

trouvée, \vec{w}^* qui fait un nombre d'erreurs e^* . Pour cela on applique l'algorithme standard du perceptron. A chaque instant t , les poids $\vec{w}(t)$ produisent un certain nombre d'erreurs $e(t)$. Si $e(t) < e^*$, alors on remplace les poids de la poche, $\vec{w}^* \leftarrow \vec{w}(t)$.

Au bout d'un certain temps t_{max} on arrête l'algorithme en prenant comme solution non pas $\vec{w}(t)$ mais les poids \vec{w}^* qui sont dans la *poche*. Si on considère que le nombre d'exemples P est fini, alors en utilisant cet algorithme pendant un temps *suffisamment long*, on peut espérer trouver les poids qui minimisent le nombre d'erreurs avec une grande probabilité. La solution trouvée n'est pas garantie d'être optimale en termes de la distance des exemples à l'hyperplan séparateur, c'est-à-dire, au sens de la stabilité (2.12). En raison de sa simplicité, cet algorithme est souvent utilisé dans les méthodes constructives présentées au Chapitre 3.

4.2 Minimerror-L

Le but essentiel de l'apprentissage est de trouver des poids qui minimisent l'erreur de généralisation ε_g . Cependant, en ne disposant que d'un ensemble d'apprentissage \mathcal{L}^α fini, on minimise le nombre d'erreurs ε_t sur les exemples de cet ensemble, en espérant que les poids trouvés minimisent aussi ε_g . Ceci peut se poser formellement comme la minimisation de la fonction de coût suivante :

$$E(\vec{w}) = \sum_{\mu=1}^P \Theta(-\gamma^\mu(\vec{w})) \quad (4.1)$$

où P est le nombre d'exemples, γ^μ est la stabilité définie par (2.12) de l'exemple μ , et Θ est la fonction de Heaviside (équation 2.8). Cette fonction compte simplement le nombre d'erreurs (ou sorties incorrectes) produits par les poids \vec{w} . La difficulté pour minimiser cette fonction est qu'elle n'est pas dérivable.

L'algorithme du perceptron [85] est capable de trouver une solution qui minimise (4.1) seulement si l'ensemble d'apprentissage est linéairement séparable, *i.e.* si une solution existe. Il y a d'autres algorithmes qui permettent de trouver des solutions optimales en termes de la stabilité [55, 86], et de la généralisation [14, 15] si l'ensemble est linéairement séparable ; mais si l'ensemble d'apprentissage ne l'est pas, ils ne s'arrêtent pas. Certains algorithmes [32, 36] détectent l'absence d'une solution sans erreurs d'apprentissage, et permettent de trouver des poids *raisonnables*, mais ils ne peuvent pas assurer que les poids minimisent le nombre d'erreurs.

Le but fondamental d'un bon algorithme d'apprentissage consiste à trouver le vecteur des poids \vec{w} qui minimise l'erreur d'apprentissage, et qui maximise les stabilités, pour avoir un apprentissage robuste. Pour cela il n'est pas suffisant de minimiser le nombre d'erreurs (le nombre d'exemples dont les stabilités $\gamma \leq 0$). Le

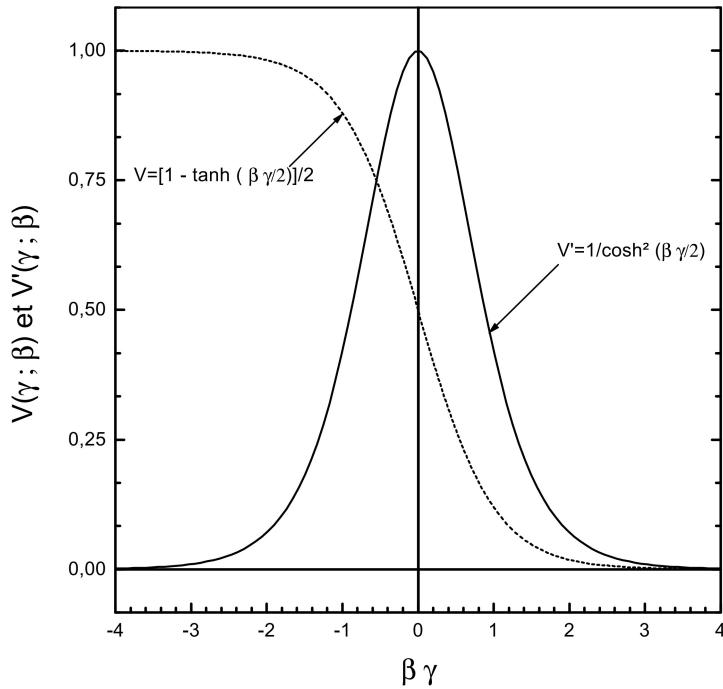


Figure 4.1: La fonction de coût $V(\gamma ; \beta) = \frac{1}{2}[1 - \tanh(\beta\gamma/2)]$ et sa dérivée.

coût (4.1) compte de la même façon tous les exemples non appris, quelles que soient leurs stabilités. Pour pouvoir extraire l'information des exemples pertinents, qui constituent les frontières entre classes, il faut modifier la fonction de coût.

L'algorithme Minimerror-L [45, 39] minimise la fonction de coût suivante :

$$E(\vec{w} ; \beta) = \sum_{\mu=1}^P V(\gamma^\mu ; \beta) \quad (4.2)$$

où :

$$V(\gamma ; \beta) = \frac{1}{2}(1 - \tanh \frac{\beta\gamma}{2}) \quad (4.3)$$

représente la contribution d'un exemple avec stabilité γ à la fonction de coût. V dépend d'un paramètre β (pour des raisons qui apparaîtront plus loin, on introduit $T = 1/\beta$, appelé *température*).

Dans la limite où $T \rightarrow 0$ ($\beta \rightarrow \infty$), on a :

$$V(\gamma ; \infty) = \begin{cases} 0 & \text{si } \gamma > 0 \\ 1 & \text{si } \gamma < 0 \end{cases} \quad (4.4)$$

Dans cette limite, la fonction $E(\vec{w} ; \infty)$ (4.2) est égale au coût (4.1) : elle compte *strictement* le nombre d'erreurs commises sur l'ensemble d'apprentissage. Dans la limite contraire, quand $T \rightarrow \infty$ ($\beta \rightarrow 0$) chaque exemple contribue au coût proportionnellement à sa stabilité.

A température T finie, les exemples avec une grande stabilité positive $\gamma \gg 0$ ont $V \approx 0$ et ceux avec $\gamma \ll 0$ ont $V \approx 1$. Donc, pour les exemples qui se trouvent loin de l'hyperplan séparateur, $E(\vec{w} ; \beta)$ compte le nombre de fautes ; mais les exemples dont les stabilités se trouvent *dans* une fenêtre de largeur $\approx 2T$ des deux cotés de l'hyperplan séparateur ($-2/\beta < \gamma < 2/\beta$), contribuent à la fonction de coût proportionnellement à $1 - \beta\gamma/2$: même les exemples bien appris contribuent au coût (4.2). Gordon et Grempel [42] ont montré que si l'ensemble \mathcal{L}^α est non linéairement séparable, si β est suffisamment grand (strictement dans la limite $\beta \rightarrow \infty$) les poids minimisent la fonction de coût (4.2) minimisant le nombre d'erreurs d'apprentissage. Par contre, si l'ensemble d'apprentissage est linéairement séparable, il existe une valeur optimale de β , telle que les poids trouvés avec Minimerror-L généralisent avec un erreur ε_g numériquement indiscernable de la valeur minimale, qui correspond au perceptron Bayesien [81].

A température T finie, (4.2) est dérivable et l'on peut chercher son minimum par une descente en gradient, ce que l'on ne peut pas faire avec le coût (4.1). Les poids sont modifiés itérativement :

$$\vec{w}(t+1) \leftarrow \vec{w}(t) + \delta\vec{w}(t) \quad (4.5)$$

$$\delta\vec{w}(t) = -\varepsilon \frac{\partial E(\vec{w} ; \beta)}{\partial \vec{w}} \quad (4.6)$$

L'équation (4.6) peut s'écrire comme d'autres algorithmes d'apprentissage du type itératif Hebbien [81] :

$$\delta\vec{w}(t) \propto \sum_{\mu=1}^P c^\mu(t) \tau^\mu \vec{\xi}^\mu \quad (4.7)$$

où le coefficient c^μ dépend de l'algorithme¹. Minimerror-L a comme coefficient :

$$c^\mu \propto \frac{1}{\cosh^2(\beta\gamma^\mu/2)} \quad (4.8)$$

¹Par exemple, le *Perceptron de Stabilité Maximale* (MSP) [41] a $c^\mu = \Theta(\kappa - \gamma^\mu)$, où κ est la stabilité imposée à l'exemple le moins stable. Pour la règle de Widrow-Hoff, $c^\mu = 1 - \gamma^\mu$.

qui a son maximum à $\gamma = 0$ et décroît exponentiellement pour des $|\gamma| \gg 2T$. Ceci signifie que la contribution la plus importante à la modification des poids provient des exemples situés dans une fenêtre de largeur $\approx 2T$ des deux cotés de l'hyperplan séparateur, avec des stabilités positives ou négatives. Les exemples se trouvant loin en-dehors de cette fenêtre auront des coefficients exponentiellement petits.

La descente peut être faite par la méthode de gradient simple (4.5) ou de gradient conjugué [75]). Raffin et Gordon [81] ont utilisé cette dernière méthode pour trouver le minimum de (4.2) sur des ensembles linéairement séparables à entrées binaires. Nous avons fait plusieurs tests sur des ensembles non linéairement séparables en utilisant les deux méthodes et nous avons trouvé que la méthode du gradient simple proposée par Gordon et Berchier [39] a des performances supérieures à celle du gradient conjugué, comme on le montre plus loin sur des exemples.

Dans les applications, la valeur de β qui donne les meilleurs résultats est inconnue, car elle dépend de l'ensemble d'apprentissage. L'idée intuitive la plus importante de l'algorithme consiste à l'ajuster en cours d'apprentissage, en combinant un *recuit déterministe* avec la descente en gradient [39] : à chaque itération² on décroît T (d'où le nom de recuit). La variation de T implique une modification du coût, ce qui amène à considérer à chaque pas un coût *different*, contrôlé par la température. Cette procédure permet d'utiliser l'information des exemples qui sont de plus en plus *proches* de l'hyperplan pour le positionner.

Afin de trouver le minimum du coût, on cherche d'abord la direction de la descente en gradient à haute température T_0 [39]. Puisque tous les exemples contribuent à l'apprentissage avec la même "force", ceci revient à utiliser la règle de Hebb. Au fur et à mesure des itérations, on décroît T : la fenêtre devient de plus en plus étroite. Quand la température est suffisamment basse, le nombre d'exemples dans la fenêtre de largeur $2T$ est négligeable et l'apprentissage s'arrête.

A partir de (4.2) et (4.3) on aura :

$$\frac{\partial E(\vec{w} ; \beta)}{\partial w_i} = \sum_{\mu} \frac{\partial V(\gamma^{\mu} ; \beta)}{\partial w_i} \quad (4.9)$$

Où chaque terme de la somme est :

$$\frac{\partial V(\gamma^{\mu} ; \beta)}{\partial w_i} = -\frac{\beta}{4\|\vec{w}\|} \frac{\xi_i^{\mu} \tau^{\mu}}{\cosh^2(\frac{\beta\gamma^{\mu}}{2})} \left\{ \frac{\xi^{\mu} \tau^{\mu}}{\|\vec{w}\|} - \gamma^{\mu} \frac{w_i}{\|\vec{w}\|^2} \right\} \quad (4.10)$$

Mais l'implantation de Minimerror (Gordon et Berchier [39]) utilise seulement :

²Une itération dans notre contexte représente le passage de tout l'ensemble d'apprentissage qui fournit l'information pour trouver la bonne position de l'hyperplan séparateur

$$\frac{\partial V(\gamma^\mu ; \beta)}{\partial w_i} = -\frac{\beta}{4} \frac{\xi_i^\mu \tau^\mu}{\cosh^2(\frac{\beta \gamma^\mu}{2})} \quad (4.11)$$

où le préfacteur $\varepsilon\beta/4$ est remplacé par un seul facteur ϵ :

$$\delta \vec{w}(t) = -\epsilon \frac{\xi_i^\mu \tau^\mu}{\cosh^2(\frac{\beta \gamma^\mu}{2})} \quad (4.12)$$

et le dernier terme de (4.10) est remplacé par une normalisation des poids :

$$\vec{w}(t+1) \leftarrow \frac{\vec{w}(t+1)}{\|\vec{w}(t+1)\|} \quad (4.13)$$

Le choix de la température a une influence directe sur l'aspect du "paysage" défini par la fonction $E(\vec{w} ; \beta)$: à haute température T on aura un paysage de "collines" douces avec des variations d'amplitude peu importantes entre un sommet et une vallée. Par contre, à basses températures, le paysage aura des formes de paliers avec des variations d'amplitude très fortes entre configurations voisines.

Quand le critère d'arrêt a été satisfait (voir 4.2.2), on fait une dernière minimisation par gradient conjugué, ce qui permet de trouver le minimum du coût. Ce dernier pas doit être fait avec une fonction de coût légèrement différente [81] :

$$E^*(\vec{w} ; \beta) = \sum_{\mu=1}^P V(\gamma^\mu ; \beta) + (\sqrt{N+1} - \|\vec{w}\|)^2 \quad (4.14)$$

car, n'ayant pas le droit de normaliser les poids \vec{w} dans la méthode du gradient conjugué, le dernier terme de (4.14) contraint la recherche des poids \vec{w} dans l'hypersphère de rayon $\sqrt{N+1}$.

L'algorithme modifie itérativement les poids suivant (4.5) et (4.6) en utilisant pour chaque pas une température différente. En effet, après avoir modifié tous les poids, la valeur de β est augmentée suivant :

$$\beta(t+1) = \beta(t) + \delta\beta(t) \quad (4.15)$$

où $\delta\beta(t)$ est petit ³ pour que la fonction de coût soit peu modifiée.

Puisque $T = 1/\beta$, la nouvelle température est :

$$T(t+1) = \frac{T(t)}{1 + T(t) \cdot \delta\beta(t)} \simeq T(t) - T^2(t)\delta\beta(t) \quad (4.16)$$

³En pratique, $\delta\beta(t)$ est de l'ordre de 10^{-3} .

La température décroît de plus en plus lentement lors des itérations successives. Ce procédé de recherche du minimum à des températures décroissantes est appelé *recuit déterministe*.

Empiriquement, Gordon et Berchier [39] ont trouvé que l'on peut accélérer la convergence si $\delta\beta$ est adapté dynamiquement, de manière à introduire des pas plus grands lorsque le nombre d'erreurs d'apprentissage ne varie pas. Pour cela, on compte le nombre d'itérations t_s successives pendant lesquelles le nombre d'exemples mal classés n'est pas modifié, et $\delta\beta(t)$ est donné par :

$$\delta\beta(t+1) = \delta\beta_0 \log(1 + t_s) \quad (4.17)$$

où $\delta\beta_0$ est une constante. Si à l'itération t le nombre d'erreurs d'apprentissage a diminué, on remet le compteur à zéro $t_s(t) = 0$ et $\delta\beta(t+1) = 0$: l'itération suivante se fait à la même température. Par contre, si $t_s(t) \neq 0$, la température est diminuée avec des pas qui augmentent logarithmiquement avec le temps écoulé depuis la dernière modification du nombre d'erreurs.

Le pas de l'algorithme ϵ , souvent appelé *taux d'apprentissage*, est aussi adaptable. Il est fonction de l'itération t et de β . On commence avec un $\epsilon = \epsilon_0$ petit, de l'ordre de 10^{-2} . On vérifie la valeur de ϵ toutes les t_0 itérations : si le nombre d'erreurs à l'instant t n'a pas changé, on ne modifie pas ϵ . Par contre, si ce nombre a changé, on réduit ϵ en le multipliant par un facteur q entre 0 et 1. Empiriquement, nous avons choisi comme facteur $q = 0.8$, de sorte que :

$$\epsilon(t+1) = q\epsilon(t) \quad (4.18)$$

permet de chercher de plus en plus finement le minimum.