélération du processus d'apprentissage et à l'atteinte d'un meilleur taux de classification de chaque solution explorée. N'ayant pas implanté une version efficace des techniques d'optimisation d'une partie des paramètres à la fois, nous n'avons mis aucune conclusion sur leur impact réel.

[2]Extrapolation quadratique
[3]Extrapolation cubique
[4]Extrapolation Régression Linéaire

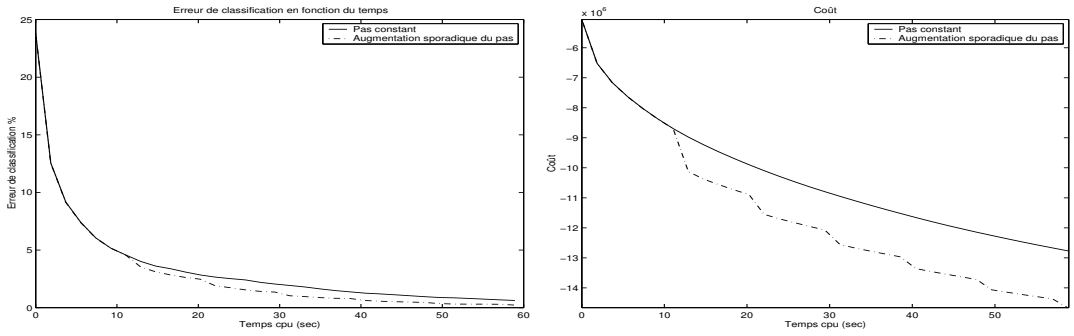

FIG. 5.21 – Augmentation sporadique du pas

Solution	**Accéleration de l'apprentissage**	**Meilleure classification**
RIPS	non	non
RICC	oui	oui
Opt. parties	-	-
Extrapol.	non	non
AD	non	non

TAB. 5.1 – Impacts réels des solutions (voir page xxii pour la définition des acronymes)

CHAPITRE 6

CONCLUSION

Les objectifs de nos travaux de recherche étaient d'étudier le comportement des réseaux de grande capacité, de présenter les problèmes d'optimisation pouvant expliquer leur inefficacité et d'explorer certaines solutions.
Pour réaliser une telle étude expérimentale et de telles comparaisons, nous avons développé une librairie de réseaux de neurones en C++ du nom de «fLayers» qui nous a permis d'avoir un contrôle total sur le mécanisme d'apprentissage d'un réseau de neurones.
Plusieurs travaux[3,4,8] mentionnent l'inefficacité de ce type de réseaux sans toutefois la documenter. Si notre réseau est totalement interconnecté, notre étude expérimentale confirme l'existence d'un problème d'optimisation lorsqu'on utilise des réseaux de neurones ayant un grand nombre de paramètres (grande capacité). Cette étude présente des résultats expérimentaux montrant que l'apprentissage devient moins efficace plus la capacité du réseau de neurones est importante et que cette capacité supplémentaire semble être difficilement exploitable.
Nous avons présenté certains des problèmes qui ralentissent ou limitent l'optimisation. Ces problèmes ont été regroupés sous le terme des *problèmes de la rétro-propagation*. Ces problèmes sont : *le problème de déplacement de la cible*, *le problème des gradients contradictoires*, *le problème d'atténuation et de dilution du gradient*, *le problème d'inexistence d'un mécanisme de spécialisation* et *le problème de symétrie*.
Nous croyons que les *problèmes de la rétro-propagation* prennent de l'importance plus le réseau a de capacité et qu'ils entraînent l'inefficacité du processus d'apprentissage ainsi qu'une difficulté d'exploitation d'une capacité supérieure. Nous avons exploré plusieurs approches qui atténuent et ou éliminent certains des *problèmes de la rétro-propagation*.
Contrairement aux observations obtenues avec un réseau totalement interconnecté, avec un réseau découplé, nous avons réussi à accélérer ou du moins réduire la détérioration de la vitesse d'apprentissage lorsque le nombre de neurones cachés était supérieur. Cette observation accentue l'importance *du problème des gradients contradictoires* car une architecture découplée élimine ce problème.
Les solutions que nous avons explorées sont : les réseaux incrémentaux par ajout de neurones au niveau de la couche cachée, les réseaux incrémentaux par ajout de couches cachées, les réseaux avec optimisation d'une partie des paramètres à la fois, les réseaux avec prédiction sporadique des valeurs futures des paramètres par extrapolation quadratique.

6.1 Observations

Les réseaux incrémentaux par ajout de couches cachées (§5.3) nous ont permis d'accélérer le processus d'apprentissage et d'améliorer significativement le taux d'apprentissage. Par contre, les réseaux incrémentaux par ajout de neurones au niveau de la couche cachée n'ont permis aucune accélération de l'apprentissage (§5.2). Les techniques de prédictions sporadiques des valeurs futures des poids synaptiques et des biais (5.6) n'ont permis, dans le meilleur des cas, aucune amélioration significative de l'apprentissage.

6.2 Contributions expérimentales

Notre étude contribue de façon importante à une meilleure compréhension des problèmes d'optimisation des réseaux de neurones de grande capacité. Les trois principales contributions sont :

1. Présentation d'indices expérimentaux de l'accroissement des problèmes d'optimisation plus les réseaux ont de capacité
2. Présentation d'une solution qui accélère l'apprentissage et améliore l'exploitation d'une grande capacité
3. Présentation d'une solution qui élimine la détérioration de la vitesse d'apprentissage lorsque le nombre de neurones cachés est augmenté

6.3 Travaux futurs

Sachant que le comportement de l'apprentissage peut varier d'un ensemble d'entraînement à l'autre et que nous avons réalisé la majorité de nos expériences sur «Letters»[1], plusieurs de nos observations prendraient de l'importance si un comportement identique était observé sur d'autres ensembles d'entraînement.
Sachant que l'apprentissage n'est pas vraiment ralentit lorsqu'on utilise une capacité supérieure dans le contexte d'un réseau d'architecture découplée et qu'il est possible de paralléliser l'apprentissage par un facteur du nombre de classes, il pourrait être intéressant de faire une étude des performances de cette architecture pour des problèmes de «data-mining» ayant un grand nombre de classes. Dans le contexte où nous avons accès à une grappe d'ordinateurs, cette étude permettrait de tirer avantage des caractéristiques de simplicité de parallélisation d'une architecture

[1]Nom d'un ensemble d'entraînement sur lequel nous avons effectué la majorité de nos expériences

découplée.
Il serait tout à fait pertinent de poursuivre les recherches pour comprendre les raisons expliquant les résultats décevants obtenus avec les réseaux incrémentaux par ajout de neurones au niveau de la couche cachée. Malgré le fait que notre hypothèse (5.2.2) basée sur les points critiques semble expliquer le comportement sur de petits ensembles de données, nous ne savons pas si celle-ci est applicable dans le contexte où les réseaux ont un grand nombre de neurones cachés.
Sachant que nos expériences nous ont montré que l'ajout d'une seule couche cachée permet d'accélérer l'apprentissage, il serait intéressant de comprendre pourquoi ce principe ne fonctionne pas pour l'ajout de plusieurs couches successives.

Bref, ce mémoire se veut une étude préliminaire démontrant l'importance des différents problèmes d'optimisation des réseaux de neurones de grande capacité. Beaucoup de travaux de recherche restent à faire.

BIBLIOGRAPHIE

[1] C. M. Bishop. *Neural Networks for Pattern Recognition.* OXFORD UNIVERSITY PRESS, 1995.

[2] R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of svms for very large scale problems, 2002.

[3] M. W. Craven and J. W. Shavlik. Using neural networks for data mining. *Future Generation Computer Systems*, 13(2–3) :211–229, 1997.

[4] J. Deogun, V. Raghavan, A. Sarkar, and H. Sever. Data mining : Research trends, challenges, and applications, 1997.

[5] R. O. DuDa. *Pattern Classification Second Edition.* JOHN WILET & SONS, 2000.

[6] J. L. Elman. Learning and development in neural networks : The importance of starting small. *Cognition*, 48(1) :71–9, 1993.

[7] A. Engelbrecht and I. Cloete. A sensitivity analysis algorithm for pruning feedforward neural networks, 1996.

[8] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, Denver 1989, 1990. Morgan Kaufmann, San Mateo.

[9] P. Frasconi, M. Gori, and A. Tesi. Successes and failures of backpropagation : a theoretical investigation. In O. Omidvar, editor, *Progress in Neural Networks*, volume 5. Ablex Publishing, 1993.

[10] J. Fritsch and M. Finke. applying divide and conquer to large scale pattern recognition tasks. In *Neural Networks : Tricks of the Trade*, pages 315–342, 1996.

[11] K. Fukumizu. Local minima and plateaus in multilayer neural networks, 1999.

[12] T.-Y. Kwok and D.-Y. Yeung. Constructive feedforward neural networks for regression problems : A survey, 1995.

[13] T. Y. Kwok and D. Y. Yeung. Constructive algorithms for structure learning in feedforward neural networks for regression problems. *IEEE Trans. on Neural Networks*, 8(3) :630–645, 1997.

[14] S. Lawrence, C. L. Giles, and A. Tsoi. What size neural network gives optimal generalization ? Convergence properties of backpropagation. Technical Report UMIACS-TR-96-22 and CS-TR-3617, Institute for Advanced Computer Studies, University of Maryland, April 1996.

[15] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Mueller. Efficient BackProp. *Lecture Notes in Computer Science*, 1524, 1998.

[16] Y. Liu and X. Yao. A population-based learning algorithm which learns both architectures and weights of neural networks, 1996.

[17] N. J. Radcliffe and I. W. Flockhart. Scalable data mining systems.

[18] T. Ragg, S. Gutjahr, and H. Sa. Automatic determination of optimal network topologies based on information theory and evolution, 1997.

[19] M. Ring. Learning sequential tasks by incrementally adding higher orders. In *Advances in Neural Information Processing Systems 5*, pages 999–999, Denver, CO, 1993.

[20] H. Schwenk and Y. Bengio. Training methods for adaptive boosting of neural networks. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[21] F. J. Śmieja. Neural network constructive algorithms : Trading generalization for learning efficiency ? Technical Report 636, =German National Center of Computer Science, 1991.
[22] M. Wynne-Jones. Node splitting : A contructive algorithm for feed-forward neural networks. *NIPS Volume 4*, 4, 1991.
[23] X. Yao. Evolving artificial neural networks. *PIEEE : Proceedings of the IEEE*, 87, 1999.

Annexe I

Comparaison du temps CPU : N versus 2N neurones

En théorie, un réseau de 2N neurones cachés doit exécuter un peu moins de 2 fois le nombre d'opérations comparativement à un réseau de N neurones cachés pour effectuer une itération. Posons un réseau de S sorties, N neurones cachés et E neurones d'entrée. Pour des fins de simplification, nous ne considérerons pas les biais. Voici le nombre d'opérations à exécuter par itération en fonction du nombre de neurones cachés, de neurones d'entrée et de neurones de sortie :

nb de neurones cachés	nb * et + du gradient	nb appels fct d'activ. et calcul de δ
N	N*(E+S)	S+N
N/2	N/2*(E+S) ou A	S+N/2 ou B

TAB. I.1 – Nombre d'opérations à exécuter

Voici la comparaison en fonction du nombre d'opérations nécessaires pour N/2 neurones cachés :

Type d'opération	N neurones cachés	N/2 neurones cachés
nb * et + du gradient	2A	A
nb appels fct d'activ. et calcul de δ	2B-S/2	B

TAB. I.2 – Comparaison N vs 2N neurones cachés

Ce tableau nous indique qu'un réseau de N neurones cachés exécute moins de deux fois le nombre d'opérations qu'un réseau de N/2 neurones cachés pour une itération.

En pratique, nous devrions nous attendre à ce qu'un réseau de capacité doublée prenne deux fois plus de temps de calcul pour effectuer une itération. En utilisant la librairie fLayers, nous obtenons un ratio de 1.85.

Annexe II

Profiling

Voici les résultats de «profiling» exécutés sur la base de données «Letters» (26 classes 100 époques). Cette annexe a pour but de présenter le pourcentage du temps passé dans la propagation et dans la rétro-propagation dépendamment de la configuration utilisée.

II.1 Config. 1 : Architecture standard (100 neurones cachés, MSE)

La proportion du temps d'apprentissage relié à la propagation est de 66.88%. Ce pourcentage se sépare de la façon suivante :

%	Fonction
47.04	Connector : :fprop()
17.78	TanhMSE : :apply()
2.06	SigmMSE : :apply()

La proportion du temps d'apprentissage relié à la rétro-propagation est de 30.9 %. Ce pourcentage se sépare de la façon suivante :

%	Fonction
16.76	Layer : :ComputeHiddenSensitivity()
14.14	Connector : :updateOnFly()

II.2 Config. 2 : Architecture standard (100 neurones cachés, LogSoftMax)

La proportion du temps d'apprentissage relié à la propagation est de 66.88%. Ce pourcentage se sépare de la façon suivante :

%	Fonction
30.93	Connector : :fprop(void)
30.74	TanhMSE : :apply(void)
5.11	LogSoftMax : :apply(void)

La proportion du temps d'apprentissage relié à la rétro-propagation est de 31.23 %. Ce pourcentage se sépare de la façon suivante :

%	Fonction
9.32	Layer : :ComputeHiddenSensitivity(void)
3.88	LogSoftMax : :ComputeOutputSensitivity(void)
0.79	Layer : :bprop(bool)
0.56	TanhMSE : :multiplyByDerivative(void)

II.3 Config. 3 : Architecture découplée (9 neurones cachés/sortie, LogSoftMax)

La proportion du temps d'apprentissage relié à la propagation est de 48.89%. Ce pourcentage se sépare de la façon suivante :

%	Fonction
24.29	Connector : :fprop(void)
14.64	TanhMSE : :apply(void)
3.69	Topology : :fprop(bool)
2.02	Layer : :fprop(void)
1.30	LogSoftMax : :apply(void)
1.20	LogSoftMax : :ComputeOutputSensitivity(void)
1.03	Connector : :bprop(bool)
0.72	Topology : :bprop(float,bool)

La proportion du temps d'apprentissage relié à la rétro-propagation est de 49.89%. Ce pourcentage se sépare de la façon suivante :

%	Fonction
23.13	Layer : :ComputeHiddenSensitivity(void)
21.52	Connector : :updateOnFly(void)
3.08	Layer : :bprop(bool)
2.16	TanhMSE : :multiplyByDerivative(void)

Annexe III

Perturbation

La présente annexe a pour but de présenter l'effet d'une perturbation constante au cours de l'apprentissage. À chaque époque, nous avons appliqué une perturbation identique de 10% à chaque paramètre. L'erreur, après l'application de la perturbation, est représentée par des points dans le graphique ci-dessous (IV.1). Cette figure représente la détérioration de l'erreur après l'application de la perturbation. Nous constatons que l'impact de la perturbation augmente au cours de l'apprentissage.

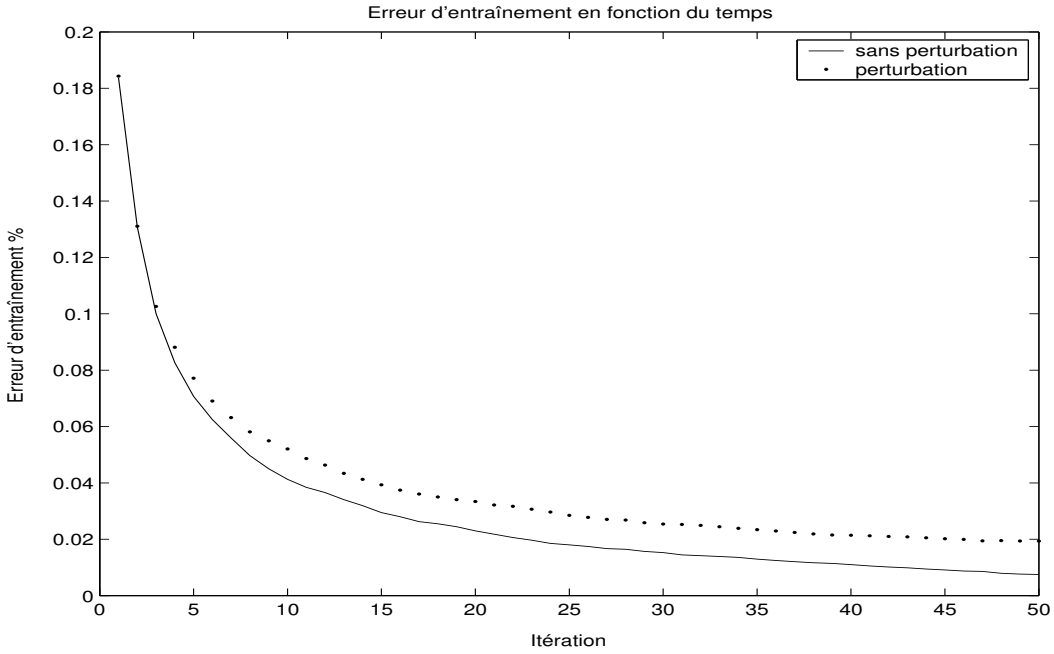

FIG. III.1 – Influence d'une perturbation constante

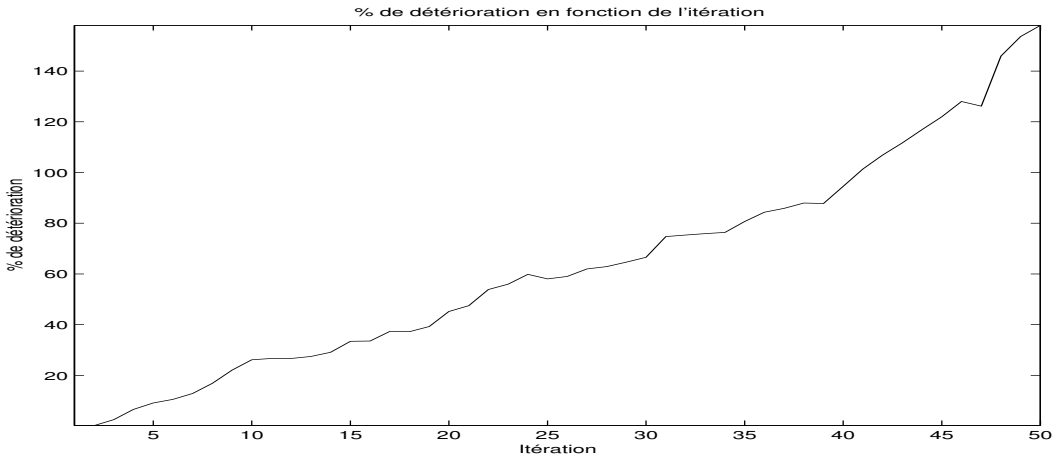

FIG. III.2 – Perturbation

Annexe IV

Perturbation sporadique

La présente annexe a pour but de présenter l'influence d'une perturbation sporadique sur la vitesse d'apprentissage. Dans le contexte où la fonction de coût est une vallée dont le gradient suit le creux, il pourrait être intéressant de perturber, à l'occasion, la solution courante. Cette perturbation permettrait de se déplacer vers une solution où le gradient serait plus élevé, entraînant ainsi une convergence plus rapide. Nous avons expérimenté cette idée avec une perturbation de 5% et voici les résultats : Comme nous pouvons le constater, cette technique ne marche pas sur la base de données «Letters». Par contre, il est intéressant de mentionner que dans cet exemple, la perturbation ne semble pas changer ou modifier la tendance de l'évolution de l'erreur.

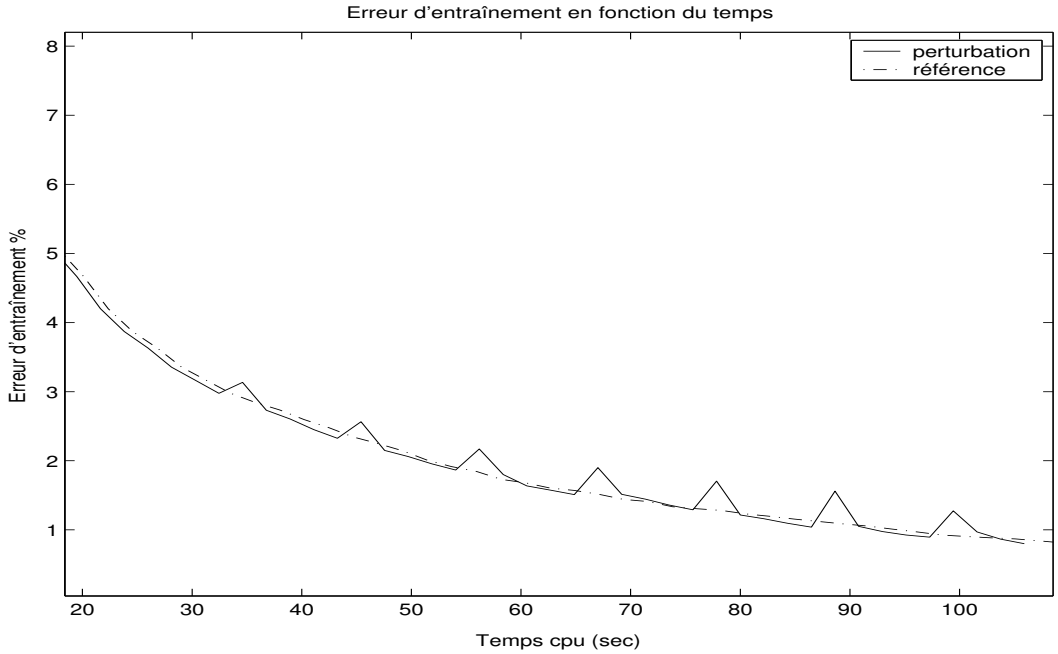

FIG. IV.1 – Influence d'une perturbation sporadique

Annexe V

Optimisation sélective

La présente annexe a pour but de présenter certain résultat intéressants concernant les divers mécanismes de sélection de paramètres à optimiser :

- Optimisation circulaire
- Optimisation des paramètres associés au neurone caché ayant la sortie la plus élevée
- Optimisation des paramètres associés au neurone caché dont la dérivée de la fonction d'activation est la plus élevée
- Optimisation des paramètres associés au neurone ayant la valeur de sortie la plus élevée
- Optimisation des paramètres associés au neurone caché ayant le facteur de *sensibilité* le plus élevé
- Optimisation des paramètres ayant les gradients les plus élevés §4.4.1

Ces expériences ont été réalisées sur la base de donnée MLDB. La configuration est un réseau de 3 neurones cachés.

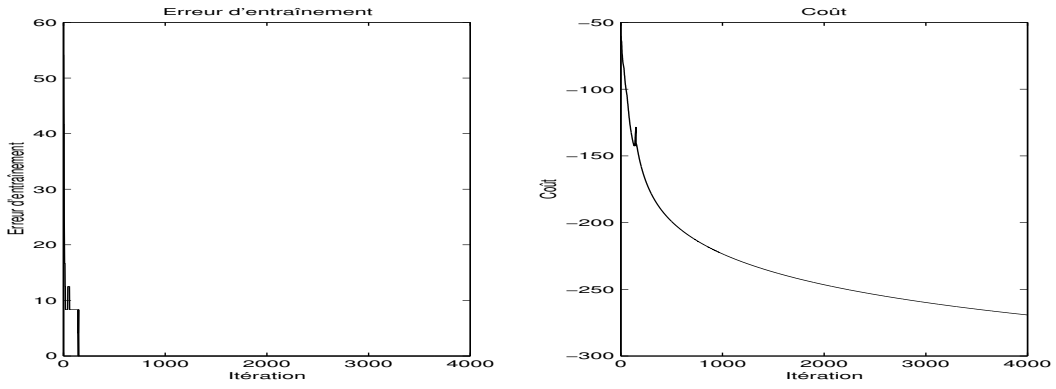

FIG. V.1 – Optimisation standard

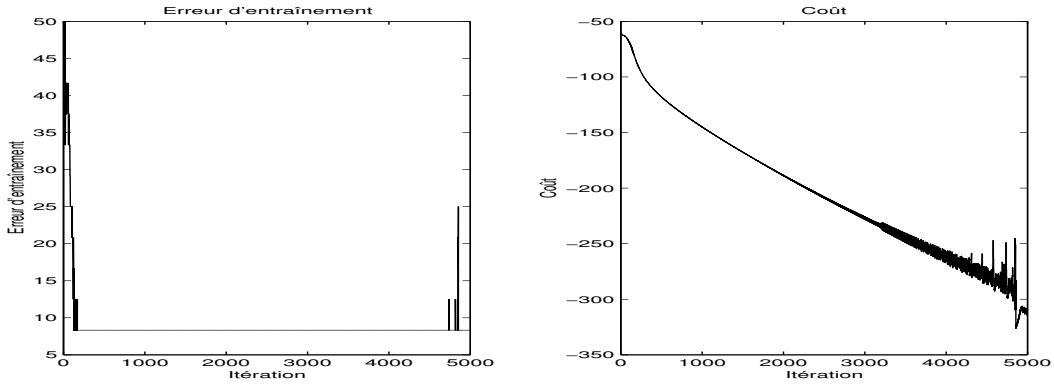

FIG. V.2 – Optimisation circulaire

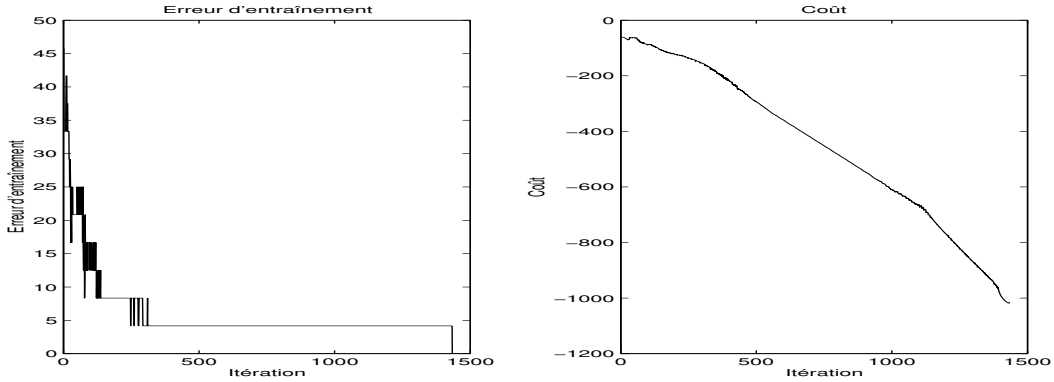

FIG. V.3 – Optimisation : Max(dérivée de la fonction d'activation)

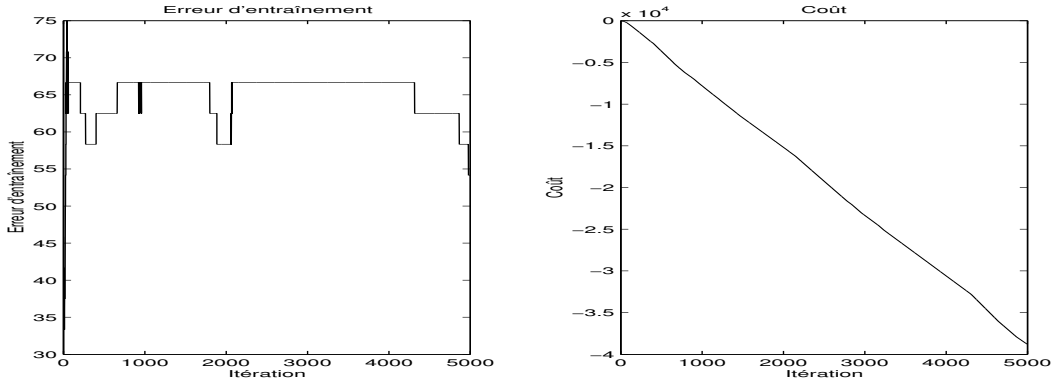

FIG. V.4 – Optimisation : Max(sortie)

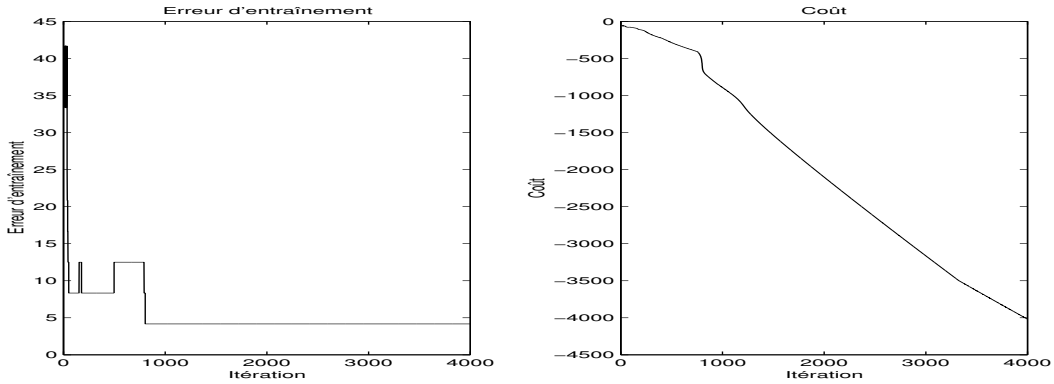

FIG. V.5 – Optimisation : Max(*sensibilité*)

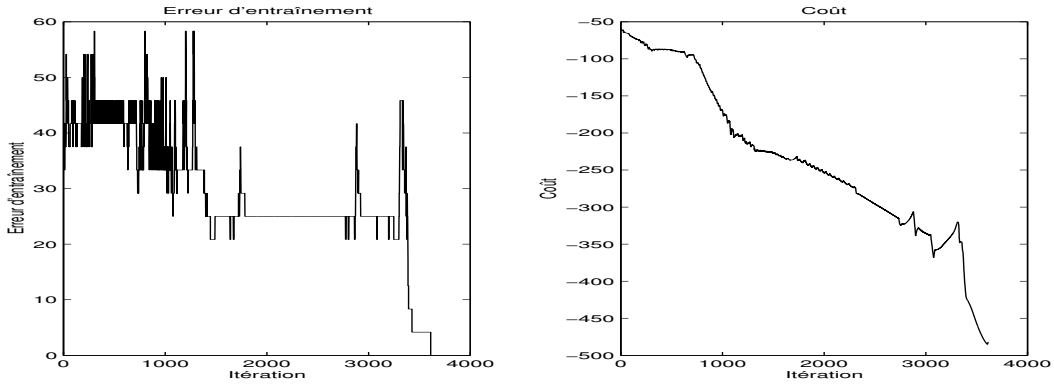

FIG. V.6 – Optimisation : Max(Gradient)