Chapitre 6 Apprentissage des réseaux de neurones et régularisation

Après une introduction rapide aux réseaux de neurones et à la problématique de la classification, l'essentiel de ce chapitre est consacré à l'apprentissage, et notamment aux problèmes liés au surapprentissage dans les problèmes de classification. Nous montrons que dans certains cas, les "méthodes actives" comme la régularisation par le *weight decay* sont indispensables pour limiter le surapprentissage. Cette technique exige néanmoins la détermination de paramètres supplémentaires, appelés hyperparamètres. L'approche bayesienne propose une solution de principe à cette détermination, que nous présentons dans ce chapitre, et dont nous décrirons l'application dans les chapitres suivants.

6.1 Problématique de la classification supervisée

6.1.1 La catégorisation de textes est un problème de classification supervisée

Le problème du filtrage de textes pour un thème donné est abordé dans ce mémoire comme un problème de classification supervisée à deux classes : la classe des textes pertinents et la classe des textes non pertinents. Pour construire un filtre relatif à un thème donné, il faut donc disposer d'exemples de chaque classe, préalablement étiquetés comme pertinents ou non pertinents. Grâce à ces deux ensembles de textes, il est possible de construire un classifieur grâce à un algorithme d'apprentissage. Si cet apprentissage est correctement réalisé, le modèle est capable d'estimer, pour chaque nouveau texte, sa probabilité de pertinence pour le thème considéré.

6.1.2 Théorème de Bayes

Le théorème de Bayes fournit un cadre théorique pour la problématique de la classification à deux classes, et il intervient également dans l'approche bayesienne exposée au paragraphe 6.6.
Si l'on considère un problème à deux classes C_1 et C_2, le théorème de Bayes permet de calculer les probabilités *a posteriori* connaissant les distributions des observations *a priori*.

$$P(C_1|x) = \frac{p(x|C_1)\,P(C_1)}{p(x)}$$

$P(C_1|x)$ est la probabilité *a posteriori* d'appartenir à la classe C_1 connaissant le vecteur des descripteurs x, $p(x|C_1)$ est la densité de probabilité du vecteur x dans la classe C_1, $P(C_1)$ est la probabilité *a priori* de la classe C_1 et $p(x)$ est la densité de probabilité non conditionnelle définie par :

$$p(x) = p(x|C_1)\,P(C_1) + p(x|C_2)\,P(C_2)$$

Dans le cas d'un problème de classification, cette formule définit une règle de décision : la probabilité de mauvaise classification est minimisée en sélectionnant la classe qui a la plus grande probabilité *a posteriori*.

Ce théorème est au cœur de la problématique de la classification : on peut distinguer (i) les méthodes de classification qui essayent de modéliser les densités de probabilités pour calculer les probabilités *a priori*, et (ii) les méthodes qui essayent de modéliser directement les probabilités *a posteriori*. Le détail de ces différentes méthodes peut être trouvé dans [Bishop, 1995] ou [Stoppiglia, 1997] ; les réseaux de neurones utilisés dans ce mémoire appartiennent à la deuxième catégorie.

6.2 Généralités sur les réseaux de neurones

Cette partie est une présentation succincte des principales propriétés des réseaux de neurones. L'accent est surtout mis sur les algorithmes utilisés et sur le problème du surajustement. Une présentation plus générale des réseaux de neurones et de leurs applications à d'autres tâches que la classification de textes peut être trouvée dans [Dreyfus *et al.*, 1999].

6.2.1 Le neurone formel

Un neurone formel est une fonction algébrique paramétrée, à valeurs bornées, de variables réelles appelées entrées.

En règle générale, le calcul de la valeur de cette fonction peut se décomposer en deux étapes :

- une combinaison linéaire des entrées :

$$v = w_0 + \sum_{i=1}^{n} w_i.x_i$$

Les w_i sont appelés poids synaptiques ou simplement **poids**, w_0 est appelé **biais**. Le biais peut être considéré comme la pondération de l'entrée 0 fixée à 1. v est appelé **potentiel** du neurone.

- La sortie du neurone est :

$$y = f(v) = f(\sum_{i=0}^{n} w_i.x_i)$$

La fonction f est la fonction **d'activation** du neurone. Dans la suite de ce mémoire, on considérera trois types de fonctions d'activation :

La fonction **identité** : $f(v) = v$.

*La fonction **sigmoïde** :* $f(v) = \tanh(v)$. C'est une fonction bornée à valeurs réelles comprises entre -1 et +1.

La fonction **logistique** : $f(v) = 1/(1 + \exp(-v))$. C'est une fonction bornée à valeurs réelles comprises entre 0 et 1.

6.2.2 Réseaux de neurones non bouclés

Un réseau de neurones non bouclé est une composition de fonctions réalisée par des neurones formels interconnectés entre eux. Certaines applications peuvent nécessiter plusieurs sorties (dans le cas d'une classification à plusieurs classes par exemple), mais dans notre cas, tous les réseaux utilisés ont une seule sortie.

Les possibilités d'arrangements entre les neurones sont multiples. La configuration la plus classique est appelée *perceptron multicouche*. Dans cette architecture, les neurones sont organisés en couches comme le montre la Figure 6.1 : une couche intermédiaire entre les entrées et les sorties appelée couche cachée et un neurone (ou une couche de neurones) de sortie. Les connexions se font d'une couche à la suivante sans qu'il y ait de connexion entre couches non adjacentes. Cette architecture est également appelée réseau à deux couches puisqu'il y a deux couches de poids ajustables : celle qui relie les entrées aux neurones cachés et celle qui relie les neurones cachés au neurone de sortie.

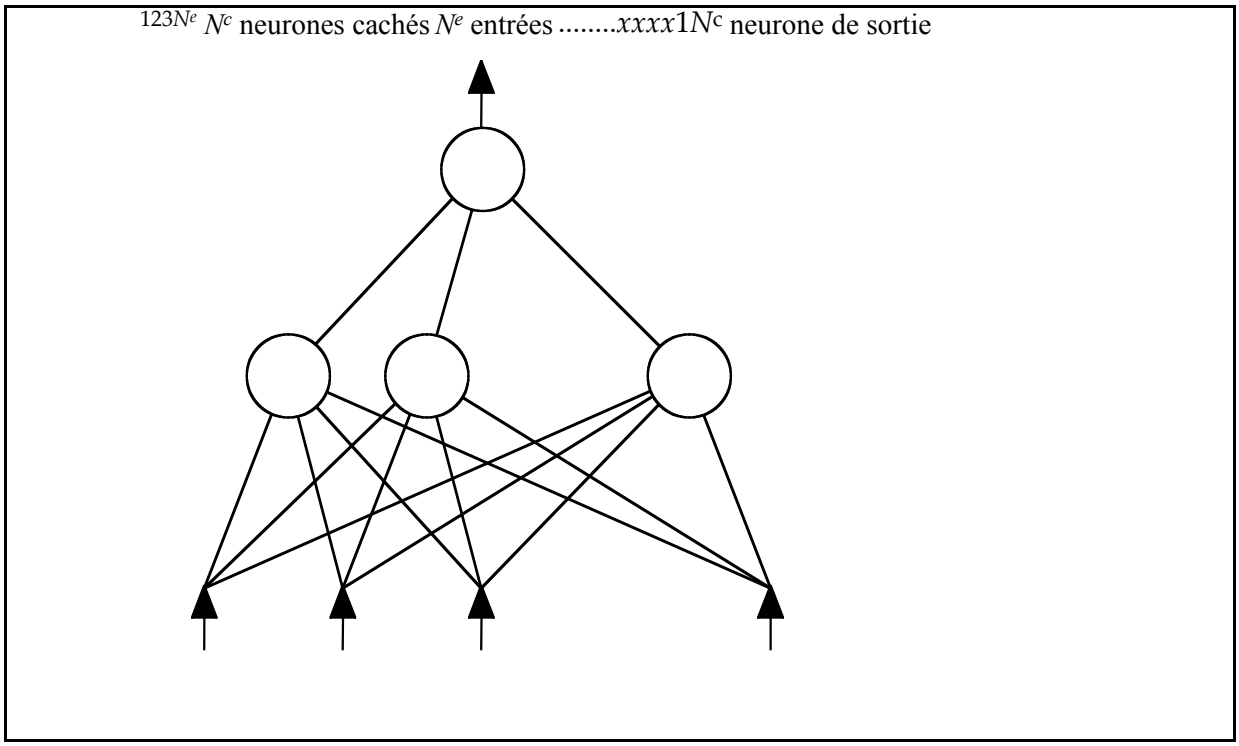


Figure 6.1 : *Réseau à couches avec N_e entrées, N_c neurones cachés et un neurone de sortie.*

Les neurones de la couche cachée sont appelés *neurones cachés*. Une fois l'architecture à deux couches choisie, il faut fixer le nombre de neurones cachés. Plus ce nombre est élevé, plus le nombre de degrés de liberté est élevé et plus la fonction modélisée par le réseau de neurone peut être complexe.

La Figure 6.2 montre deux exemples de fonctions réalisées par un réseau de neurones ; la partie gauche est obtenue avec un réseau comportant deux neurones cachés et la partie droite avec un réseau comportant dix neurones cachés. Dans le deuxième cas, la fonction obtenue comporte plus de degrés de liberté.

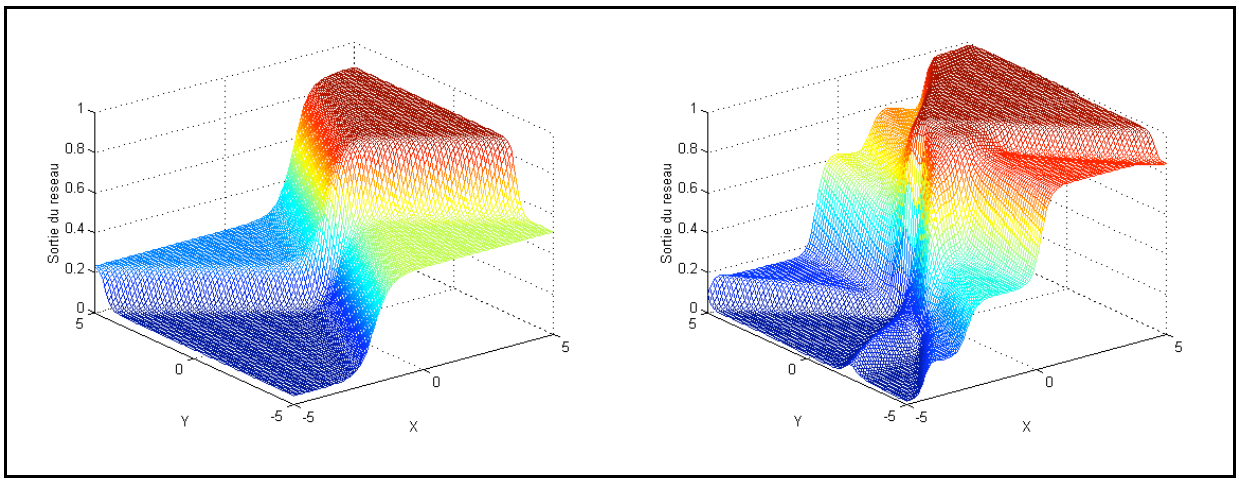


Figure 6.2 : *Réseau à deux couches avec deux entrées et un biais, les fonctions d'activations des neurones cachés sont des fonctions sigmoïdes et la sortie est une fonction logistique. Le quadrant de gauche est la sortie d'un réseau de neurones à deux neurones cachés, le quadrant de droite est la sortie d'un réseau à dix neurones cachés. Les poids de la première couche sont choisis aléatoirement dans l'intervalle [-4 ; +4], les poids de la deuxième couche sont choisis aléatoirement dans l'intervalle [-2 ; +2].*

6.2.3 Propriétés des réseaux de neurones

Les réseaux de neurones à couches, présentés au paragraphe précédent, ont la propriété générale d'être des approximateurs universels parcimonieux. Il s'agit en fait de deux propriétés distinctes détaillées ci-dessous.

6.2.3.1 La propriété d'approximation universelle

La propriété d'approximation universelle a été démontrée par [Cybenko, 1989] et [Funahashi, 1989] et peut s'énoncer de la façon suivante :

Toute fonction bornée suffisamment régulière peut être approchée uniformément, avec une précision arbitraire, dans un domaine fini de l'espace de ses variables, par un réseau de neurones comportant une couche de neurones cachés en nombre fini, possédant tous la même fonction d'activation, et un neurone de sortie linéaire.

Cette propriété justifie l'utilisation de l'architecture présentée précédemment. Comme le montre ce théorème, le nombre de neurones cachés doit être choisi convenablement pour obtenir la précision voulue.

6.2.3.2 La propriété de parcimonie

Lorsque l'on cherche à modéliser un processus à partir des données, on s'efforce toujours d'obtenir les résultats les plus satisfaisants possibles avec un nombre minimum de paramètres ajustables. Dans cette optique, [Hornik *et al.*, 1994] ont montré que :

Si le résultat de l'approximation (c'est-à-dire la sortie du réseau de neurones) est une fonction non linéaire des paramètres ajustables, elle est plus parcimonieuse que si elle est une fonction linéaire de ces paramètres. De plus, pour des réseaux de neurones à fonction d'activation sigmoïdale, l'erreur commise dans l'approximation varie comme l'inverse du nombre de neurones cachés, et elle est indépendante du nombre de variables de la fonction à approcher. Par conséquent, pour une précision donnée, donc pour un nombre de neurones cachés donné, le nombre de paramètres du réseau est proportionnel au nombre de variables de la fonction à approcher.

Ce résultat s'applique aux réseaux de neurones à fonction d'activation sigmoïdale puisque la sortie de ces neurones n'est pas linéaire par rapports aux poids synaptiques. Cette propriété montre l'intérêt des réseaux de neurones par rapport à d'autres approximateurs comme les polynômes dont la sortie est une fonction linéaire des paramètres ajustables : pour un même nombre d'entrées, le nombre de paramètres ajustables à déterminer est plus faible pour un réseau de neurones que pour un polynôme. Cette propriété devient d'autant plus intéressante dans le cas du filtrage de textes car le nombre d'entrées est typiquement de l'ordre de plusieurs dizaines.

6.3 Apprentissage des réseaux de neurones

Une fois l'architecture d'un réseau de neurones choisie, il est nécessaire d'effectuer un apprentissage pour déterminer les valeurs des poids permettant à la sortie du réseau de

neurones d'être aussi proche que possible de l'objectif fixé. Dans le cas d'un problème de régression, il s'agit d'approcher une fonction continue, dans le cas d'un problème de classification supervisée, il s'agit de déterminer une surface de séparation.

Cet apprentissage s'effectue grâce à la minimisation d'une fonction, appelée *fonction de coût*, calculée à partir des exemples de la base d'apprentissage et de la sortie du réseau de neurones ; cette fonction détermine l'objectif à atteindre.

Dans les travaux présentés dans ce mémoire, nous avons effectué cette minimisation en deux temps : un algorithme de descente du gradient, simple à mettre en œuvre et efficace loin du minimum, commence la minimisation, puis une méthode de quasi-Newton, très efficace proche du minimum, la termine[1].

Dans la suite de ce chapitre, nous utiliserons les notations suivantes : N est le nombre d'exemples de la base d'apprentissage ; à chaque exemple i est associée sa classe t_i (codée +1 ou 0) ; chaque exemple est représenté par un vecteur x_i de dimension n ; les poids du réseau sont représentés par un vecteur w ; la sortie du réseau de neurones associée au vecteur d'entrée x_i est notée y_i.

6.3.1 Algorithmes de minimisation

6.3.1.1 Principe des algorithmes

Soit *J(w)* la fonction de coût (le choix de la forme de cette fonction est expliqué au paragraphe 6.3.2). Les algorithmes utilisés nécessitent que *J(w)* soit dérivable par rapport aux poids. Le principe de ces méthodes est de se placer en un point initial, de trouver une direction de descente du coût dans l'espace des paramètres w, puis de se déplacer d'un pas dans cette direction. On atteint un nouveau point et l'on itère la procédure jusqu'à satisfaction d'un critère d'arrêt. Ainsi, à l'itération k, on calcule :

$$w_k = w_{k-1} + \square_{k-1} . d_{k-1}$$

$\square_k$ est le pas de la descente et d_k est la direction de descente : les différents algorithmes se distinguent par le choix de ces deux quantités.

---

[1] La méthode de Levenberg-Marquardt, également très efficace, ne s'applique qu'aux fonctions de coût quadratiques, ce qui n'est pas le cas dans les travaux que nous présentons.

6.3.1.2 Descente du gradient

L'algorithme le plus simple consiste à choisir comme direction de descente l'opposé du gradient de la fonction de coût ($d_k = -\nabla J(w) = -\text{Grad}(J(w_k))$). Cette méthode est efficace loin du minimum et permet uniquement de s'en approcher. Pour cette raison, la détermination du pas n'est pas cruciale : loin du minimum, il faut seulement vérifier que le pas n'est ni trop petit ni trop grand. En pratique, on utilise, selon les cas, deux méthodes :

- soit un asservissement par la norme du gradient :

$$\varepsilon_k = \frac{\varepsilon_0}{1 + \left\| \nabla J(w_k) \right\|}$$

  où ε_0 est une constante qui vaut typiquement 0,01.

- soit la méthode de Goldstein [Minoux, 1983] pour laquelle le pas est adapté afin de satisfaire deux conditions :

1. $J(w_k + \rho_k d_k) < J(w_k) + m_1 \rho_k \nabla J^T(w_k) d_k$
2. $J(w_k + \rho_k d_k) > J(w_k) + m_2 \rho_k \nabla J^T(w_k) d_k$

La première condition s'assure que le pas choisi n'est pas trop grand (sinon l'algorithme risque d'avoir un comportement oscillatoire), alors que la deuxième s'assure qu'il n'est pas trop petit (sinon l'algorithme a une convergence très lente). Les valeurs habituelles pour les deux paramètres m_1 et m_2 sont respectivement 0,1 et 0,7.

Ces deux méthodes de recherche du pas sont "économiques", car elles ne demandent pas de calculs inutiles de gradient (seul celui dans la direction de descente est nécessaire).

6.3.1.3 La méthode de Newton

La méthode de Newton utilise la courbure (dérivée seconde) de la fonction de coût pour atteindre le minimum. La modification des paramètres s'écrit ainsi :

$$w_k = w_{k-1} - H_{k-1}^{-1} . \nabla J\left(w_{k-1}\right)$$

La direction de descente est $-H_{k-1}^{-1} . \nabla J(w_{k-1})$ où H_{k-1}^{-1} est l'inverse du hessien de la fonction de coût, et le pas est constant fixé à un.

Cet algorithme converge en une seule itération pour une fonction quadratique. C'est donc un algorithme qui est inefficace loin du minimum de la fonction et très efficace près du minimum.

Dans la pratique, le calcul du hessien et surtout de son inverse est à la fois complexe et source d'instabilités numériques ; on utilise de préférence une méthode de "quasi-Newton".

6.3.1.4 La méthode de quasi-Newton

Les méthodes de quasi-Newton consistent à approcher l'inverse du hessien plutôt que de calculer sa valeur exacte.

La modification des paramètres s'écrit :

$$w_k = w_{k-1} - \eta_{k-1}.M_{k-1}.\nabla J(w_{k-1})$$

La suite M_k est construite de façon à converger vers l'inverse du hessien avec M_0 égale à la matrice identité. Cette suite est construite grâce à la méthode dite BFGS [Broyden, 1970] [Flechter, 1970] [Goldfarb, 1970] [Shanno, 1970], dont la vitesse de convergence est beaucoup plus grande que celle de la méthode du gradient. De plus, elle est relativement insensible au choix du pas, qui peut être déterminé économiquement par la méthode de Goldstein.

6.3.1.5 Problème des minima locaux

Les minima trouvés par les algorithmes précédents sont des minima locaux. Le minimum trouvé dépend du point de départ de la recherche c'est-à-dire de l'initialisation des poids. En pratique, il faut effectuer plusieurs minimisations avec des initialisations différentes, pour trouver plusieurs minima et retenir le "meilleur". Il est néanmoins impossible et généralement inutile, de s'assurer que le minimum choisi est le minimum global. Les réseaux de neurones à couches présentent des symétries, si bien que l'on peut montrer que pour une architecture avec N_c neurones cachés, il existe $2^N c\ N_c!$ minima équivalents [Bishop, 1995].

6.3.2 Choix de la fonction de coût

Le choix de la fonction de coût est conditionné par l'objectif à atteindre.

6.3.2.1 Erreur quadratique

Pour les problèmes de régression, l'ensemble d'apprentissage est constitué d'exemples pour lesquels la sortie désirée t est une variable continue. La fonction de coût la plus utilisée est l'erreur quadratique sur la base d'apprentissage : elle consiste à minimiser la somme des carrés des erreurs entre la sortie du réseau et la valeur réelle de la sortie.

$$J(w) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i(w) - t_i\right)^2$$

Cette fonction de coût est issue du principe de maximum de vraisemblance avec une hypothèse gaussienne sur la distribution des sorties. Pour les problèmes de classification à deux classes, la sortie désirée est une variable binaire codée 1 ou 0 selon que l'exemple appartient respectivement à C_1 ou C_0. L'hypothèse gaussienne sur la distribution des sorties n'est alors clairement plus vérifiée. Cependant, si l'apprentissage est effectué en minimisant l'erreur quadratique, la sortie du réseau de neurones peut être interprétée comme la probabilité *a posteriori*, au sens du théorème de Bayes, d'appartenance à la classe C_1 [Richard et Lippman, 1991].

6.3.2.2 Entropie croisée

L'entropie croisée, comme l'erreur quadratique moyenne est issue du principe du maximum de vraisemblance. Comme l'hypothèse sous-jacente pour l'utilisation de l'erreur quadratique est erronée pour les problèmes de classification, un autre modèle est construit pour tenir compte de la spécificité du codage utilisé dans ces problèmes.

Considérons un problème de classification à deux classes C_1 et C_0 où les sorties t sont codées 1 ou 0. Afin que la sortie du réseau de neurones approche la probabilité *a posteriori* d'appartenir à la classe C_1, considérons tout d'abord la probabilité d'observer l'une ou l'autre des valeurs de la sortie t en un point de l'espace x si la sortie du modèle est y :

$$p(t|x) = y^t.(1 - y)^{1-t}$$

La probabilité d'observer l'ensemble d'apprentissage en supposant que les données sont indépendantes s'écrit :

$$\prod_{i=1}^{N} (y_i)^{t_i}(1 - y_i)^{1-t_i}$$

Pour maximiser cette fonction, on préfère minimiser l'opposé de son logarithme. La fonction de coût utilisée est donc finalement :

$$J(w) = -\sum_{i=1}^{N}\left\{t_i.\ln\, y_i(w) + (1 - t_i).\ln\,(1 - y_i(w))\right\}$$

Cette fonction, appelée *entropie croisée*, atteint son minimum lorsque $t_i = y_i$ pour tout i. Par construction, la sortie du réseau est interprétée comme la probabilité *a posteriori* d'appartenir à la classe C_1.

6.3.3 Calcul du gradient de la fonction de coût

Les méthodes de minimisation exposées au paragraphe 6.3.1 nécessitent le calcul du gradient de la fonction de coût par rapport aux poids du réseau. Les fonctions de coût présentées étant additives, le gradient total est la somme de tous les gradients partiels calculés pour chacun des exemples de la base d'apprentissage :

$$\nabla J(w) = \sum_{i=1}^{N} \nabla J^i(w)$$

Pour chaque exemple, le gradient partiel $\nabla J^i(w)$ est effectué de manière économique grâce à l'algorithme de rétropropagation [Rumelhart *et al.*, 1986]. La mise en œuvre de cet algorithme, nécessite l'expression analytique de la quantité $\frac{\partial J(w)}{\partial y_i(w)}$ où $y_i(w)$ est la sortie du réseau pour l'exemple *i*.

- Si la fonction de coût est l'erreur quadratique, alors $\frac{\partial J(w)}{\partial y_i} = (y_i - t_i) = e_i$: c'est la différence entre la sortie du réseau et la sortie désirée, c'est-à-dire l'erreur de modélisation, on parle alors de "rétropropagation de l'erreur".
- Si la fonction de coût est l'entropie croisée, alors $\frac{\partial J(w)}{\partial y_i} = \frac{y_i - t_i}{y_i.(1 - y_i)}$.

La modification des poids peut être effectuée, soit après chaque calcul de gradient partiel, soit après le calcul du gradient total. Dans toute la suite de ce mémoire, les modifications sont effectuées après le calcul du gradient total.

6.4 Le problème de surajustement

6.4.1 Définition du surajustement

Si l'on considère un ensemble d'apprentissage et une fonction de coût quadratique, en vertu de la propriété d'approximation universelle exposée au paragraphe 6.2.3.1, il est toujours possible d'obtenir une fonction de coût aussi petite que l'on veut sur l'ensemble d'apprentissage, à condition de mettre suffisamment de neurones cachés. Cependant, le but de l'apprentissage n'est pas d'apprendre exactement la base d'apprentissage, mais le modèle sous-jacent qui a servi à engendrer les données. Or, si la fonction apprise par le réseau de neurones est ajustée

trop finement aux données, elle apprend les particularités de la base d'apprentissage au détriment du modèle sous-jacent : le réseau de neurones est *surajusté*.

6.4.2 Biais et variance des modèles

Le surajustement est souvent expliqué grâce aux concepts de biais et variance introduits dans la communauté des réseaux de neurones par [Geman *et al.*, 1992].

Si l'on considère plusieurs ensembles d'apprentissage, le biais rend compte de la différence moyenne entre les modèles et l'espérance mathématique de la grandeur à modéliser. Le biais est donc lié à la valeur du bruit du processus que l'on cherche à modéliser. La variance rend compte des différences entre les modèles selon la base d'apprentissage utilisée.

On parle souvent de compromis entre le biais et la variance. Si un modèle est trop simple par rapport au processus à modéliser, alors son biais est élevé, mais sa variance est faible puisqu'il est peu influencé par les données. Si un modèle est trop complexe, son biais est faible puisqu'il est capable de s'ajuster exactement à la base d'apprentissage, mais sa variance est élevée puisqu'une nouvelle base avec une réalisation différente du bruit peut entraîner un modèle très différent : c'est le cas du surajustement.

Ainsi, la complexité du modèle doit être ajustée pour trouver un compromis entre le biais et la variance. Dans leur article [Geman *et al.*, 1992] contrôlent la complexité du modèle et donc le surajustement en limitant le nombre de neurones cachés.

Cependant [Gallinari et Cibas, 1999] ont montré que cette vision théorique avait des limites pour un réseau à couches dont l'apprentissage était effectué avec une base d'apprentissage comprenant peu d'exemples. En étudiant différentes architectures pour un problème de régression, ils ont montré que le biais et la variance n'évoluent pas nécessairement en sens contraire lorsque le nombre de neurones cachés augmente. Dans leur cas, un modèle avec quinze neurones cachés à une variance plus élevée qu'un modèle avec soixante neurones cachés.

En résumé, le surajustement ne s'explique pas seulement par le compromis biais-variance, notamment lorsque le nombre d'exemples est faible. De plus, l'interprétation du surajustement en ces termes a été développée pour les problèmes de régression et ne se transpose pas simplement aux problèmes de classification.

6.4.3 Deux exemples artificiels de surajustement

Nous présentons ci-dessous deux problèmes artificiels pour illustrer simplement comme se manifeste le phénomène de surajustement. Le premier problème est un exemple de régression : le réseau de neurones doit approcher une fonction continue ; le deuxième problème est un exemple de classification : le réseau de neurones doit définir une frontière de séparation.

Ces deux exemples artificiels de nature différente montrent que le surajustement se traduit différemment selon le problème.

6.4.3.1 Le surajustement pour les problèmes de régression

Dans le cas d'une régression, les données de la base d'apprentissage sont bruitées. Donc, si le modèle possède trop de degrés de liberté, il peut s'ajuster localement à certains points, et apprendre la réalisation particulière du bruit sur la base d'apprentissage et non pas le processus lui-même.

Supposons que l'on cherche à modéliser un processus f comme celui qui est représenté sur la Figure 6.3. On dispose d'un ensemble d'apprentissage constitué de cinquante points choisis aléatoirement auquel est ajouté un bruit gaussien ε de variance $5{,}0.10^{-3}$.

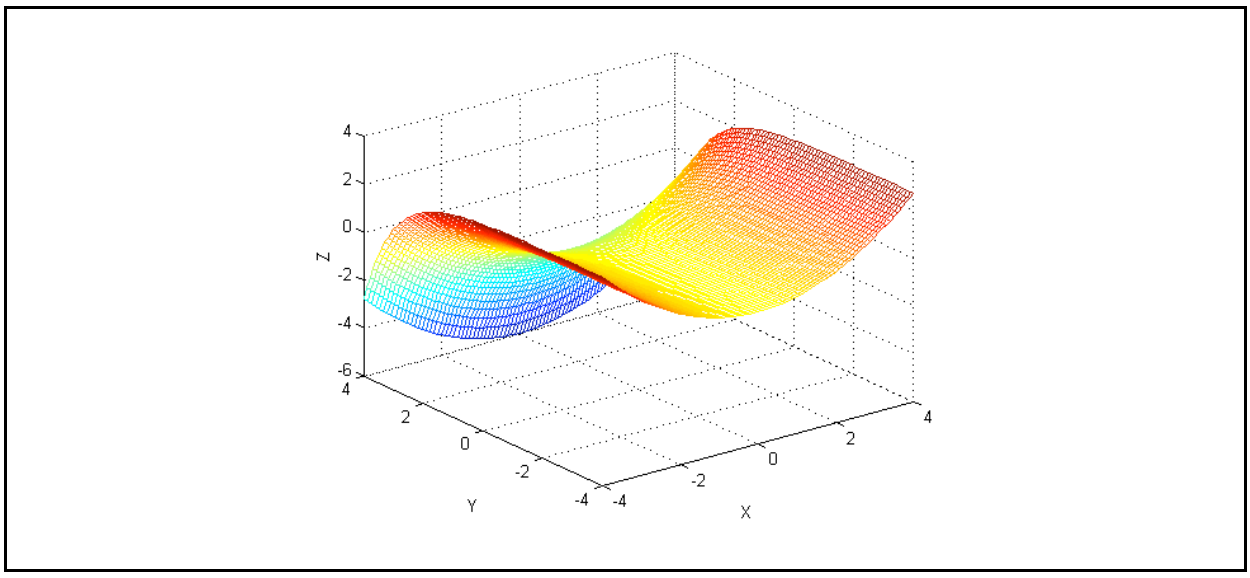


Figure 6.3 : *Représentation de la fonction* $z = f(x, y) = -0{,}5 + 0{,}2\,x^2 - 0{,}1\,\exp(y)$.

L'ensemble d'apprentissage est constitué de cinquante exemples pour lesquels la sortie t_i est calculée par :

$$t_i = f(x_i, y_i) + \varepsilon_i$$

L'apprentissage est réalisé en minimisant l'erreur quadratique moyenne sur l'ensemble d'apprentissage. Les surfaces modélisées après apprentissage par un réseau à trois neurones cachés et par un réseau à dix neurones cachés sont représentées sur la Figure 6.4. Le coût quadratique sur la base d'apprentissage est plus faible avec le réseau comprenant dix neurones cachés qu'avec le réseau en contenant deux ($6{,}0.10^{-4}$ contre $3{,}7.10^{-3}$). On voit nettement sur cette figure que le réseau avec dix neurones cachés a utilisé ses degrés de liberté pour s'ajuster localement à certains points et que le modèle trouvé est loin de la surface théorique de la Figure 6.3, contrairement à la surface modélisée par le réseau avec trois neurones cachés.

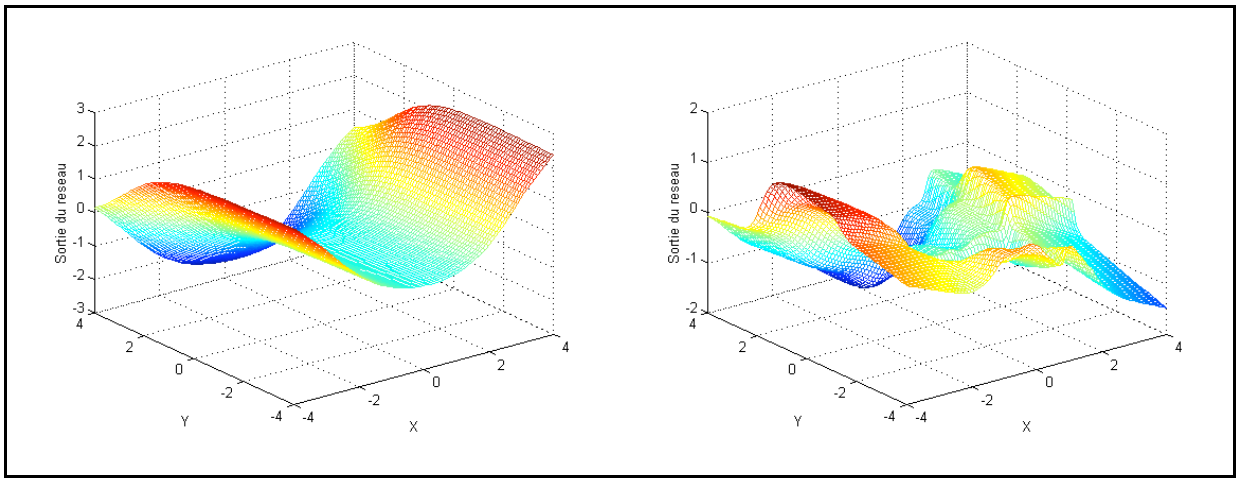


Figure 6.4 : *Surfaces modélisées par le réseau de neurones après apprentissage. Quadrant de gauche : réseau avec deux neurones cachés. Quadrant de droite : réseau avec dix neurones cachés.*

Dans cet exemple artificiel, la variance du modèle à dix neurones cachés est donc grande alors que son biais est faible ; le modèle avec trois neurones cachés semble être un bon compromis entre les deux exigences.

6.4.3.2 Le surajustement pour les problèmes de classification supervisée

Pour la classification supervisée, les données ne sont pas bruitées puisque l'on considère que le superviseur qui attribue les classes sur l'ensemble d'apprentissage ne fait pas d'erreur. Cependant, il arrive que, pour un même point de l'espace des entrées, la probabilité d'appartenance à une classe ne soit pas égale à 1 ou 0 : le problème n'est pas linéairement séparable. La sortie du réseau doit alors être la probabilité *a posteriori* au sens du théorème de

Bayes pour ce point. Si le réseau s'ajuste trop finement à la base d'apprentissage, il surestime ou sous-estime cette probabilité.

Considérons un problème de classification à deux classes avec deux entrées x et y issues de distributions gaussiennes notées $N(\square ; \square)$ où $\square$ est la moyenne de la distribution et $\square$ son écart-type. Pour la première classe, la distribution selon x est une combinaison de deux distributions $N(-2 ; 0,5)$ et $N(0 ; 0,5)$, et celle selon y est issue de $N(0 ; 0,5)$. Pour la deuxième classe, la distribution est issue de $N(-1 ; 1)$ pour x, et $N(1 ; 0,5)$ pour y.

La Figure 6.5 montre le résultat d'un tirage aléatoire des points pour les deux classes et la probabilité *a posteriori* calculée grâce à la formule de Bayes (pour ce problème artificiel, les densités de probabilités sont connues et il est donc possible de calculer la probabilité *a posteriori* théorique). La surface de cette figure est la sortie idéale que doit avoir un réseau de neurones après l'apprentissage.

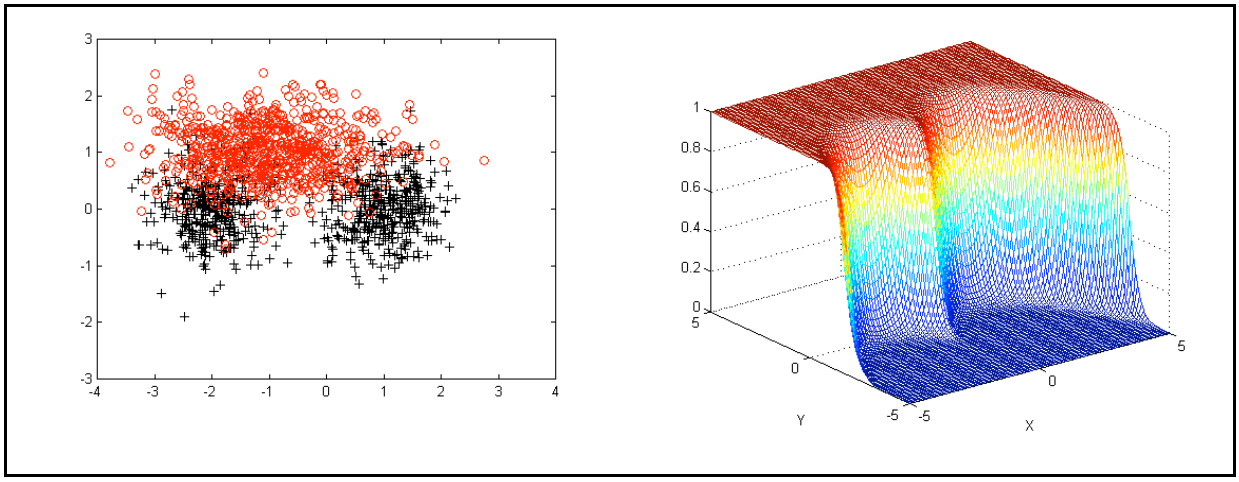


Figure 6.5 : *Répartition des points et distribution théorique de la probabilité a posteriori.*

L'un des causes du surajustement est le trop grand nombre de degrés de liberté de la fonction par rapport au modèle. Ce problème est illustré par la Figure 6.6 : la base d'apprentissage est constituée de 500 points, les réseaux de neurones sont des réseaux à couches dont on fait varier la complexité grâce au nombre de neurones cachés. Le modèle de gauche avec deux neurones cachés est bien adapté au modèle : la sortie du réseau est très proche de la sortie théorique. Le modèle de droite avec dix neurones cachés dispose clairement de trop de neurones cachés et s'ajuste pour passer exactement par certains points.

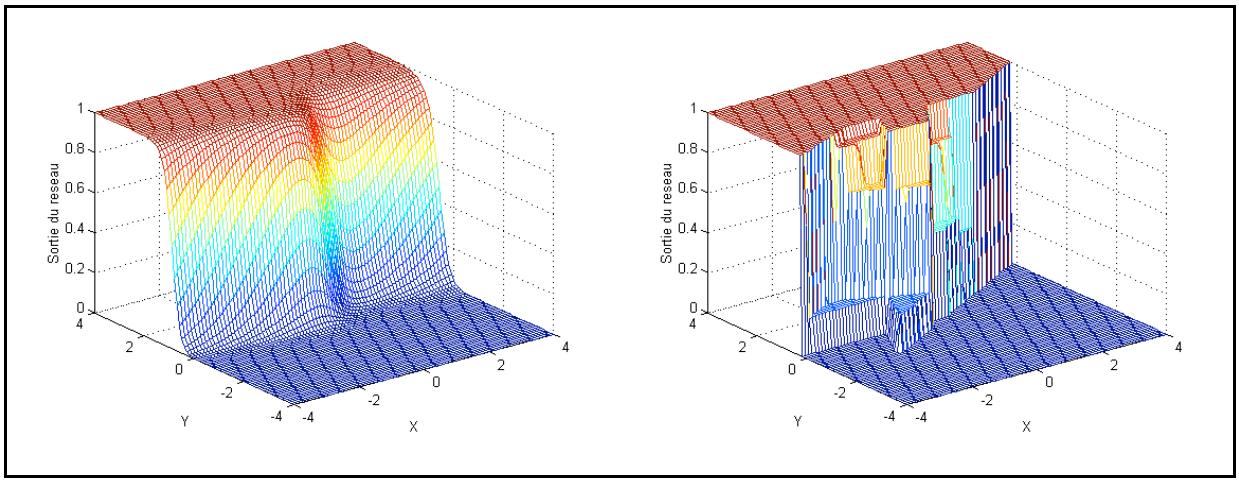


Figure 6.6 : *Sortie d'un réseau avec deux neurones cachés à gauche et dix neurones cachés à droite après apprentissage sur un ensemble de 500 points.*

Dans ce cas, si l'on sait détecter le surajustement, il suffit de réduire le nombre de neurones cachés pour trouver la bonne architecture.

Lorsque le nombre de points disponibles pour l'apprentissage diminue, le phénomène précédent s'accentue et le surajustement peut être observé même pour des architectures très simples. La Figure 6.7 montre la sortie d'un réseau à deux neurones cachés après un apprentissage avec un ensemble contenant cinquante points.

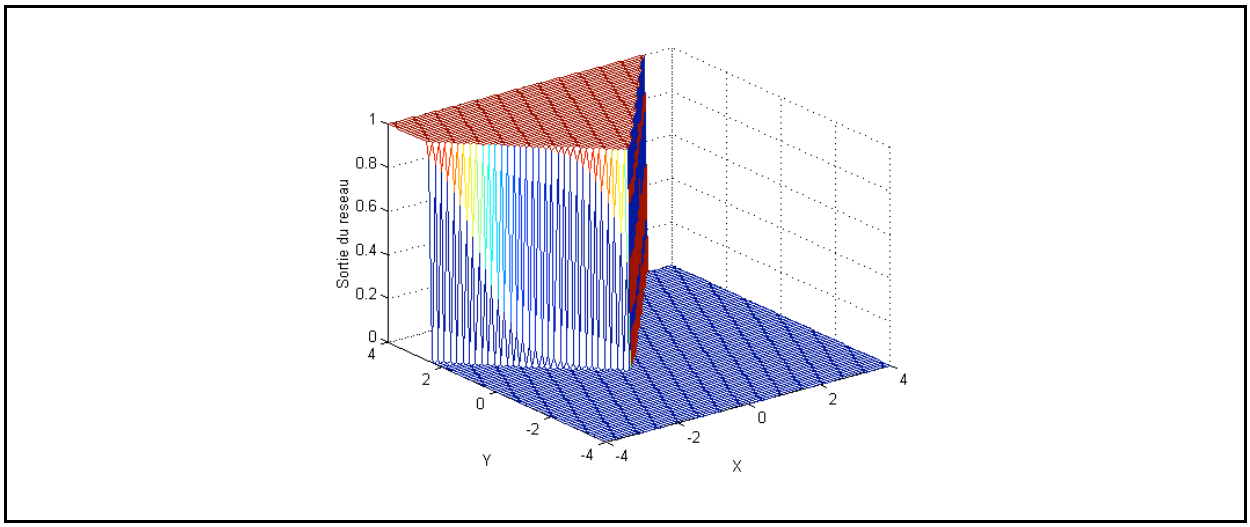


Figure 6.7 : *Sortie d'un réseau avec deux neurones cachés après apprentissage avec un ensemble de cinquante points.*

Dans ce cas, la sortie est un échelon, dont la frontière dépend grandement de la base d'apprentissage puisque cette frontière se place selon les points qui figurent dans cette zone :

le modèle trouvé a une variance élevée. De plus, pour un point situé à proximité de la frontière, le réseau produit une sortie qui vaut 0 ou 1 et qui a donc la même valeur que pour un point situé par exemple en (-2, -2) et dont la classe n'est pas du tout ambiguë. Dans ce cas, la sortie du réseau n'est plus une probabilité, mais une sortie binaire, il n'est plus possible notamment de classer les exemples par ordre de pertinence : le réseau de neurones ne fait plus de nuance. Il n'est plus possible de tracer des courbes rappel-précision, ni de changer le seuil de décision afin d'obtenir un filtre favorisant la précision ou le rappel.

6.4.3.3 Conclusions sur l'étude de ces deux exemples

Pour la classification, comme pour la régression, et pour une architecture donnée, le surajustement est d'autant plus marqué que le nombre d'exemples est faible par rapport à la dimension du vecteur d'entrée et la complexité de la fonction à approcher.

Ces exemples montrent que dans le cas d'un problème de classification, le surajustement est beaucoup plus localisé que pour un problème de régression. Dans le premier cas, le surajustement intervient dans la zone frontière entre les deux classes, alors que dans le second, il se fait sur l'ensemble du domaine.

6.5 Les méthodes pour limiter le surajustement

On distingue deux familles de méthodes pour prévenir le surajustement : les méthodes passives et les méthodes actives. Les philosophies de ces deux familles de méthodes sont différentes.

- Les méthodes passives essayent de détecter le surajustement *a posteriori* pour supprimer les mauvais modèles. Parmi les méthodes les plus classiques figurent l'utilisation d'une base de validation pendant l'apprentissage, et les mesures de critère d'information.
- Les méthodes actives interviennent pendant la phase d'apprentissage pour empêcher le modèle de faire du surajustement. Les méthodes de régularisation comme l'arrêt prématuré ou la pénalisation entrent dans ce cadre.

6.5.1 Les méthodes passives

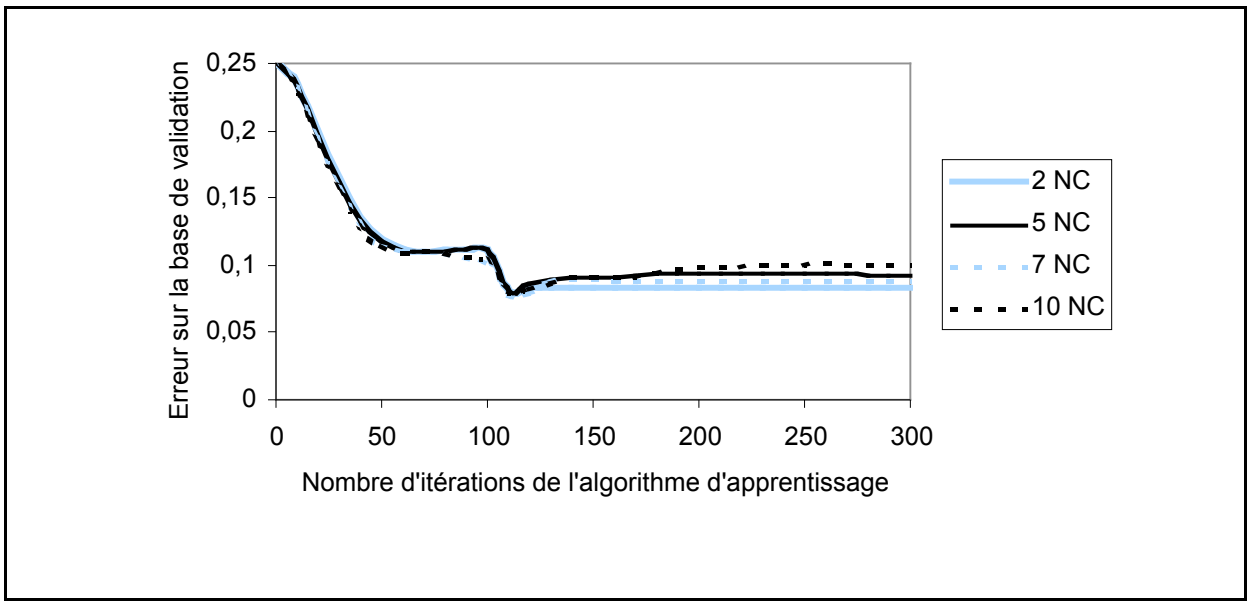
6.5.1.1 Utilisation d'une base de validation pendant l'apprentissage

Le principe consiste à mesurer les performances pendant l'apprentissage sur une base de validation qui est différente de la base d'apprentissage. Lorsque le modèle n'est pas trop ajusté

aux données de l'apprentissage, les fonctions de coût sur la base de validation et d'apprentissage diminuent ensemble. Lorsque le modèle commence à être surajusté, la fonction de coût sur la base d'apprentissage continue de diminuer, alors que la fonction de coût sur la base de validation augmente.

Cette méthode est surtout efficace pour les problèmes de régression, car comme l'a montré la Figure 6.4, le réseau tend à s'ajuster aux données sur l'ensemble de l'espace : les variations de la fonction de coût sur la base de validation sont plus facilement détectables. Dans l'exemple de la régression du paragraphe 6.4.3.1, si l'on mesure les performances sur une base de validation comprenant 500 exemples générés de la même manière que la base d'apprentissage, alors l'erreur quadratique commise par le réseau comprenant deux neurones cachés vaut $6{,}9.10^{-3}$ tandis qu'avec dix neurones cachés, cette erreur est de $2{,}4.10^{-2}$ ce qui montre que ce dernier a mal appris le processus.

Pour le problème de classification, le surajustement ne se produit pas uniformément sur l'espace, mais a tendance à apparaître dans les zones frontières entre les deux classes comme l'a montré l'exemple artificiel du paragraphe 6.4.3.2. Dans ce cas, la dégradation des performances sur une base de validation est moins évidente. Pour le problème de classification présenté au paragraphe 6.4.3.2, la Figure 6.8 montre l'évolution, pendant l'apprentissage, de l'erreur quadratique moyenne (EQMV) et du taux d'exemples mal classés sur une base de validation contenant 300 points (la base d'apprentissage est constituée de 500 exemples).



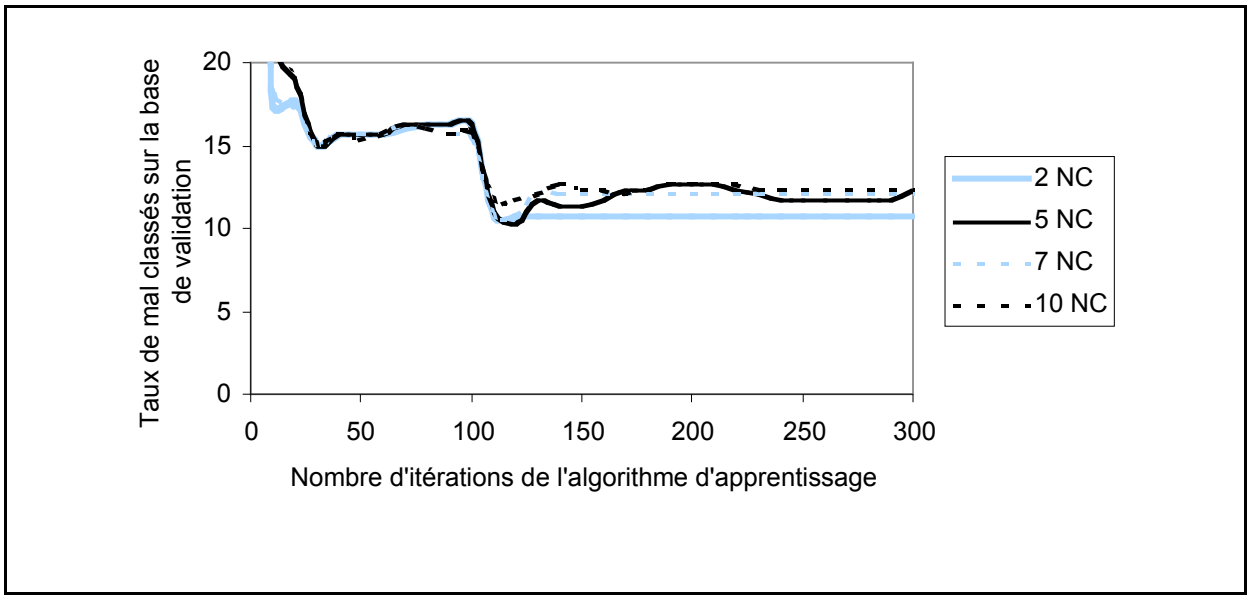


Figure 6.8 : *Évolution de la fonction de coût (EQMV) et du taux d'exemples mal classés sur la base de validation pour différentes architectures. L'axe des abscisses représente le nombre d'itérations de l'algorithme d'apprentissage.*

Ces courbes montrent que les performances sur la base validation se dégradent peu lorsque le nombre de neurones cachés augmente même pour le modèle avec dix neurones cachés, alors que la surface de la Figure 6.6 a montré clairement que le réseau de neurones était surajusté. Ceci est dû au fait que le surajustement se produit principalement dans la zone frontière et concerne peu de points de la base de validation.

En conclusion, cette méthode ou les méthodes dérivées comme le leave-one-out (cf. [Monari, 1999] pour une étude théorique et pratique du leave-one-out) ne semblent pas les plus adaptées pour éviter le surajustement dans les problèmes de classification.

6.5.1.2 Les critères d'information

Les critères d'information associent à chaque modèle une mesure qui tient compte à la fois de la qualité de l'approximation sur la base de l'apprentissage et de la complexité du modèle. On retrouve donc, en général, deux termes pour ces fonctions : le premier est d'autant plus petit que l'approximation du modèle sur la base d'apprentissage est bonne et le deuxième augmente avec la complexité du modèle. Le meilleur modèle est celui pour lequel cette mesure est la plus petite.

Parmi les mesures couramment utilisées figurent le critère d'Akaike [Akaike, 1974] ou le critère d'information développé par [Schwartz, 1978], et connu sous le nom de BIC. En pratique, ces mesures doivent être utilisées lorsque le nombre d'exemples d'apprentissage est grand devant le nombre de paramètres du modèle. Sur un exemple pour lequel le nombre d'exemples d'apprentissage est faible, ces critères conduisent à de mauvais modèles et ne sont pas utilisables [Gallinari et Cibas, 1999].

6.5.1.3 Conclusion sur les méthodes passives

Les méthodes passives, issues de la description du problème de surajustement en termes de biais-variance ne propose comme solution qu'une limitation de la complexité du modèle par l'intermédiaire d'une limitation du nombre de neurones cachés.

6.5.2 Les méthodes actives : les méthodes de régularisation

Les méthodes de régularisation, par opposition, peuvent être qualifiées d'actives, car elles ne cherchent pas à limiter la complexité du réseau, mais elles contrôlent la valeur des poids pendant l'apprentissage. Il devient possible d'utiliser des modèles avec un nombre élevé de poids et donc un modèle complexe, même si le nombre d'exemples d'apprentissage est faible.

[Bartlett, 1997] a montré que la valeur des poids était plus importante que leur nombre afin d'obtenir de modèles qui ne sont pas surajustés. Il montre, que si un grand réseau est utilisé et que l'algorithme d'apprentissage trouve une erreur quadratique moyenne faible avec des poids de valeurs absolues faibles, alors les performances en généralisation dépendent de la taille des poids plutôt que de leur nombre.

Plusieurs méthodes de régularisation existent dans la littérature, comme *l'arrêt prématuré* (*early stopping*) qui consiste à arrêter l'apprentissage avant la convergence ou les méthodes de pénalisation. Les méthodes de pénalisation ajoutent un terme supplémentaire à la fonction de coût usuelle afin de favoriser les fonctions régulières :

$$J' = J + \lambda\Omega$$

J est une fonction de coût comme celles présentées au paragraphe 6.3.2, et Ω est une fonction qui favorise les modèles réguliers. L'apprentissage est réalisé en minimisant la nouvelle fonction J'. Un modèle qui a bien appris la base d'apprentissage correspond à une valeur faible de J, alors qu'une fonction régulière correspond à une fonction Ω faible : l'apprentissage doit

trouver une solution qui satisfasse ces deux exigences. Parmi les différentes formes possibles pour la fonction Ω, la méthode du *weight decay* est souvent utilisée, car elle est simple à mettre en œuvre, et plusieurs études ont montré qu'elle conduisait à de bons résultats (voir par exemple [Hinton, 1987] [Krogh et Hertz, 1992] [Gallinari et Cibas, 1999]) ; de plus, elle trouve une interprétation théorique dans l'approche bayesienne développée au paragraphe 6.6.

6.5.2.1 Arrêt prématuré

Comme nous l'avons vu précédemment, l'apprentissage consiste à minimiser, grâce à un algorithme itératif, une fonction de coût calculée sur la base d'apprentissage. La méthode de l'arrêt prématuré (*early stopping*) consiste à arrêter les itérations avant la convergence de l'algorithme. Si la convergence n'est pas menée à son terme, le modèle ne s'ajuste pas trop finement aux données d'apprentissage : le surajustement est limité.

Pour mettre en œuvre cette méthode, il faut déterminer le nombre d'itérations à utiliser pendant l'apprentissage. La méthode la plus classique consiste à suivre l'évolution de la fonction de coût sur une base de validation, et à arrêter les itérations lorsque le coût calculé sur cette base commence à croître. Cependant, comme le montre la Figure 6.8, cette méthode peut être inapplicable, car il est difficile de déterminer avec précision le moment exact où il faut arrêter l'apprentissage puisque les performances sur la base de validation ne se dégradent pas nettement.

On préfère donc utiliser les méthodes de régularisation, d'autant que [Sjöberg, 1994] a montré que l'arrêt prématuré était identique à un terme de pénalisation dans la fonction de coût.

6.5.2.2 Weight Decay

Lorsque les poids du réseau sont grands en valeur absolue, les sigmoïdes des neurones cachés sont saturées, si bien que les fonctions modélisées peuvent avoir des variations brusques. Pour obtenir des fonctions régulières, il faut travailler avec la partie linéaire des sigmoïdes, ce qui implique d'avoir des poids dont la valeur absolue est faible.

Pour illustrer ce propos, on reprend le problème artificiel de classification introduit au paragraphe 6.4.3.2. La Figure 6.9 montre, pour différentes architectures, l'évolution de la somme des carrés des poids pendant l'apprentissage. Excepté pour le modèle comprenant deux neurones cachés, toutes les architectures obtiennent des poids très grands en valeur absolue.

Ces grandes valeurs des poids conduisent à des surfaces de séparation avec des variations brusques comme l'a montré la sortie du modèle comprenant dix neurones cachés (Figure 6.6).

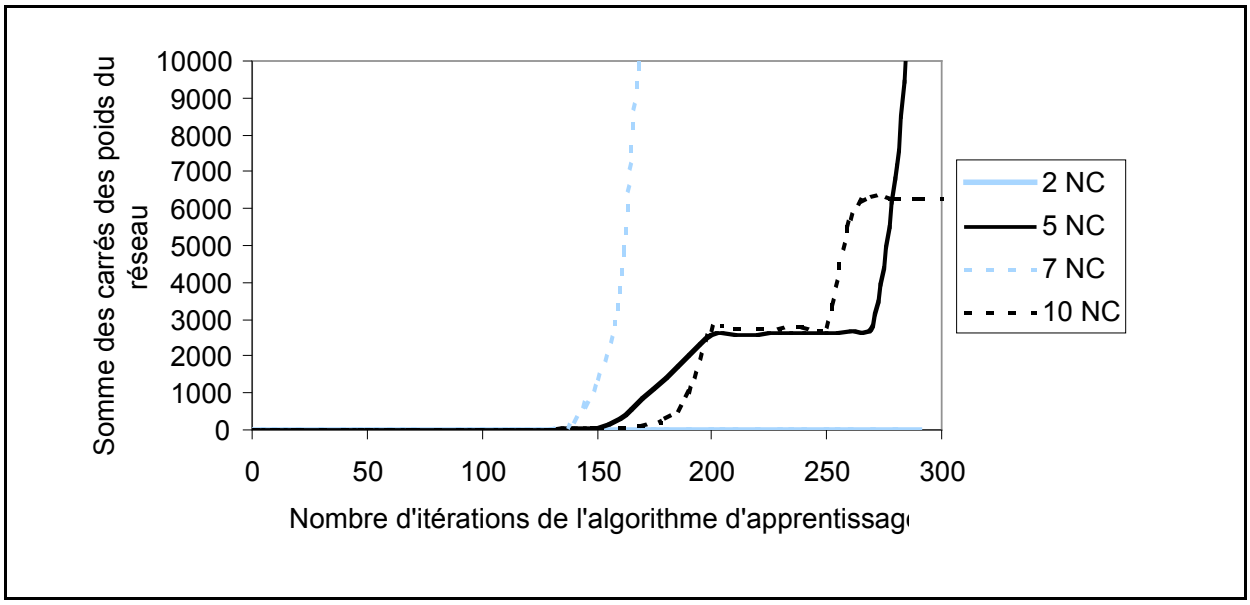


Figure 6.9 : *Évolution de la moyenne des carrés des poids* $\frac{1}{p}\sum_{i=1}^{p} w_i^2$ *en fonction du nombre d'itérations de l'algorithme d'apprentissage. La courbe correspondant à l'architecture à deux neurones cachés est confondue avec l'axe.*

La méthode de régularisation du *weight decay* limite la valeur absolue des poids en utilisant $\Omega = \frac{1}{2}\sum_{i=1}^{p} w_i^2$.

L'apprentissage s'effectue en minimisant :

$$J^{\Omega} = J + \frac{\alpha}{2}\sum_{i=1}^{p} w_i^2$$

où p est le nombre de poids que comporte le réseau.

Cette méthode est appelée *ridge regression* dans le cas de modèles linéaires par rapport aux paramètres [Saporta, 1990].

α est un *hyperparamètre* qui détermine l'importance relative des deux termes dans la nouvelle fonction de coût. Si α est trop grand, les poids tendent rapidement vers zéro, le modèle ne tient plus compte des données. Si α est trop petit, le terme de régularisation perd de son importance et le réseau de neurones peut donc être surajusté. Dans le cas intermédiaire, les poids après l'apprentissage ont des valeurs modérées.

Cette méthode présente l'avantage d'être très simple à mettre en œuvre, puisque le gradient de J' se calcule très simplement à partir du gradient de J et du vecteur des poids du réseau w :

$$\nabla J' = \nabla J + \alpha w$$

Il suffit d'ajouter la quantité αw au vecteur ∇J calculé par l'algorithme de rétropropagation.

En pratique, pour tenir compte du caractère différent des poids en fonction des couches, il faut considérer plusieurs hyperparamètres [MacKay, 1992b] :

$$J' = J + \frac{\alpha_1}{2} \sum_{w_i \in W_0} w_i^2 + \frac{\alpha_2}{2} \sum_{w_i \in W_1} w_i^2 + \frac{\alpha_3}{2} \sum_{w_i \in W_2} w_i^2$$

W_0 représente l'ensemble des poids reliant les biais aux neurones cachés, W_1 représente l'ensemble des poids reliant les entrées aux neurones cachés et W_3 représente l'ensemble des poids reliés au neurone de sortie (y compris le biais du neurone de sortie). Le modèle comprend trois hyperparamètres α_1, α_2, α_3, qui doivent être déterminés.

L'une des solutions consiste à tester plusieurs valeurs pour ces hyperparamètres et à conserver le meilleur modèle par une méthode de validation croisée. Mais comme il y a trois hyperpraramètres à déterminer, le nombre de valeurs à tester est rédhibitoire. L'approche bayesienne expliquée au paragraphe 6.6 propose une solution théorique pour déterminer ces valeurs.

6.5.3 Exemple d'utilisation des techniques de régularisation

L'exemple présenté ci-dessous illustre les notions de surajustement et montre l'impact de l'utilisation des méthodes d'arrêt prématuré et du *weight decay*. Il s'agit d'un exemple réel de filtrage de dépêches AFP ; le filtre sélectionne les dépêches relatives au thème des *participations* que nous avons déjà présenté au chapitre 5.

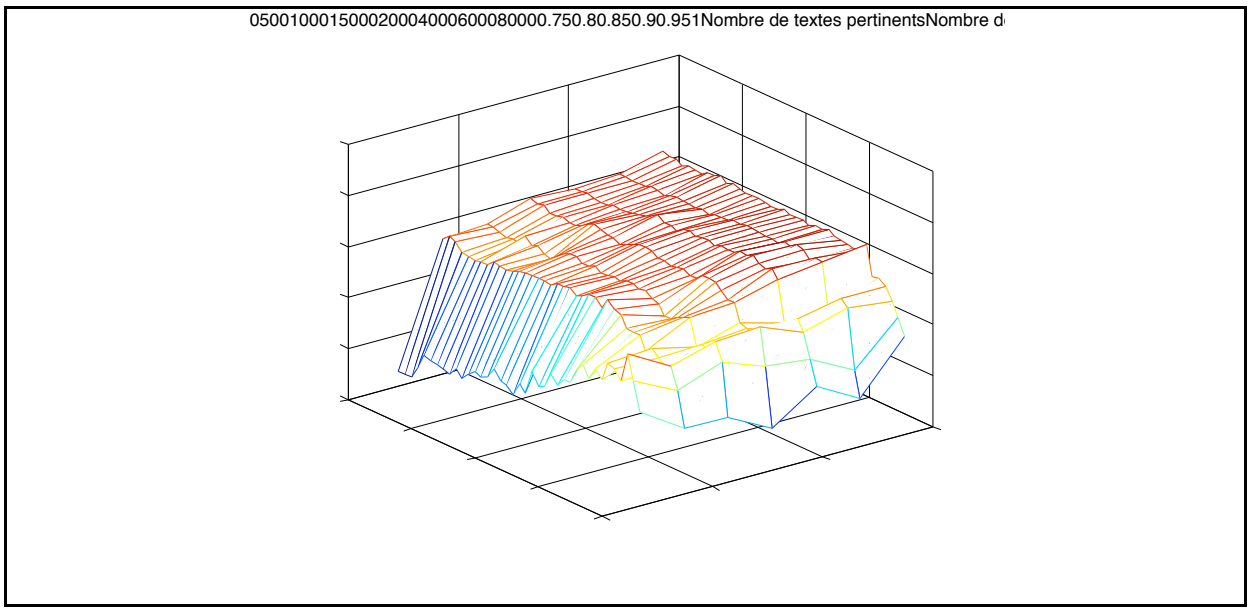
La base d'apprentissage est constituée de 1400 exemples de dépêches pertinentes au maximum et de 8000 dépêches non pertinentes. Plusieurs apprentissages sont réalisés avec un réseau contenant un unique neurone sigmoïde, et avec des tailles de la base d'apprentissage différentes ; les performances sont évaluées sur une base de test indépendante, qui comprend

200 dépêches pertinentes et 1000 dépêches non pertinentes. La mesure utilisée est la mesure *F*, le seuil de décision étant ajusté de façon à maximiser cette valeur sur la base de test.

La Figure 6.10 montre l'évolution des performances sur la base de test et l'évolution de la norme des poids, en fonction de l'évolution des proportions des exemples pertinents et non pertinents sur la base d'apprentissage. L'axe des abscisses représente le nombre de dépêches pertinentes présentes dans la base d'apprentissage et l'axe des ordonnées le nombre de dépêches non pertinentes. Pour chaque composition de la base d'apprentissage, la figure du haut rapporte les performances sur la base de test sur l'axe *z,* alors que celle du bas montre la norme euclidienne des poids du réseau après apprentissage. L'apprentissage est effectué sans aucune méthode de régularisation.

Les résultats montrent que lorsque le nombre d'exemples est faible, la norme euclidienne des poids est très grande et les performances sont faibles. Il n'est pas possible de simplifier l'architecture du réseau puisqu'il ne comporte qu'un seul neurone : l'utilisation d'une méthode de régularisation est obligatoire.

La méthode de l'arrêt prématuré a été utilisée sur le même problème de façon très simple : l'algorithme de minimisation implémente uniquement une descente de gradient simple. Plusieurs initialisations sont testées, et celle qui donne le coût le plus faible sur la base d'apprentissage est conservée. Les performances sont calculées sur la base de test comme précédemment.



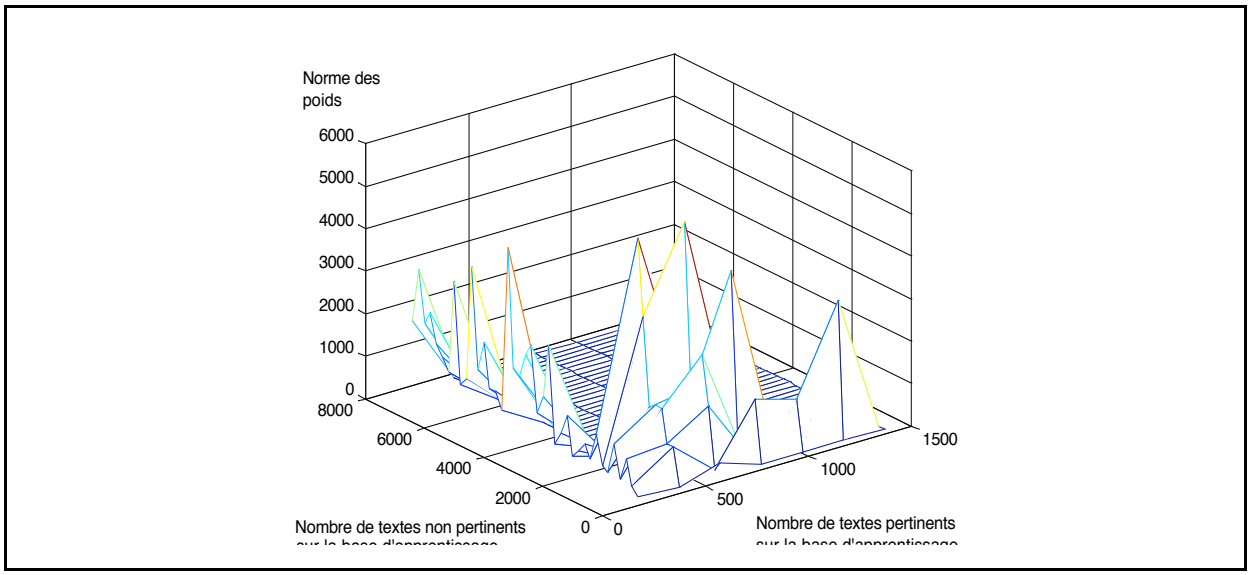


Figure 6.10 : *Évolution des performances (calcul de F sur la base de test), et de la norme euclidienne des poids, en fonction des proportions de dépêches pertinentes et non pertinentes dans l'ensemble d'apprentissage. Le nombre d'exemples pertinents varie de 200 à 1800 par pas de 200, et le nombre d'exemples non pertinents varie de 200 à 8000 par pas de 200. L'apprentissage est effectué par une descente de gradient suivi de la méthode de quasi-Newton sans aucun terme de régularisation.*

Les résultats, présentés à la Figure 6.11, montrent que, grâce à cette méthode, les performances dans la zone où le nombre d'exemples de la base d'apprentissage est faible sont nettement améliorées. En revanche, dans la zone où le nombre d'exemples est grand, les performances sont moins élevées qu'avec la méthode simple : notre implémentation de l'arrêt prématuré empêche d'exploiter toute la connaissance disponible dans la base d'apprentissage.

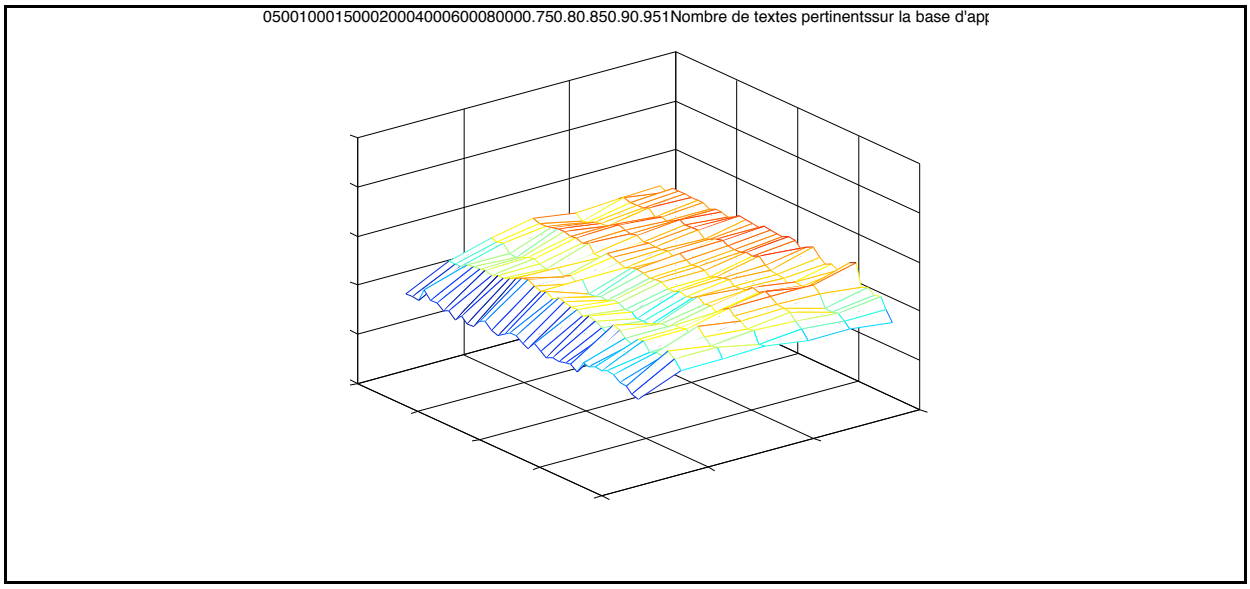


Figure 6.11 : *Arrêt prématuré : évolution des performances en fonction du nombre de dépêches pertinentes et non pertinentes sur l'ensemble d'apprentissage. Le réseau est un simple neurone sigmoïde, la minimisation de la fonction de coût s'effectue avec 800 itérations de gradient simple.*

Le graphe de la norme de poids n'est pas présenté, car dans tous les cas, les normes obtenues sont très faibles.

La méthode du *weight decay* a également été utilisée sur cet exemple, en utilisant deux hyperparamètres : un pour le biais (λ_b) et un pour les connexions entre les entrées et le neurone de sortie (λ_e). On ne s'intéresse pas, ici, à l'optimisation de ces hyperparamètres : leurs valeurs sont donc constantes durant l'apprentissage. Les résultats présentés à la Figure 6.12 montrent que, dans la zone où le nombre d'exemples est faible, les performances sont nettement améliorées par rapport à la méthode sans régularisation, et, dans la zone où le nombre d'exemples est élevé, les performances ne sont pas modifiées par rapport à l'optimum obtenu sans régularisation.

Comme précédemment, le graphe qui rend compte de l'évolution des normes n'est pas présenté, puisque les normes restent faibles quel que soit le nombre d'exemples d'apprentissage.

Cet exemple montre que le manque d'information contenu dans la base d'apprentissage peut être compensé avantageusement grâce à une méthode de régularisation comme l'arrêt prématuré ou le *weight decay*.

Ces résultats montrent la nécessité d'utiliser des méthodes de régularisation pour les problèmes de filtrage, car il est fréquent que le nombre d'exemples pertinents disponibles pour fabriquer un filtre ne dépasse pas la centaine. La méthode du *weight decay* semble préférable à la méthode de l'arrêt prématuré, car, quel que soit le nombre d'exemples disponibles, elle permet d'obtenir des résultats optimum.

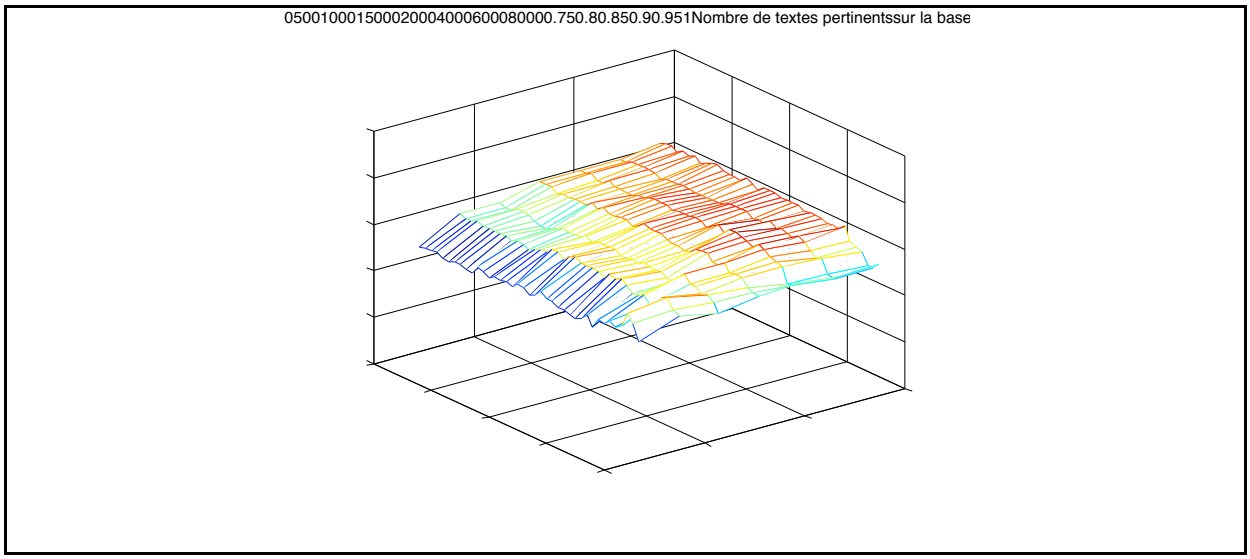


Figure 6.12 : *Weight decay : évolution des performances en fonction du nombre de dépêches pertinentes et non pertinentes sur l'ensemble d'apprentissage. Le réseau est un simple neurone sigmoïde. Les paramètres de régularisation sont* $\alpha_b = 0{,}001$ *et* $\alpha_e = 1$.

6.6 L'approche bayesienne

L'approche bayesienne a été appliquée ces dernières années aux réseaux de neurones par différents auteurs, notamment dans les travaux de [MacKay, 1992a], [MacKay, 1992b], [Neal, 1994], repris dans [Neal, 1996]) et [Buntine et Weigend, 1991] et plus récemment par [Thodberg, 1996]. Une synthèse de ces différentes approches peut être trouvée dans [Bishop, 1995].

6.6.1 Principe de l'approche bayesienne

Dans le paragraphe 6.3, l'apprentissage était effectué en trouvant une valeur du vecteur des poids qui minimise une fonction de coût issue du principe de maximum de vraisemblance. Dans l'approche bayesienne, tous les paramètres, notamment les poids du réseau, sont considérés comme des variables aléatoires issues d'une distribution de probabilité. L'apprentissage d'un réseau de neurones consiste à déterminer la distribution de probabilité des poids connaissant les données d'apprentissage : on attribue aux poids une probabilité fixée *a priori*, et, une fois que les données d'apprentissage ont été observées, cette probabilité *a priori* est transformée en probabilité *a posteriori* grâce au théorème de Bayes.

Ainsi, si D représente l'ensemble des données d'apprentissage, $p(w)$ est la densité de probabilité *a priori* des poids, $p(D|w)$ la densité de probabilité d'observer les données

connaissant les poids du réseau, et $P(w|D)$ la probabilité *a posteriori* que l'on cherche à déterminer, alors le théorème de Bayes s'écrit :

$$P(w|D) = \frac{p(D|w)\ p(w)}{p(D)}$$

Pour un problème de classification, la probabilité d'observer les données connaissant les poids a été calculée au paragraphe 6.3.2.2 :

$$p(D|w) = \prod_{i \in D} (y_i(w))^{t_i}(1 - y_i(w))^{1-t_i} = \exp(-E_c(w))$$

Si l'on fait une hypothèse gaussienne pour la probabilité *a priori* des poids, elle s'écrit alors :

$$p(w) = \frac{1}{Z_w(\alpha)} \exp\left(-\frac{\alpha}{2}\sum_{i=1}^{p} w_i^2\right)$$

où $Z_w(\alpha)$ est une constante de normalisation qui ne dépend que de α :

$$Z_w(\alpha) = \left(\frac{2\pi}{\alpha}\right)^{\frac{1}{2}}$$

Comme les quantités $p(D)$ et $Z_w(\alpha)$ ne dépendent pas des poids du réseau, maximiser la probabilité *a posteriori* des poids du réseau revient à minimiser la quantité :

$$J(w) = E_c(w) + \frac{\alpha}{2}\sum_{i=1}^{p} w_i^2$$

On retrouve la fonction de coût avec un terme de *weight decay* introduite au paragraphe 6.5.2.2. Le terme de régularisation trouve, avec l'approche bayesienne, une interprétation naturelle, et la valeur de l'hyperparamètre α est liée à la variance de la probabilité *a priori* des poids.

Le formalisme décrit ici correspond à l'utilisation d'un seul hyperparamètre ; l'utilisation de plusieurs hyperparamètres correspond à des probabilités *a priori* différentes pour les différentes familles de poids. Comme ces probabilités sont indépendantes, la probabilité globale est le produit des probabilités, et, du fait des propriétés mathématiques de l'exponentielle, la nouvelle fonction de coût peut s'écrire, par exemple, pour trois hyperparamètres :

$$J(w) = E_c(w) + \frac{\alpha_1}{2}\sum_{w_1} w_i^2 + \frac{\alpha_2}{2}\sum_{w_2} w_i^2 + \frac{\alpha_3}{2}\sum_{w_3} w_i^2$$

Les hyperparamètres α_1, α_2, α_3 sont liés aux variances des distributions gaussiennes.

Il est possible de choisir d'autres formes pour la probabilité *a priori* des poids. [Buntine et Weigend, 1991] choisissent, par exemple, une probabilité *a priori* fondée sur l'entropie et aboutissent à une fonction de coût de la forme :

$$J(w) = E_c(w) + \alpha \sum_{i=1}^{p} \frac{w_i^2}{1 + w_i^2}$$

6.6.2 Les avantages de l'approche bayesienne

D'un point de vue théorique, [Neal, 1996] a montré que, lorsque les probabilités *a priori* des poids sont convenablement choisies, il n'est pas nécessaire de limiter la taille du réseau pour éviter le surajustement, et le nombre de neurones cachés peut tendre vers l'infini. Selon cette étude, le seul facteur qui doit limiter la taille du réseau est la capacité des ordinateurs utilisés et le temps disponible pour effectuer les calculs nécessaires.

D'un point de vue pratique, la théorie de l'approche bayesienne pour l'apprentissage des réseaux de neurones apporte d'importantes améliorations :

- Le concept de régularisation peut être interprété de façon naturelle dans le contexte bayesien.
- Les hyperparamètres intervenant dans la fonction de régularisation sont calculés lors de la phase d'apprentissage sans utiliser de base de validation. Le détail des calculs est précisé au paragraphe 6.6.5.
- Le calcul de l'évidence explicité au paragraphe 6.6.6 permet de sélectionner, parmi une famille de modèles, le meilleur modèle, uniquement grâce à la base d'apprentissage.
- Comme tous les calculs se font à partir de la base d'apprentissage, il n'est plus nécessaire de disposer d'une base de validation. Il est donc possible d'utiliser toutes les données dont on dispose pour estimer les poids du réseau.
- Des barres d'erreurs peuvent être calculées pour les problèmes de régression.
- L'incertitude sur les poids peut être prise en considération pour corriger la probabilité calculée par un réseau dans un problème de classification.

- Les entrées peuvent être sélectionnées grâce à la méthode *Automatic Relevance Determination* : un hyperparamètre est associé à chaque entrée et après l'apprentissage, les hyperparamètres avec de grandes valeurs indiquent des entrées non pertinentes.

6.6.3 Les inconvénients de l'approche bayesienne

Comme les paramètres utilisés sont maintenant issus de distributions de probabilité, il est nécessaire, pour connaître un paramètre, de calculer des intégrales faisant intervenir les distributions des autres paramètres. Il est, en général, impossible de calculer ces intégrales analytiquement, et plusieurs approches ont été proposées pour effectuer ces calculs. Mais soit ces méthodes sont très lourdes à implémenter, soit elles reposent sur des approximations qui peuvent fausser les résultats.

Finalement les résultats théoriques proposés par l'approche bayesienne sont souvent inapplicables en l'état dans le cadre des réseaux de neurones.

Dans ses travaux, Neal [Neal, 1992] utilise des méthodes de Monte Carlo couplées à des modèles de Markov cachés pour calculer les différentes intégrales intervenant dans les différentes étapes. Les calculs sont très lourds à mettre en place et nécessitent beaucoup de temps de calcul. Nous n'avons pas cherché dans ce travail à utiliser cette approche.

MacKay [MacKay, 1992a] [MacKay, 1992b] [MacKay, 1992c] a proposé des approximations reposant sur des hypothèses gaussiennes des probabilités *a posteriori*. Grâce à ces hypothèses, les calculs d'intégrales se trouvent simplifiés et peuvent être effectués plus ou moins simplement. Ces approximations sont parfois discutables, surtout pour les problèmes de classification. Néanmoins, grâce à ces approximations, les calculs sont simplifiés de sorte que l'approche bayesienne devient utilisable en pratique. L'approche proposée par MacKay est connue sous le nom de *evidence framework*.

6.6.4 Principe de l'approximation gaussienne

Dans son approche, MacKay considère une approximation gaussienne de la probabilité *a posteriori* des poids. Cette approximation est obtenue en effectuant un développement au second ordre de la fonction de coût *J(w)* autour de son minimum :

$$J(w) = J(w_{MP}) + \frac{1}{2}(w - w_{MP})^T A (w - w_{MP})$$

w_{MP} est la valeur la plus probable des poids et A est le hessien de la fonction de coût $J(w)$.

Avec cette approximation de la fonction J(w), la probabilité *a posteriori* des poids s'écrit :

$$p(w|D) = \frac{1}{Z_J}\exp\left(-J(w_{MP}) - \frac{1}{2}(w - w_{MP})^T A\,(w - w_{MP})\right)$$

Z_j est une constante de normalisation appropriée à l'approximation gaussienne, dont la valeur est obtenue en considérant le calcul de l'intégrale d'une gaussienne :

$$Z_J(\alpha) = \exp\left(-J(w_{MP})\right)(2\pi)^{\frac{p}{2}}\left(\det(A)\right)^{-\frac{1}{2}}$$

Grâce à ces approximations, les calculs d'intégrales sont simplifiés.

6.6.5 Calcul des hyperparamètres

Le traitement correct des hyperparamètres dans l'approche bayesienne implique le calcul des intégrales sur l'ensemble de leurs valeurs possibles. Par exemple, pour connaître la probabilité *a posteriori* des poids si le modèle comporte un hyperparamètre α, il faut calculer :

$$p(w|D) = \int p(w,\alpha|D)d\alpha$$

Deux approches différentes ont été proposées dans la littérature : la maximisation et l'intégration.

6.6.5.1 Calcul des hyperparamètres par le principe de maximisation

Le principe de la maximisation a été proposé par MacKay et repose sur les techniques développées par [Gull, 1988]. Si la densité de probabilité de l'hyperparamètre α est très étroite autour de sa valeur α_{MP} la plus probable, alors :

$$p(w|D) \approx p(w,\alpha_{MP}|D)\int p(\alpha)d\alpha = p(w,\alpha_{MP}|D)$$

Cette méthode consiste à calculer la valeur la plus probable pour les hyperparamètres, et à faire la suite des calculs avec ces valeurs pour les hyperparamètres.

La valeur la plus probable de l'hyperparamètre α est déterminée grâce au théorème de Bayes :

$$p(\alpha|D) = \frac{p(D|\alpha)\,p(\alpha)}{p(D)}$$

Donc, si la probabilité de l'hyperparamètre est uniforme, le maximum de la probabilité *a posteriori* de l'hyperparamètre est obtenu lorsque $p(D|\alpha)$ est maximum :

$$p(D|\alpha) = \int p(D|w)\,p(w|\alpha)\,dw$$

Dans le formalisme développé par MacKay, la quantité $p(D|\alpha)$ est appelée *evidence* de l'hyperparamètre.

La densité de probabilité *a priori* des poids $p(w)$ et la quantité $p(D|w)$ ont été calculées précédemment ; on a donc :

$$p(D|\alpha) = \frac{1}{Z_w(\alpha)} \int \exp\left(-J(w)\right) dw$$

soit :

$$p(D|\alpha) = \frac{Z_j(\alpha)}{Z_w(\alpha)}$$

En utilisant le résultat de l'approximation gaussienne de la probabilité *a posteriori,* indiqué au paragraphe 6.6.4, le logarithme de l'évidence de l'hyperparamètre s'écrit :

$$\ln p\left(D|\alpha\right) = -J\left(w_{MP}\right) - \frac{1}{2}\ln \det\left(A\right) + \frac{p}{2}\ln \alpha$$

Pour trouver le maximum de cette expression, afin de maximiser l'évidence de l'hyperparamètre, il faut donc dériver l'expression précédente par rapport à α.

Pour calculer la dérivée du déterminant de la matrice A, on écrit :

$$A = \nabla\nabla J(w) = \nabla\nabla E_c(w) + \alpha I = H + \alpha I$$

I est la matrice identité et H est le hessien de la fonction de coût non régularisée (l'entropie croisée), c'est donc une matrice de dimension (p, p). Si l'ensemble des valeurs propres de la matrice H est noté $\{\lambda_i\}$, alors la matrice A a pour valeurs propres l'ensemble $\{\alpha+\lambda_i\}$. Par conséquent :

$$\frac{\partial \ln\left(\det A\right)}{\partial \alpha} = \frac{\partial}{\partial \alpha} \ln\left(\prod_{i=1}^{p} \left(\lambda_i + \alpha\right)\right) = \frac{\partial}{\partial \alpha} \sum_{i=1}^{p} \ln\left(\lambda_i + \alpha\right) = \sum_{i=1}^{p} \frac{1}{\lambda_i + \alpha} = Tr\left(A^{-1}\right)$$

On obtient finalement :

$$\alpha_{MP} = \frac{\gamma}{\left\| w_{MP} \right\|^2}$$

$$\gamma = p - \alpha.\mathrm{Tr}(A^{-1}) = \sum_{i=1}^{p} \frac{\lambda_i}{\lambda_i + \alpha}$$

Une interprétation de ce calcul a été donnée par [Gull, 1989] : γ est le nombre de paramètres bien déterminés, c'est-à-dire le nombre de paramètres dont la valeur est effectivement déterminée par les données d'apprentissage plutôt que par les probabilités *a priori*.

Si plusieurs hyperparamètres interviennent dans la fonction de coût, le calcul de γ précédent n'est plus valable, il est nécessaire de calculer une valeur de γ_k différente pour chaque hyperparamètre α_k :

$$\gamma_k = \sum_j \left(\frac{\lambda_j}{\alpha_k + \lambda_j} \left(V^T I_k V \right)_{jj} \right)$$

et

$$\alpha_k = \frac{\gamma_k}{\sum_{w \in W_k} w^2}$$

où W_k représente le sous-groupe de poids liés à l'hyperparamètre α_k, $\{\lambda_j\}$ représente l'ensemble des valeurs propres de la matrice A, V est la matrice des vecteurs propres, et I_k est une matrice dont tous les termes sont nuls sauf les éléments diagonaux liés au groupe de poids gouvernés par l'hyperparamètre pour lesquels la valeur est 1.

Il est aisé de voir que dans le cas où il n'y a qu'un seul paramètre, on retrouve la formule précédente, puisqu'on a alors $\lambda_j = \lambda_j + \alpha$ et $V^T V = I$ car, la matrice A étant symétrique, la matrice V est orthogonale.

Par conséquent, contrairement au cas précédent, il faut également calculer les matrices de vecteurs propres, ce qui demande plus de temps que pour le calcul unique des valeurs propres [Press *et al.*, 1992].

La détermination des hyperparamètres par cette méthode nécessite donc plusieurs approximations, notamment l'approximation gaussienne de la probabilité *a posteriori*. De plus dans le calcul de la dérivée par rapport à α, les termes $\frac{\partial \lambda_i}{\partial \alpha}$ ont été négligés.

6.6.5.2 Calcul des hyperparamètres par le principe d'intégration

Une autre méthode du calcul des hyperparamètres a été appliquée par [Buntine et Weigend, 1991] et [Williams, 1995]. Elle consiste à calculer analytiquement l'intégrale qui fait intervenir les hyperparamètres. En faisant des hypothèses pour la probabilité *a priori* des hyperparamètres de la forme $p(\ln \alpha) = 1$, le calcul exact de l'intégrale devient alors possible. Les auteurs définissent des valeurs efficaces pour les hyperparamètres :

$$\alpha_{eff} = \frac{p}{\left\| w_{MP} \right\|^2}$$

La minimisation de la fonction de coût s'effectue en recalculant α à chaque itération de l'algorithme de minimisation. Cette méthode est également appelée *MAP* pour *Maximum a posteriori*.

6.6.5.3 Maximisation ou intégration

Il y a eu un grand débat dans la communauté pour savoir si l'intégration était préférable à la maximisation. *A priori* l'intégration semble être la bonne méthode puisqu'elle correspond à l'application de la théorie. Cependant, dans une publication plus récente, [MacKay, 1999] affirme la supériorité de la méthode de maximisation sur la méthode d'intégration.

Cependant, les deux expressions sont très proches et le deuxième résultat peut être considéré comme une approximation au premier ordre du premier (qualifiée de "cheap and cheerful" dans [MacKay, 1992a]) en prenant $\beta = p$.

6.6.5.4 Implémentation et convergence

L'hyperparamètre est initialisé à une valeur aléatoire, qui ne doit pas être trop grande afin que les poids ne tendent pas vers zéro dès les premières itérations. Ensuite, après un certain nombre d'itérations pour l'algorithme de minimisation, l'hyperparamètre est estimé à nouveau régulièrement selon l'une des deux formules suivantes :

- soit $\alpha_n = \frac{\gamma_n}{\sum_{i=1}^{p} w_i^2}$

  où la quantité γ_n représente un nombre de termes effectifs ; elle est calculée par :

  $$\gamma_n = p - \alpha_{n-1}.\mathrm{Tr}(A^{-1}),$$

  ce qui nécessite le calcule de la matrice A^{-1} ; le calcul de la trace est alors immédiat ;

- soit $\gamma = \sum_{i=1}^{p} \frac{\lambda_i}{\lambda_i + \alpha}$,

  ce qui nécessite le calcul des valeurs propres de la matrice *H*.

Dans tous les cas, il est nécessaire de calculer la matrice du hessien de la fonction de coût non régularisée (l'entropie croisée dans notre cas). Ce calcul peut être effectué par un algorithme inspiré de la rétropropagation [Bishop, 1992]. Il nécessite un nombre de calculs en $O(p^2)$ ce qui n'est pas prohibitif même pour des réseaux comprenant une centaine de poids. De plus, ces calculs n'interviennent que lors de l'apprentissage, et, comme il n'est plus utile, théoriquement, d'effectuer de validations croisées, le temps consacré au calcul des hyperparamètres est gagné par ailleurs.

Il est également possible d'utiliser des approximations de la fonction du hessien, mais dans les expériences qui suivent, le calcul exact a été utilisé. Les calculs d'inverse de matrice ou de valeurs propres ont été implémentés selon [Press *et al.*, 1992].

Lorsque les hyperparamètres sont estimés, la surface de coût est modifiée et une nouvelle minimisation est effectuée, jusqu'à ce que les hyperparamètres soient à nouveau modifiés.

Les deux étapes suivantes sont répétées un certain nombre de fois jusqu'à trouver des poids et des hyperparamètres qui n'évoluent plus :

1. minimisation partielle de la fonction de coût régularisée.
2. estimation des hyperparamètres.

La convergence de cet algorithme n'a pas, à notre connaissance, été démontrée pour les modèles non linéaires comme les réseaux de neurones. De plus, différents problèmes numériques peuvent se poser avec cette méthode pendant l'apprentissage : en effet, le minimum trouvé lors de la phase d'apprentissage est un minimum pour la fonction de coût régularisée, mais pas pour la fonction de coût non régularisée. La matrice *H* n'est donc pas nécessairement définie positive : certaines valeurs propres peuvent être négatives et entraîner une valeur de γ négative. Pour remédier à ce problème, une méthode *ad hoc* peut être utilisée : ne pas tenir compte des valeurs propres négatives comme il est suggéré dans la FAQ sur l'approche bayesienne[1].

[1] http://wol.ra.phy.cam.ac.uk/mackay/Bayes_FAQ.html

6.6.6 Sélection de modèles : calcul de l'évidence d'un modèle

6.6.6.1 Principe et calcul de l'évidence d'un modèle

D'après l'ensemble de la théorie, chaque modèle H_i trouvé est lié à une probabilité, le meilleur modèle étant celui pour lequel la probabilité est la plus grande connaissant les données. Un modèle est défini par son architecture, les valeurs de ses hyperparamètres et la distribution *a posteriori* de ses poids.

Il est donc nécessaire de calculer pour chaque modèle sa probabilité *a priori* ; or, toujours selon le théorème de Bayes :

$$P(H_i|D) = \frac{p(D|H_i)\, P(H_i)}{p(D)}$$

A priori, chaque modèle étant équiprobable, la quantité $P(H_i)$ est la même pour tous les modèles. De plus, comme le dénominateur ne dépend pas du modèle, seule la quantité $p(D|H_i)$ est déterminante. Cette quantité est appelée *évidence* du modèle et doit être calculée.

$$p(D|H_i) = \int p(D|w,H_i)\, p(w|H_i) dw$$

A partir des approximations déjà effectuées pour le calcul des hyperparamètres et d'autres approximations qui ne sont pas reprises ici, MacKay propose une formule pour calculer le logarithme de l'*évidence* :

$$\ln(p(D|H_i) = -J(w_{MP}) - \frac{1}{2}\ln(\det(A)) + \frac{p}{2}\ln(\alpha_{MP}) + \frac{1}{2}\ln\frac{2}{\gamma} + \ln N_c! + 2\ln N_c$$

Pour un modèle donné, plus cette quantité est grande, plus le modèle a une probabilité *a posteriori* élevée ; entre plusieurs modèles, il faut donc sélectionner celui qui possède la plus grande *évidence.*

6.6.6.2 Lien entre l'évidence et les performances en généralisation

Le modèle ainsi retenu est supposé être le meilleur modèle au sens de la théorie. Or, dans la pratique, le meilleur modèle est celui qui a les meilleures performances en généralisation. Il est donc important de vérifier si la notion d'évidence est corrélée à la performance en généralisation, et plus particulièrement si les modèles avec les plus grandes évidences ont les meilleures performances.

Selon Bishop [Bishop, 1995], il existe plusieurs raisons pour lesquelles cette corrélation est souvent mauvaise :

- La base de test étant de taille finie, l'estimation des performances sur cette base n'est pas précise, et dépend évidemment du choix de cette base.
- Il peut exister plusieurs modèles différents qui font exactement les mêmes prédictions et qui ont donc la même performance en généralisation ; cependant, le calcul de l'évidence va favoriser le modèle le plus simple.
- La performance est généralement calculée avec la valeur la plus probable des poids, or l'approche bayesienne nécessiterait de tenir compte de la distribution des poids.
- Le calcul de l'évidence présenté ici résulte d'approximations qui ne sont pas nécessairement justifiées.
- Numériquement, le calcul de l'évidence peut être instable, notamment le calcul du logarithme du déterminant de la matrice A.

Selon MacKay, une mauvaise corrélation entre ces deux mesures révèlent un modèle mal adapté c'est-à-dire un nombre d'hyperparamètres mal choisis. Ainsi, dans [MacKay, 1992b], il utilise un modèle avec un seul hyperparamètre et trouve une mauvaise corrélation. Lorsqu'il utilise trois hyperparamètres, la corrélation devient nettement meilleure.

[Thodberg, 1996] étudie également, sur un problème de régression, la corrélation entre ces deux valeurs. Cette corrélation est loin d'être parfaite, mais cependant, l'ensemble des modèles avec l'évidence la plus élevée ont bien les meilleures performances en généralisation.

6.7 Conclusion

Ce chapitre a permis de rappeler les propriétés principales des réseaux de neurones utilisés dans la suite de ce mémoire. La définition du surajustement a été rappelée afin de fixer précisément la nature du problème, et sa spécificité dans le cas de la classification.

Nous avons montré, sur un problème artificiel et sur un problème réel de filtrage, la nécessité d'ajouter un terme de *weight decay* à la fonction de coût usuelle pour les problèmes où le nombre d'exemples de la base d'apprentissage est limité. L'étude du cas réel a montré que les mauvaises performances obtenues lorsque la base d'apprentissage possédait peu d'exemples n'étaient pas uniquement dues au manque d'information, mais également à la façon de conduire

l'apprentissage. Sur cet exemple, l'ajout du terme de *weight decay* compense en grande partie le manque d'informations.
L'introduction de l'approche bayesienne propose un cadre théorique pour le terme de *weight decay* et résout, théoriquement, le problème de la détermination des hyperparamètres.

Enfin, l'intérêt de l'ajout d'un terme de *weight decay* a été prouvé sur problème particulier de filtrage ; les chapitre 7 et 8 vont permettre de tester cette approche, ainsi que les formules de calcul des hyperparamètres, sur un ensemble de thèmes différents.