Aspects théoriques de l'optimisation d'une règle d'apprentissage

Samy Bengio Yoshua Bengio* Jocelyn Cloutier Jan Gecsei

Université de Montréal, Département IRO
Case Postale 6128, Succ. "A",
Montréal, QC, Canada, H3C 3J7
tel: (514) 343-6111 ext. 3545, fax: (514) 343-5834
e-mail: bengio@iro.umontreal.ca

Résumé

Ayant exposé dans de précédentes publications (voir [Beng90, Beng92] notamment) l'idée que l'on pouvait optimiser des règles d'apprentissage paramétriques pour réseaux de neurones, nous montrons dans cet article comment développer, par la méthode du Lagrangien, le gradient nécessaire à l'optimisation d'une règle d'apprentissage par descente du gradient. Nous présentons aussi les bases théoriques qui permettent d'étudier la généralisation à de nouvelles tâches d'une règle d'apprentissage dont les paramètres ont été estimés à partir d'un certain ensemble de tâches. Enfin, nous exposons brièvement les résultats d'une expérience consistant à trouver, par descente du gradient, une règle d'apprentissage pouvant résoudre plusieurs tâches booléennes linéairement et non linéairement séparables.

Abstract

In previous work ([Beng90, Beng92]) we discussed the subject of parametric learning rules for neural networks. In this article, we develop a method to calculate the gradient needed for the optimization of the learning rule. The method is based on the Lagrangian formalism. We also present a theoretical basis permitting to study the generalization of a learning rule whose parameters are estimated from a set of learning tasks. Finally, we present briefly the results of an experiment consisting in finding a learning rule for a number of linearly as well as nonlinearly separable boolean problems.

Mots-clés: apprentissage, optimisation, descente du gradient, capacité, généralisation.
Keywords: learning, optimization, gradient descent, capacity, generalization.

*Yoshua Bengio est actuellement chez AT&T Bell Laboratories, room 4G305, Holmdel NJ 07733

1 Introduction

Le moteur principal des réseaux de neurones réside dans leur capacité de s'adapter à leur environnement, et ce grâce à des *règles d'apprentissage synaptique* qui dictent les variations des paramètres (poids des connexions) de ces réseaux.

Nous avons proposé dans [Beng90] une méthode permettant de chercher de nouvelles règles d'apprentissage en utilisant des outils d'optimisation. Cette méthode considère une règle d'apprentissage comme une fonction paramétrique, dont les entrées sont locales, et qui serait la même pour la plupart des neurones. Cette méthode est décrite à la section 2.

Nous montrons de plus dans cette section comment on peut définir la capacité d'une règle d'apprentissage parametrisée. Ceci permet d'étudier la généralisation à de nouvelles tâches d'une règle d'apprentissage dont les paramètres ont été estimés à partir d'un certain ensemble de tâches.

La section 3 consiste à dériver les formules nécessaires pour effectuer une descente du gradient sur les paramètres afin d'optimiser une règle d'apprentissage. Pour ce faire, nous utilisons le formalisme du Lagrangien, qui fut aussi utilisé par Le Cun ([LeCu88]) pour étudier l'algorithme de la rétropropagation de l'erreur [Rume86].

Pour finir, nous décrivons à la section 4 une expérience ayant permis de trouver, grâce à cette méthode, une règle d'apprentissage pour résoudre des tâches booléennes linéairement et non linéairement séparables.

2 Optimisation de la règle d'apprentissage

Dans cette section, nous nous proposons de faire un bref rappel de l'idée d'optimiser des règles d'apprentissage. On pourra retrouver de plus amples explications dans [Beng90, Beng91a, Beng91b, Beng92]. Tout d'abord, nous considérons la règle d'apprentissage comme une fonction paramétrique, et nous proposons d'optimiser les paramètres de cette fonction à l'aide d'un algorithme d'optimisation, par exemple la descente du gradient. Pour ce faire nous faisons les hypothèses suivantes:

- la même règle est utilisée dans plusieurs neurones (cette contrainte peut être étendue à une règle pour chaque type de neurone ou de synapse[1]).
- il existe une dépendance (éventuellement stochastique) entre la modification synaptique et certaines informations disponibles localement près de la synapse (en d'autres termes, la modification synaptique n'est pas totalement le fruit du hasard).
- cette dépendance peut être approximée par une fonction paramétrique, c'est-à-dire une fonction qui s'exprime sous la forme

$$f(x_1, x_2, ...x_n; \theta) \tag{1}$$

  où les x_i sont les variables de la fonction et θ est un ensemble de paramètres.

[1] On sait maintenant qu'il existe dans le cerveau différentes sortes de neurones et de synapses, bien qu'on ne connaisse pas encore très bien leur fonctionnement [Gard87].

2.1 Variables et paramètres de la règle d'apprentissage

L'espace des algorithmes d'apprentissage étant très large, nous proposons de le limiter en s'inspirant de mécanismes synaptiques connus pour nous aider à concevoir la forme de la règle que nous cherchons. Ainsi, les seules variables d'entrée (les x_i de l'equation (1)) que nous considérons doivent être locales à la synapse, telles l'activité du neurone pré-synaptique, l'activité du neurone post-synaptique, le poids de la synapse, l'activité d'un neurone facilitateur, (on dit aussi neurone modulateur car ils influencent, *modulent*, directement une synapse) ou encore la concentration d'un modulateur chimique actif (voir figure 1). Notons que le fait de contraindre la règle d'apprentissage à être biologiquement plausible ne doit pas être perçu comme une contrainte artificielle mais plutôt comme une façon de restreindre l'espace à analyser à un sous-espace où l'on sait qu'il existe au moins une bonne solution (celle utilisée par notre cerveau). Nous supposons que ces contraintes devraient faciliter la recherche d'une nouvelle règle d'apprentissage. Il faut cependant se garder d'oublier que les modèles que nous utilisons sont très simplifiés et qu'il existe encore de grandes lacunes dans nos connaissance du cerveau, en particulier en ce qui concerne l'apprentissage.

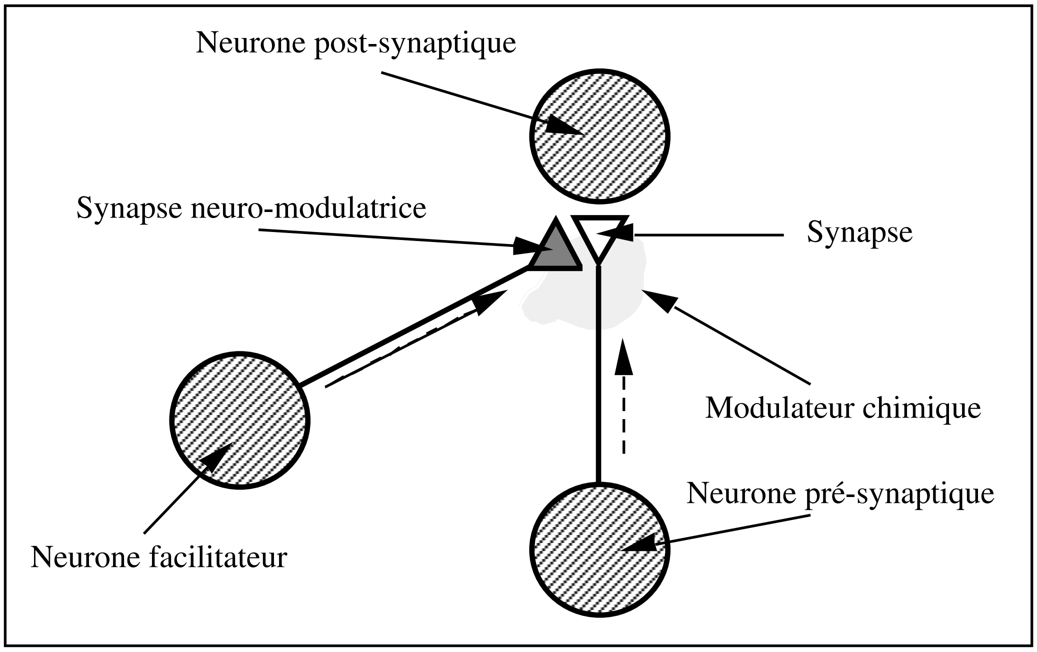

Figure 1: Divers éléments se trouvant aux alentours d'une synapse peuvent influencer son efficacité.

Si l'on considère $w(i,j)$ comme le poids d'une connexion synaptique, alors l'équation (2) donne la formule générale du changement de poids[2]:

$$\Delta w(i,j) = \Delta w(\text{variables locales}; \theta) \tag{2}$$

La modification synaptique $\Delta w(i,j)$ de la connexion entre le neurone i et le neurone j est calculée à l'aide de la fonction $\Delta w()$ selon les variables disponibles à cette synapse (les variables locales) et θ, un ensemble de paramètres. C'est justement ces paramètres que nous proposons d'optimiser afin d'améliorer la règle d'apprentissage. Dans cet article, nous utiliserons la règle paramétrique suivante:

$$\Delta w(i,j) = \theta_0 + \theta_1 y(i) + \theta_2 x(j) + \theta_3 y(mod(j)) + \theta_4 y(j) y(mod(j)) + \theta_5 y(i) x(j) + \theta_6 y(i) w(i,j) \tag{3}$$

[2]Cette discussion peut être aisément généralisée au cas où l'on admet plusieurs synapses entre deux neurones.

où $w(i,j)$ représente la connexion entre les neurones i et j, $x(j)$ représente le potentiel d'activation du neurone j (ou potentiel post-synaptique), $y(i)$ représente la sortie du neurone i (pré-synaptique), et $y(mod(j))$ représente la sortie du neurone modulant le neurone j.

Nous avons donc utilisé des connaissances biologiques pour le *design* de la règle d'apprentissage. Plus spécifiquement, nous y avons intégré les modules suivants:

- $y(i) \times y(mod(j))$: ce terme est utilisé dans les modèles de [Hawk89a, Hawk83].
- $y(i) \times x(j)$: ce terme représente le mécanisme Hebbien [Hebb49].
- $y(i) \times w(i,j)$: ce terme correspond à un processus d'oubli, utilisé notamment dans le modèle de [Gluc87].

2.2 Apprentissage et optimisation

Dans un système d'apprentissage, le but est de découvrir, d'apprendre, la relation existant entre deux variables aléatoires X et Y dont les distributions de probabilité ne sont pas connues. En général, on se restreint au cas où Y est une fonction de X, ou bien on cherche seulement à obtenir l'espérance de Y étant donné X: $E[Y \mid X]$. Soit un réseau de neurones calculant la fonction suivante

$$\hat{y} = f(x) \tag{4}$$

où $f(\cdot)$ est une fonction définie par la structure (fixe) et les poids (variables) du réseau, x est une instance de X et $\hat{y}$ est la réponse du réseau sur présentation de x. Soit $y \in Y$ la valeur qu'on veut associer à $x \in X$, alors l'apprentissage dans un réseau de neurones est un problème d'optimisation qui consiste à minimiser une mesure de la différence entre y et $\hat{y}$ pour tous les couples $(x, y) \in (X, Y)$. La fonction de coût la plus utilisée est le critère des moindres carrés, c'est-à-dire

$$C = \frac{1}{2} \sum_{y \in Y} (y - \hat{y})^2 \tag{5}$$

Cependant, pour pouvoir minimiser parfaitement C, il faut non seulement avoir une structure de réseau ayant un pouvoir d'expression suffisamment grand pour englober la solution, mais de plus, il faut avoir accès à toutes les instances $(x, y) \in (X, Y)$, ce qui est en général impraticable. On se contente donc de minimiser C sur un sous-ensemble (X^*, Y^*) en espérant que ce sous-ensemble soit assez grand et représentatif pour que le réseau de neurones généralise, c'est-à-dire qu'il réponde adéquatement à des exemples x tels que $x \in X$ mais $x \notin X^*$. Néanmoins, des études théoriques telles que celles de Baum et Haussler [Baum89] ou Vapnik [Vapn82, Vapn71] permettent de donner une borne supérieure sur le nombre d'exemples nécessaires pour généraliser avec une erreur maximale donnée. Ces théorèmes permettent d'exprimer l'erreur espérée (dans le pire des cas) sur de nouveaux exemples en fonction de la capacité du réseau (par exemple la VC-dimension).

2.3 Capacité d'une règle d'apprentissage

Dans cet article nous discutons de la possibilité "d'entraîner" une règle d'apprentissage (telle que celle de l'équation (3)), c'est-à-dire d'optimiser ses paramètres (dans l'équation (3), les paramètres sont les θ_i) de telle manière à ce qu'un réseau dont les poids sont adaptés à l'aide de cette règle aie la meilleure performance possible sur un certain nombre de tâches. Pour que cette règle soit utile, il faudrait qu'on puisse ensuite l'utiliser pour entraîner des réseaux sur de nouvelles tâches. Cela pose la question de la généralisation d'une règle d'apprentissage. Afin de pouvoir étudier la généralisation d'une règle d'apprentissage, il nous faut définir une mesure de *capacité* pour une règle d'apprentissage.

2.3.1 Capacité d'une famille de fonctions d'un ensemble quelconque vers R

Avant de définir la capacité d'une règle d'apprentissage, il nous faut définir une mesure de capacité pour des fonctions d'un ensemble arbitraire vers l'ensemble des réels R. Cette définition est une extention de la définition de la dimension Vapnik-Chervonenkis (dénotée VCdim) (voir [Vapn82], [Baum89]) similaire à celle implicite dans [Vapn82].

Soit F une famille paramétrisée de fonctions de l'ensemble X à l'ensemble des réels R, avec les paramètres θ:

$$F(\theta) : X \rightarrow R \tag{6}$$

Pour chaque $f = F(\theta) \in F$ on dénote par $I(f)$ la fonction de seuil vers $\{0, 1\}$ définie ainsi:

$$I(f)(x; \theta, \beta) = \begin{cases} 1 & \text{si } f(x; \theta) \geq \beta \\ 0 & sinon \end{cases} \tag{7}$$

On peut aussi définir pour l'ensemble F

$$I(F) = \{I(f) : f \in F\} \tag{8}$$

Dans le cas où $f(x)$ est discret, VCdim(F) = VCdim$(I(F))$. Dans les autres cas, on définit VCdim(F) = VCdim(I(F)), où β est choisi pour le pire cas (qui maximise la VCdim(F)).

2.3.2 Définition du critère d'optimisation

Dans le cas qui nous intéresse, X est un ensemble de fonctions. Pour optimiser une règle d'apprentissage, on choisit un certain nombre d'éléments $x \in X$. On échantillonne ensuite chacune de ces fonctions pour obtenir pour chacune un ensemble d'exemples d'apprentissage. Chacun de ces ensembles d'apprentissage est utilisé pour entraîner en un nombre fixe d'itérations le réseau (dont l'architecture est fixe).

Définissons d'abord un coût local $J(z, x; \theta)$ qui dépend de l'échantillonage de x, à savoir un ensemble d'apprentissage que l'on dénotera z. $J(z, x; \theta)$ est l'erreur obtenue sur l'échantillon z de la fonction $x \in X$ en entraînant le réseau pendant un nombre fixe d'itérations avec les paramètres θ de la règle d'apprentissage.

Notre fonction de X vers R peut alors être définie ainsi:

$$f(x;\theta) = E_z[J(z,x;\theta)] \tag{9}$$

$f(x;\theta)$ est donc l'espérance sur les ensembles d'apprentissage z (i.e. des échantillons de x) de la performance du réseau après une nombre fixe d'iterations. Cela nous donne une famille de fonctions

$$F : X \to R \tag{10}$$

dont la capacité est définie comme dans la sous-section précedente par VCdim(F).

On peut généraliser cet argument au cas où le réseau est en fait la concaténation d'un ensemble de sous-réseaux, chacun effectuant une tâche différente. Dans ce cas, le coût J peut être défini par exemple comme la somme des erreurs sur tous les réseaux (après apprentissage).

$$J = \sum_i J_i \tag{11}$$

Grâce au développement qui précède on peut parler de la capacité d'une règle d'apprentissage et utiliser tous les résultats généraux sur la VC-dimension pour étudier la généralisation d'une règle d'apprentissage. Par exemple, il devient clair que la performance espérée de notre règle sur des nouvelles tâches augmentera avec le nombre de fonctions utilisées pour estimer θ, mais pourrait diminuer avec le nombre de paramètres θ si l'on a pas suffisamment d'exemples de fonctions pour estimer θ. Cela justifie aussi l'utilisation de connaissances *à-priori* pour construire la forme de la règle d'apprentissage. On comprendra mieux aussi que notre règle aura des chances de généraliser sur des fonctions similaires à celles utilisées pour estimer θ (c'est-à-dire tirées d'un même ensemble de fonctions X). Il nous faudra donc choisir pour estimer θ des tâches diverses et représentatives de celles sur lesquelles on voudra ensuite appliquer la règle d'apprentissage.

3 Utilisation de la méthode du Lagrangien

Dans cette section, nous présentons une méthode basée sur le formalisme du Lagrangien pour dériver les formules nécessaires pour effectuer une descente du gradient sur les paramètres d'une règle d'apprentissage.

Le Cun, dans [LeCu88], utilise aussi la méthode du Lagrangien pour étudier les mécanismes de l'algorithme de la rétropropagation de l'erreur ([Rume86]), qui permet d'effectuer une descente du gradient à travers un réseau de neurones.

Le Lagrangien est la somme d'une fonction objective et d'un certain nombre de termes exprimant les contraintes du système, chacun multiplié par une variable, appelée *multiplicateur de Lagrange*. Il suffit alors de dériver cette fonction par rapport à toutes les variables du système et de forcer ces dérivées à zéro pour obtenir un optimum (possiblement local) de la fonction objective, ou, lorsqu'il ne se déduit pas directement, le gradient permettant de s'en approcher.

3.1 Notation

Considérons un réseau de neurones ayant des neurones modulateurs (tels que définis à la section précédente), composé de N neurones et d'une connectivité quelconque. Considérons de plus que ce réseau se verra présenté I vecteurs d'entrée successifs pour lesquels il existe un vecteur de sortie désirée d.

Dénotons par $x_p(k)$ le potentiel d'activation du neurone k à la présentation du vecteur d'entrée p. De la même façon, $y_p(k)$ représente la sortie du neurone k, $w_p(j,k)$ le poids de la connexion entre le neurone j et le neurone k, et $d_p(k)$ la valeur désirée du neurone de sortie k. Les neurones modulateurs sont dénotés de la façon suivante: $mod(k)$ représente le neurone qui module toutes les connexions arrivant au neurone k, et si $l = mod(k)$, alors $k = inv(l)$. Enfin, $source(k)$ représente l'ensemble des neurones connectés au neurone k et $dest(k)$ l'ensemble des neurones où k se connecte.

3.2 Formulation du Problème

Les équations de base régissant l'apprentissage dans un réseau de neurones avec règle d'apprentissage paramétrique et neurones modulateurs sont les suivantes: tout d'abord, la règle d'activation dicte comment calculer le potentiel d'activation d'un neurone en fonction de celui des autres neurones.

$$x_p(k) = \sum_{j \in source(k)} w_p(j,k) \cdot y_p(j) \tag{12}$$

Ensuite, une fonction de sortie permet de calculer la sortie d'un neurone en fonction de son potentiel d'activation.

$$y_p(k) = F(x_p(k)) \tag{13}$$

où $F()$ est une fonction continue (on utilise souvent une fonction sigmoïde).

Enfin, une règle d'apprentissage dicte la variation des poids dans le temps. Dans notre cas, cette règle est remplacée par une fonction paramétrique comme celle de l'équation (3).

$$\begin{aligned} w_{p+1}(j,k) &= w_p(j,k) + \\ & \quad r \sum_i \theta_i M_{i,p}(j,k,x_p(k), w_p(j,k), y_p(mod(k)), y_p(j)) \end{aligned} \tag{14}$$

où r est le taux d'apprentissage, M_i est une fonction de plusieurs valeurs locales (par exemple $w_p(j,k)$, $y_p(j)$, etc), et θ_i est le paramètre contrôlant l'influence du module M_i (par exemple, $M_i = y(j) \cdot x(k)$ est le module exprimant la règle de Hebb).

Et comme dans tout système d'apprentissage supervisé, le réseau doit évoluer de façon à minimiser un coût donné, dans notre cas, les moindres carrés.

$$\sum_{k \in sortie} C(y_p(k)) = \frac{1}{2} \sum_{k \in sortie} (d_p(k) - y_p(k))^2 \tag{15}$$

3.3 Dérivation du gradient

On peut alors écrire le Lagrangien de ce système d'équations comme suit:

$$
\begin{aligned}
& L(x, y, w, \theta, \alpha, \beta, \gamma) = \\
& \quad \sum_{p=1}^{I} \sum_{k \in sortie} C(y_p(k)) \\
& \quad + \sum_{p=1}^{I} \sum_{k=1}^{N} \alpha_p(k)(x_p(k) - \sum_{j \in source(k)} w_p(j,k) \cdot y_p(j)) \\
& \quad + \sum_{p=1}^{I} \sum_{k=1}^{N} \beta_p(k)(y_p(k) - F(x_p(k))) \\
& \quad + \sum_{p=1}^{I-1} \sum_{k=1}^{N} \sum_{j \in source(k)} \gamma_p(j,k)(w_{p+1}(j,k) - w_p(j,k) - \\
& \qquad \qquad r \sum_i \theta_i M_{i,p}(j, k, x_p(k), w_p(j,k), y_p(mod(k)), y_p(j)))
\end{aligned} \tag{16}
$$

Le Lagrangien est donc la somme de la fonction objective (équation (15)) et des contraintes du système (équations (12) à (14)). Les variables α, β, et γ représentent les multiplicateurs de Lagrange. Pour découvrir le gradient de l'erreur par rapport aux paramètres θ_i, il suffit de dériver le Lagrangien (16) par rapport à toutes les variables du système: $x_p(k)$, $y_p(k)$, $w_p(j,k)$, θ_i, $\alpha_p(k)$, $\beta_p(k)$, $\gamma_p(j,k)$, et à les forcer à 0. Après quelques substitutions, on peut isoler chaque variable, ce qui donne les équations suivantes:

La première équation nous donne le gradient par rapport aux x.

$$
\frac{\partial L}{\partial x_p(k)} = 0 \quad \rightarrow \quad \alpha_p(k) = \beta_p(k) \cdot F'(x_p(k)) + \sum_{j \in source(k)} (\gamma_p(j,k) \cdot r \sum_i \theta_i \frac{\partial M_{i,p}(j,k)}{\partial x_p(k)}) \tag{17}
$$

La deuxième équation nous donne le gradient par rapport aux y.

$$
\begin{aligned}
\frac{\partial L}{\partial y_p(k)} = 0 \quad \rightarrow \quad \beta_p(k) \quad = \quad & \sum_{j \in dest(k)} (\alpha_p(j) \cdot w_p(k,j)) \\
& + \sum_{j \in source(inv(k))} (\gamma_p(j, inv(k)) \cdot r \sum_i \theta_i \frac{\partial M_{i,p}(j, inv(k))}{\partial y_p(k)}) \\
& + \sum_{j \in dest(k)} (\gamma_p(k,j) \cdot r \sum_i \theta_i \frac{\partial M_{i,p}(k,j)}{\partial y_p(k)}) \\
& + \delta_k^{sortie}(d_p(k) - y_p(k))
\end{aligned} \tag{18}
$$

où δ_k^{sortie} vaut 1 lorsque $k \in sortie$ et 0 autrement.

L'équation (19) nous donne le gradient par rapport aux poids.

$$
\frac{\partial L}{\partial w_p(j,k)} = 0 \quad \rightarrow \quad \gamma_{p-1}(j,k) = \alpha_p(k) \cdot y_p(j) + \gamma_p(j,k) \cdot (1 + r \sum_i \theta_i \frac{\partial M_{i,p}(j,k)}{\partial w_p(j,k)}) \tag{19}
$$

Et l'équation (20) nous fournit ce que l'on cherchait: le gradient par rapport aux paramètres.

$$\frac{\partial L}{\partial \theta_i} = \sum_{p=1}^{I} \sum_{k=1}^{N} \sum_{j \in source(k)} -\gamma_p(j,k) \cdot rM_{i,p}(j,k) \tag{20}$$

Pour finir, on retrouve les équations de base du système:

$$\frac{\partial L}{\partial \alpha_p(k)} = 0 \quad \rightarrow \quad x_p(k) = \sum_{j \in source(k)} w_p(j,k) \cdot y_p(j) \tag{21}$$

$$\frac{\partial L}{\partial \beta_p(k)} = 0 \quad \rightarrow \quad y_p(k) = F(x_p(k)) \tag{22}$$

$$\begin{aligned} \frac{\partial L}{\partial \gamma_p(j,k)} = 0 \quad \rightarrow \quad w_{p+1}(j,k) \quad &= \quad w_p(j,k) + \\ & r\sum_i \theta_i M_{i,p}(j,k,x_p(k),w_p(j,k),y_p(mod(k)),y_p(j)) \end{aligned} \tag{23}$$

Afin de minimiser le coût défini dans l'équation (15), on peut modifier θ_i itérativement de la façon suivante:

$$\forall i \quad \theta_i = \theta_i - \epsilon \frac{\partial L}{\partial \theta_i} \tag{24}$$

où ϵ représente le pas de déplacement et $\frac{\partial L}{\partial \theta_i}$ le gradient par rapport aux paramètres θ_i. L'équation (24) peut se réécrire grâce à (20):

$$\forall i \quad \theta_i = \theta_i + \epsilon \sum_p \sum_k \sum_{j \in source(k)} \gamma_p(j,k) \cdot rM_{i,p}(j,k) \tag{25}$$

L'équation (25), combinée aux équations (17) à (19), nous permet donc de calculer le gradient de l'erreur par rapport aux paramètres θ_i, et partant, d'optimiser une règle d'apprentissage paramétrique par descente du gradient. En effet, on peut, à l'aide des équations (17) à (19), calculer récursivement à travers le temps tous les α, β, et γ nécessaires dans l'équation (25).

4 Résultats expérimentaux

Quelques expériences préliminaires ont été effectuées afin de vérifier la validité de notre approche. On peut en retrouver les détails dans [Beng91a, Beng91b, Beng92]. Nous décrivons rapidement ici les résultats des expériences booléennes.

Le but de ces expériences était d'explorer la possibilité d'optimiser une règle d'apprentissage pouvant être utilisée pour entraîner un réseau avec des neurones cachés. Il s'agissait de trouver une règle d'apprentissage capable de résoudre des tâches booléennes linéairement séparables (comme le ET ou le OU logique) ainsi que non linéairement séparables (comme le OU-EXCLUSIF ou le EGAL logique). La forme de la règle d'apprentissage est celle de l'équation

Table 1: Sommaire des résultats pour les expériences de fonctions booléennes. L signifie une transformation linéaire et NL une transformation non linéaire. La colonne *généralisation* indique si la règle trouvée pouvait être utilisée pour résoudre de nouvelles tâches.

#nombre de tâches	type de tâches	# nombre d'étapes d'optimisation	généralisation (à de nouvelles tâches) L	NL
1	L	3	oui	non
1	NL	15	oui	non
4	L	5	oui	non
5	4L, 1NL	100	oui	oui

(3) et le réseau de neurones utilisé est montré à la figure 2. Afin de fournir aux neurones cachés une information sur leur contribution à l'erreur, des neurones supplémentaires pouvant moduler les connexions synaptiques des neurones cachés ont été ajoutés.

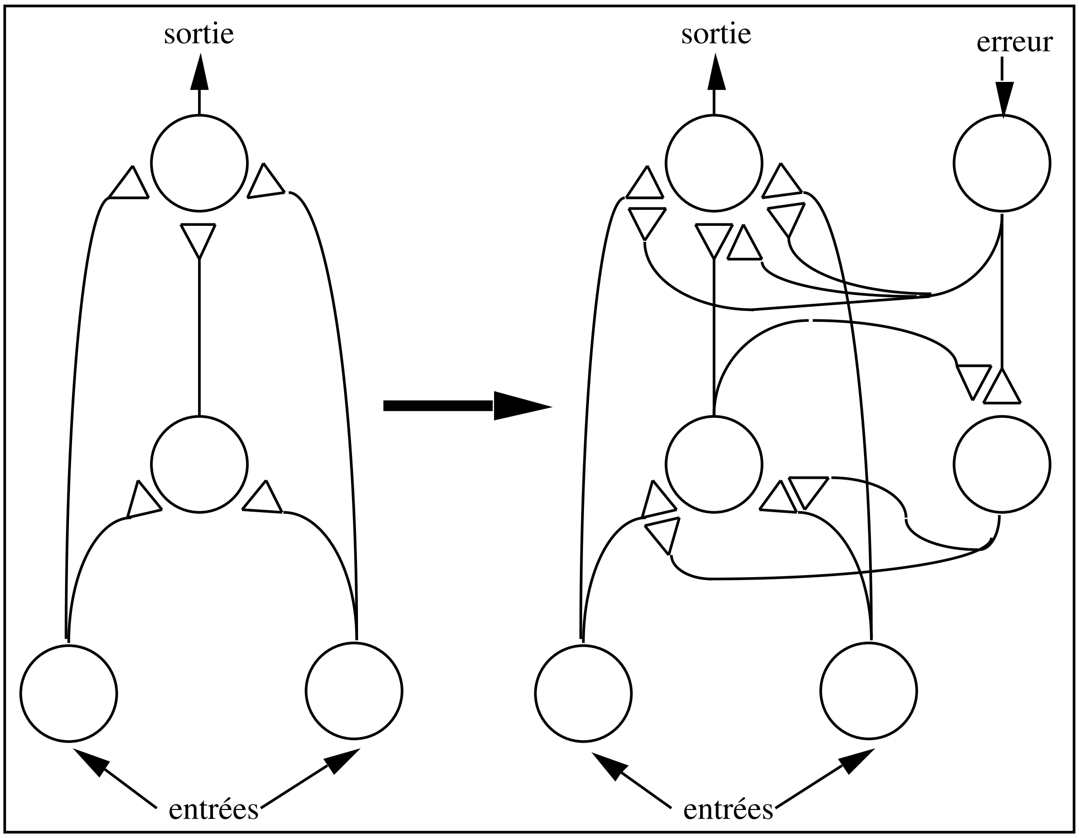

Figure 2: Architecture du réseau de neurones utilisé pour résoudre des tâches de fonctions booléennes. Des neurones modulateurs fournissent une information concernant l'erreur aux synapses des neurones cachés.

La tâche consistait à apprendre une transformation booléenne pendant un *cycle* de 800 présentations d'exemples bruités et choisis au hasard. Le bruit fut ajouté pour permettre une meilleure généralisation. Les paramètres de la règles d'apprentissage furent initialisés au hasard dans l'intervalle $[-1, 1]$ au début de l'optimisation. Avant chaque cycle, les poids du réseau étaient initialisés au hasard, de sorte que la règle d'apprentissage puisse s'appliquer à diverses conditions initiales. Les paramètres θ étaient mis à jour après chaque cycle selon la méthode décrite à la section 3. La table 1 résume les expériences réalisées. On peut voir notamment que lorsque la règle est *entraînée* avec des tâches simples (linéairement séparables), elle ne peut ensuite *généraliser* à des tâches plus complexes (non linéairement séparables). On voit aussi que le nombre de tâches utilisées pour l'entraînement des paramètres de la règle influence la généralisation de cette dernière, conformément aux résultats théoriques de la section 2.3.

De plus, afin de vérifier que la méthode d'optimisation par descente du gradient donnait de meilleurs résultats qu'une simple recherche aléatoire, ces deux techniques furent comparés. Pour la recherche aléatoire, plus de 5000 valeurs pour θ furent choisies dans $[-1, 1]^7$ (où 7 est le nombre de paramètres) et les meilleures furent conservées. Aucune règle ainsi trouvée ne put résoudre de tâche non linéairement séparable, alors que des règles trouvées par descente du gradient, non seulement pouvaient résoudre des tâches non linéairement séparables, mais pouvaient de plus généraliser à de nouvelles tâches.

5 Conclusion

Dans cet article, nous avons montré comment, grâce au formalisme du Lagrangien, on peut développer les formules nécessaires pour appliquer la descente du gradient pour l'optimisation de règles d'apprentissage paramétriques. Nous avons aussi jeté les bases théoriques qui permettent d'étudier la généralisation à de nouvelles tâches d'une règle d'apprentissage dont les paramètres ont été estimés avec un certain ensemble de tâches. Pour ce faire, nous avons montré comment on pouvait définir une mesure de capacité pour une famille de règles d'apprentissage.

Des expériences telles que celle exposée dans cet article montrent qu'il est possible de découvrir, par optimisation, des règles d'apprentissage capables de résoudre divers problèmes. On a pu ainsi découvrir des règles d'apprentissage pour des problèmes de conditionnement classique, de classification, ou encore de fonctions booléennes linéairement ainsi que non linéairement séparables [Beng92]. Ces règles d'apprentissage n'ont pas un sens mathematique évident (comme la descente du gradient par exemple, qui minimise un coût local), mais elles sont optimisées pour la classe de fonctions sur laquelle on les a entrainé. Il reste toutefois à montrer qu'il sera possible d'en faire autant avec des tâches plus complexes, ce qui demandera probablement des règles d'apprentissage plus sophistiquées. Il n'est pas sûr non plus que la descente du gradient soit la méthode d'optimisation la plus appropriée, particulièrement si l'espace à l'intérieur duquel on cherche est truffé de minima locaux. D'autres méthodes d'optimisation sont envisagées, telles que le recuit simulé ([Kirk83]) et les algorithmes génétiques ([Holl75, Gold89]). Ces dernières sont moins sensibles aux minima locaux, mais beaucoup plus lentes que la descente du gradient.

Références

[Baum89] Baum, E.R., Haussler, D. (1989) What Size Network Give Valid Generalization. *Neural Computation.* vol.1,pp.151-160.

[Beng90] Bengio, Y., Bengio, S. (1990) Learning a synaptic learning rule. Rapport technique #751. Département d'Informatique et de Recherche Opérationnelle. Université de Montréal.

[Beng91a] Bengio, Y., Bengio, S., Cloutier, J. (1991) Learning synaptic learning rules. Neural Networks for Computing. Snowbird, Utah.

[Beng91b] Bengio, Y., Bengio, S., Cloutier, J. (1991) Learning a synaptic learning rule. International Joint Conference on Neural Networks. Seattle, WA.

[Beng92] Bengio, S., Bengio, Y., Cloutier, J., Gecsei, J. (1992) On the Optimization of a Synaptic Learning Rule. Conference on Optimality in Biological and Artificial Networks. Dallas.

[Gard87] Gardner, D. (1987) Synaptic diversity characterizes biological neural networks. *Proceedings of the IEEE First International Conference on Neural Networks.* San Diego, CA. pp.IV.17-IV.22.

[Gluc87] Gluck, M.A., Thompson, R.F. (1987) Modeling the neural substrate of associative learning and memory: a computational approach. *Psychological Review.* vol.94, pp.176.

[Gold89] Goldberg, D. (1989) *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, Reading, MA.

[Hawk83] Hawkins, R.D., Abrams, T.W., Carew, T.J., Kandel, E.R. (1983) A cellular mechanism of classical conditioning in Aplysia: Activity- dependent amplification of presynaptic facilitation. *Science.* vol.219, pp.400-404.

[Hawk89a] Hawkins, R.D., Bower, G.H. (eds.) (1989) *Computational Models of Learning in Simple Neural Systems.* Academic Press.

[Hawk89b] Hawkins, R.D. (1989) A biologically based computational model for several simple forms of learning. Dans [Hawk89a]. pp.65-108.

[Hebb49] Hebb, D.O. (1949) *The Organization of behavior.* Willey. New York.

[Holl75] Holland, J. (1975) *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

[Kirk83] Kirkpatrick, S., Gellat, Jr., Vecchi, M.P. (1983) Optimization by Simulated Annealing. *Science.* 20:4598, pp.671-680.

[LeCu88] Le Cun, Y., (1988) A Theoretical Framework for Back-Propagation. in Touretzky, D., Hinton, G., Sejnowski, (eds.) *Proceedings of the 1988 Connectionist Models Summer School*, San Mateo. Morgan Kaufmann.

[Pavl32] Pavlov, I.P. (1932) *Les Réflexes Conditionnels.* Alcan, Paris.

[Rume86] Rumelhart, D.E., Hinton, G.E., Williams R.J. (1986) Learning Internal Representations by Error Propagation. in Rumelhart, D.E., McClelland, J.L. (eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition - Volume 1: Foundations.* Bradford Book. MIT Press.

[Vapn71] Vapnik, V.N., Chervonenkis, A.Y. (1971) On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications.* vol.16, no.2, pp.264-280.

[Vapn82] Vapnik, V.N. (1982) *Estimation of Dependencies Based on Empirical Data.* Springer-Verlag, New-York.