TRAVAUX PRATIQUES *Le perceptron de Rosenblatt*

Ce laboratoire consiste à évaluer la capacité du perceptron en tant que classificateur de données. Ce réseau relativement simple vous permettra aussi de faire un tour d'horizon des possibilités offertes par le simulateur JavaNNS.

Le perceptron peut être considéré comme le premier des réseaux de neurones adaptatifs, c'est-à-dire dont les poids des connexions sont changés par apprentissage. Le but du perceptron est d'associer une réponse à des signaux présentés en entrée. Le perceptron se compose, pour l'essentiel, de deux couches de neurones. À l'origine, la première couche était appelée la rétine du perceptron. La deuxième couche donne la réponse du perceptron à la stimulation donnée en entrée. Les cellules d'entrée sont reliées aux cellules de sortie grâce à des *synapses*.

L'apprentissage du perceptron s'effectue en modifiant l'intensité des synapses. Les cellules de sortie évaluent le degré de similitude de leur vecteur de poids synaptiques avec la stimulation en provenance des cellules de la rétine en effectuant la somme pondérée des intensités des cellules actives. Les cellules de sortie deviennent actives si leur degré d'activation dépasse un seuil fixé.

THÉORIE

La classification de données linéairement séparables

Un problème classique en reconnaissance de formes consiste à classifier des données selon une ou plusieurs catégories. Dans ce laboratoire, on désire utiliser le perceptron pour trouver la droite qui sépare les deux classes de données de la figure 5.a.

Chaque élément de cette figure est caractérisé par ses coordonnées x_1 et x_2, et par la famille y à laquelle il appartient. On a choisi d'associer la valeur +1 aux échantillons + et la valeur -1 aux échantillons ✱. Ces étiquettes sont arbitraires. Afin de fixer les idées, supposons que les données de la figure 5.a proviennent de mesures que l'on a faites sur des pommes et des tomates. La mesure x_1 représente le diamètre relatif du fruit par rapport à un étalon (d'où la possibilité de mesure négative) et la mesure x_2, son poids relatif. Les mesures étique-

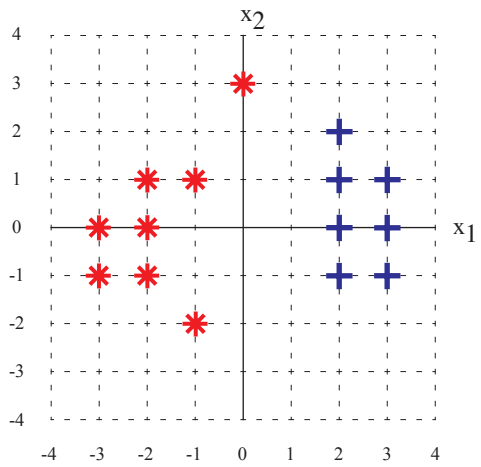

Figure 5.a Données linéairement séparables.

tées + correspondent aux tomates alors que celles étiquetées * correspondent aux pommes. En entraînant un simple perceptron avec les données de la figure 5.a, serait-il possible de déterminer automatiquement la catégorie de fruit ?

Le perceptron recevra donc 2 entrées, x_1 et x_2, et produira une seule sortie y, soit l'étiquette de l'élément correspondant. Le réseau à construire, illustré à la figure 5.b, est justement constitué de deux neurones d'entrée, l'un recevant l'entrée x_1 et l'autre, l'entrée x_2, et d'un neurone de sortie, qui indiquera la famille d'appartenance y.

Si w_1 et w_2 sont les poids entre les neurones d'entrée et le neurone de sortie, le réseau séparera l'espace de la fig. 5.a selon l'équation 5.a. Cette équation définit une droite; d'autre part, en posant pour ce laboratoire un seuil d'activation θ nul, la droite en question passera nécessairement par l'origine.

$$x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_2} \qquad (5.a)$$

L'apprentissage consiste à déterminer les poids w_1 et w_2 tels que tous les éléments + soient d'un côté de la droite, et tous les éléments * soient de l'autre côté. À cette étape, il faut donc indiquer au réseau si on veut obtenir une

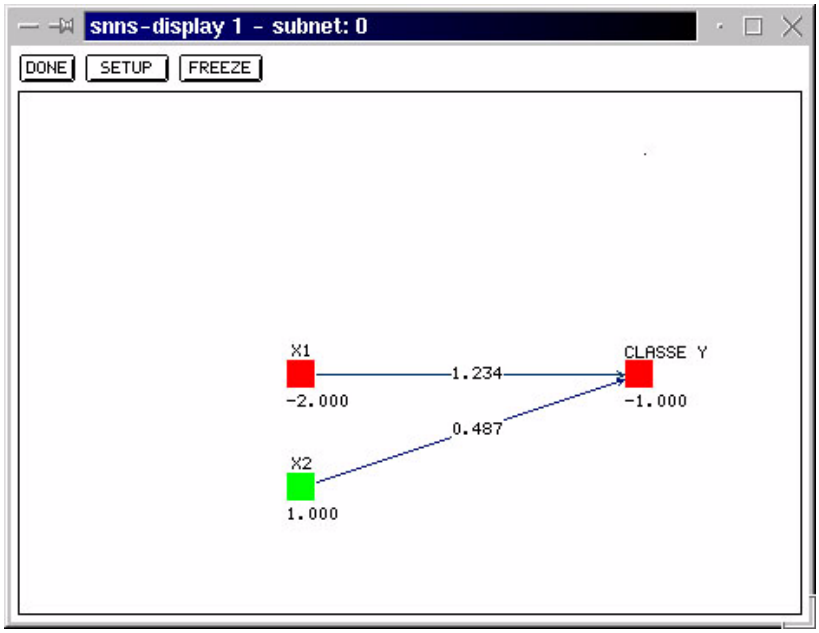

Figure 5.b *Architecture du perceptron à 2 entrées. Les chiffres indiquent la sortie des neurones et les poids des connexions.*

sortie de +1 ou -1 pour chaque paire (x_1,x_2) de la fig. 5.a. Le simulateur ajuste ses poids itérativement selon la règle du perceptron :

$$W(t+1) = W(t) + \eta(d(k) - classe(k))X_k \qquad (5.b)$$

avec $W(t) = [w_1 w_2]$ le vecteur de poids à l'itération t,

$X_k = [x_1(k)x_2(k)]$ les $k^{ièmes}$ entrées x_1 et x_2 du réseau,

d(k) la catégorie prédéterminée : +1 (+) ou -1(✱),

classe(k) la sortie du réseau, soit la catégorie calculée par le réseau, qui peut être conforme ou non à *d(k)*.

Selon cette équation, les nouveaux poids *W(t+1)* sont ceux de l'itération précédente, plus un facteur de correction qui dépend de la forme X_k observée, de l'erreur de sortie *(d(k)-classe(k))* et du taux d'apprentissage η.

La phase subséquente de généralisation consiste à observer la sortie du perceptron pour des paires (x_1,x_2) qui ne font pas partie de l'ensemble d'apprentissage.

MANIPULATIONS

Démarrage de JavaNNS

1. Lancez JavaNNS. Configurez votre environnement de simulation de façon à pouvoir visualiser votre réseau, afficher les courbes d'apprentissage, montrer le panneau général de commandes et enfin afficher la fenêtre des résultats. Votre environnement de simulation ressemblera à celui de la figure 5.c qui illustre un réseau à une couche cachée pour résoudre le problème du OU EXCLUSIF.

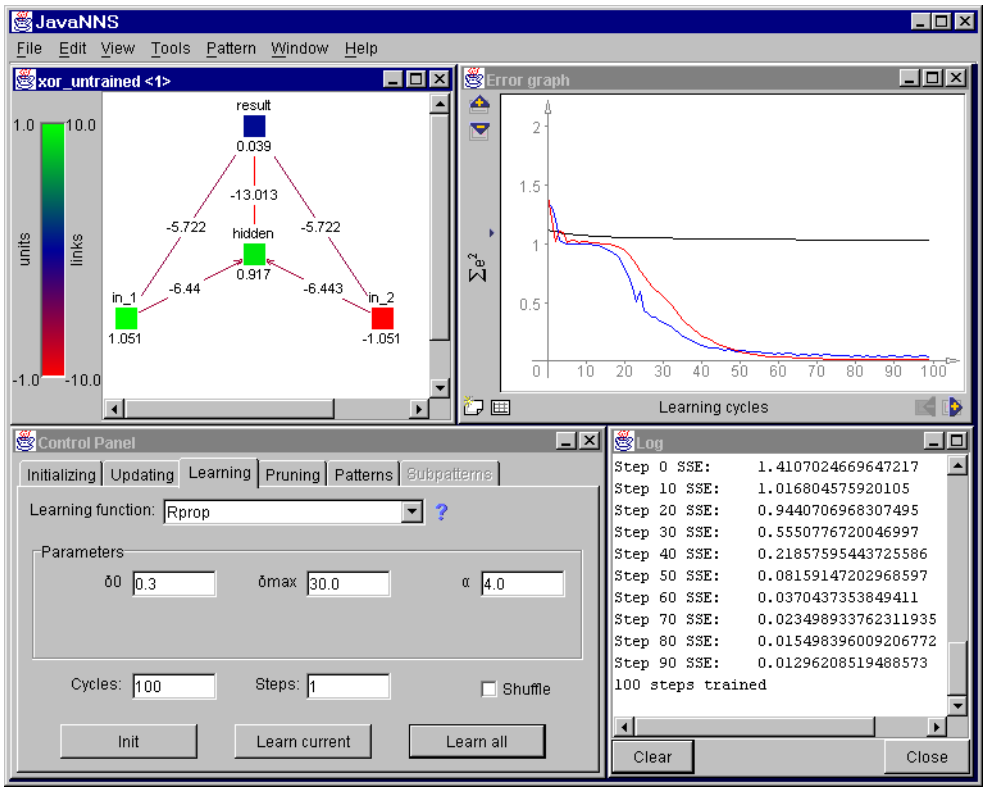

Figure 5.c Envvironnement de sinulation sous JavaNNS.
Le menu déroulant « File » permet de charger en mémoire ou de sauvegarder des descriptions de réseaux (suffixe .net), des données d'apprentissage ou de test (suffixe .pat) et des fichiers de configuration (suffixe .cfg). Le menu déroulant « View » permet de visualiser un réseau, les courbes d'erreur en apprentissage et le déroulement des opérations. Le menu déroulant « Tools » donne accès à des outils pour la conception et l'analyse de réseaux.

Construction du réseau avec l'outil *CREATE*

2. Choisissez l'item *Create⇒Layers* du menu déroulant *Tools*. Le panneau de construction de réseau apparaît (figure 5.d) et le réseau est ensuite bâti couche par couche.

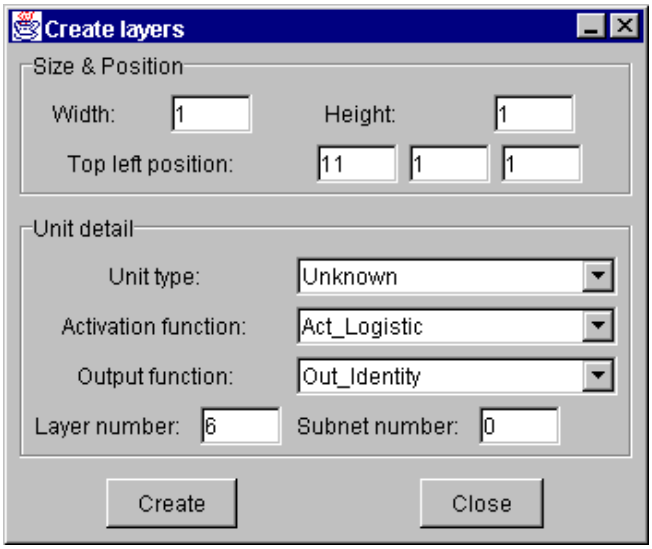

Figure 5.d *Panneau de construction des couches du réseau. Le réseau est construit couche par couche, en spécifiant pour chaque couche le type (entrée, cachée et sortie) et les fonctions d'activation et de sortie.*

3. La couche d'entrée est construite en spécifiant le type de couche (*Unit type = Input*), le nombre de neurones et leur disposition de même que leur fonction d'activation et de sortie. Entrez 1 pour le nombre d'unités dans la direction des x (*Width = 1)*, et 2 pour le nombre d'unités en direction des y (*Height = 2*) : ceci indique au simulateur qu'on désire créer une matrice de 1 x 2 neurones sur la couche. Puisque la couche d'entrée ne sert qu'à présenter au réseau la forme à classifier, les fonctions d'activation et de sortie sont unitaires (*Act_Identity* et *Out_Identity*). Les données pour la couche d'entrée sont enregistrées en cliquant sur le bouton *Create*.

4. Configurez la couche de sortie en spécifiant le type de couche (*Unit type = Output*). Entrez 1 pour le nombre d'unités en direction des x et également 1 pour le nombre d'unités en direction des y. La fonction de sortie est unitaire alors que la fonction d'activation est la fonction bipolaire (*Activation function = Act_Signum*). Le tableau 5.a résume les principales

Fonction SNNS	Nom	Expression
Fonctions d'activation		
Act_Identity	linéaire	$a(t) = net(t)$
Act_IdentityPlus Bias	linéaire avec polarisation	$a(t) = net(t) + \beta$
Act_Logistic	sigmoïde	$a(t) = \frac{1}{1 + e^{-(net + \beta)}}$
Act_Perceptron	perceptron	$a(t) = \left\{ \begin{array}{lll} 1 & si & net(t) \geq \theta \\ 0 & si & net(t) < \theta \end{array} \right\}$
Act_Signum	binaire	$a(t) = \left\{ \begin{array}{lll} 1 & si & net(t) > 0 \\ -1 & si & net(t) \leq 0 \end{array} \right\}$
Act_Signum0	binaire incluant zéro	$a(t) = \left\{ \begin{array}{lll} 1 & si & net(t) > 0 \\ 0 & si & net(t) = 0 \\ -1 & si & net(t) < 0 \end{array} \right\}$
Act_StepFunc	échelon	$a(t) = \left\{ \begin{array}{lll} 1 & si & net(t) > 0 \\ 0 & si & net(t) \leq 0 \end{array} \right\}$
Act_Tanh	tangente hyperbolique	$a(t) = \tanh(net(t) + \beta)$
Fonctions de sortie		
Out_Clip_0_1	linéaire bornée {0, 1}	$out(t) = \left\{ \begin{array}{lll} 1 & si & a(t) \geq 1 \\ a(t) & si & 0 < a(t) < 1 \\ 0 & si & a(t) \leq 0 \end{array} \right\}$
Out_Clip_1_1	linéaire bornée {-1, 1}	$out(t) = \left\{ \begin{array}{lll} 1 & si & a(t) \geq 1 \\ a(t) & si & -1 < a(t) < 1 \\ -1 & si & a(t) \leq -1 \end{array} \right\}$
Out_Identity	linéaire	$out(t) = a(t)$
Out_Threshold_ 0.5	binaire décalée	$out(t) = \left\{ \begin{array}{lll} 1 & si & a(t) > 0,5 \\ 0 & si & a(t) \leq 0,5 \end{array} \right\}$

Tableau 5.a Fonctions d'activation et de sortie usuelles

fonctions d'activation et de sortie offertes dans le simulateur. Les données pour la couche de sortie doivent être enregistrées en cliquant sur le bouton *Create*.
On a donc défini un réseau composé de deux neurones sur la couche d'entrée et d'un seul neurone sur la couche de sortie. Le réseau ne contient pas de couche cachée.

5. Les connexions entre les couches sont spécifiées par l'outil *Create⇒Connections* du menu déroulant *Tools*. Le tableau de configuration des connexions apparaît tel qu'illustré à la figure 5.e. Sélectionnez une connectivité directe totale (*Connect feed-forward)* et assurez-vous que les connexions supplémentaires directes entre les couches d'entrée et de sortie ne sont pas activées. Cette action définit une connectivité directe complète entre les couches. Le neurone de sortie du réseau sera connecté à chacun des neurones de la couche d'entrée.

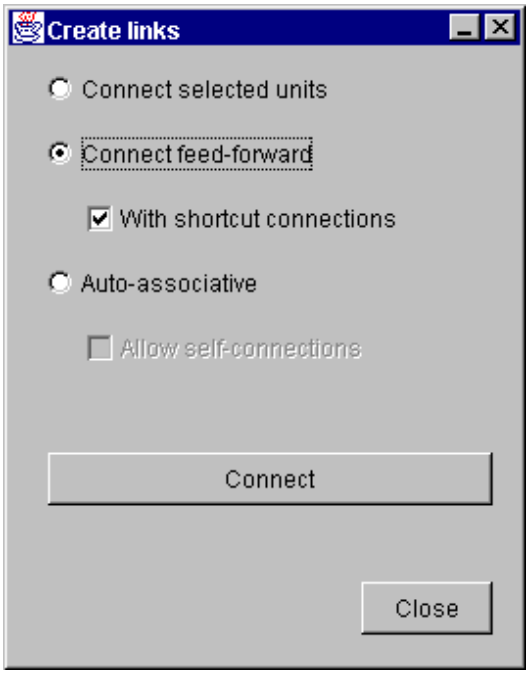

Figure 5.e Panneau de construction des liens.

6. L'identification de chaque neurone peut être personnalisée grâce au menu d'édition du simulateur. Le menu déroulant *Edit⇒Names* rend modifiable le champs de nom de tous les neurones et vous pouvez alors changer le nom

de n'importe quel neurone. Identifiez les neurones de la couche d'entrée par les noms *x1* et *s2* et le neurone de la couche de sortie par *classe y*.

Affichage du réseau avec le menu *View*

7. Le menu *View⇒Network* affiche le réseau à l'écran. Les paramètres de l'affichage comme le code de couleur utilisé et l'identification des neurones peuvent être modifiés au moyen du menu *View⇒Display settings*. La figure 5.f montre les deux panneaux qui permettent de modifier les divers paramètres d'affichage.

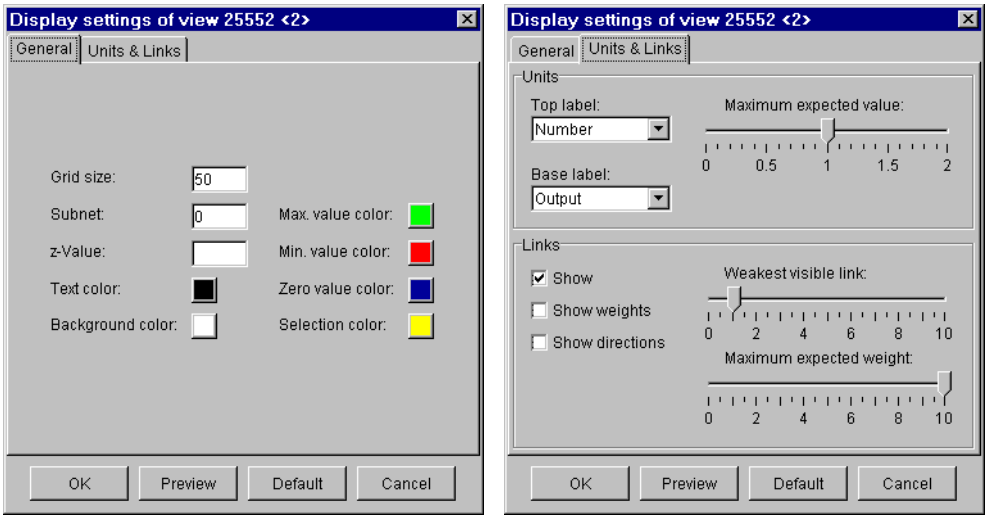

Figure 5.f *Paramètres d'affichage du réseau.*
Un code de couleurs permet de visualiser les valeurs de sortie des neurones et les valeurs de poids des connexions. On peut afficher, au dessus et au dessous de chaque neurone, divers paramètres du neurone comme son numéro, son nom, sa valeur de sortie, d'activation, etc.

À l'aide des deux panneaux d'ajustement de l'affichage, affichez le nom de chaque neurone au dessus de celui-ci et la valeur de sa sortie en dessous. Affichez les connexions du réseau de même que la valeur de poids associé à chaque lien.

8. Sauvegardez votre réseau sous le nom : *perceptron* à l'aide du menu déroulant *File⇒Save* ou *File⇒Save as*. JavaNNS rajoutera l'extension .net au nom du fichier.

Préparation des données d'apprentissage

Le perceptron est un réseau à apprentissage supervisé, c'est-à-dire qu'il apprend par l'exemple. On doit donc lui fournir des exemples de données déjà classifiées pour qu'il apprenne à les reconnaître. Ce modèle n'est d'ailleurs pas sans rappeler l'apprentissage chez les humains.

Les données d'apprentissage sont fournies au simulateur dans un fichier texte. La figure 5.g ci-dessous illustre la syntaxe du fichier texte qui doit être respectée afin que le simulateur puisse procéder à l'apprentissage de la base d'exemples.

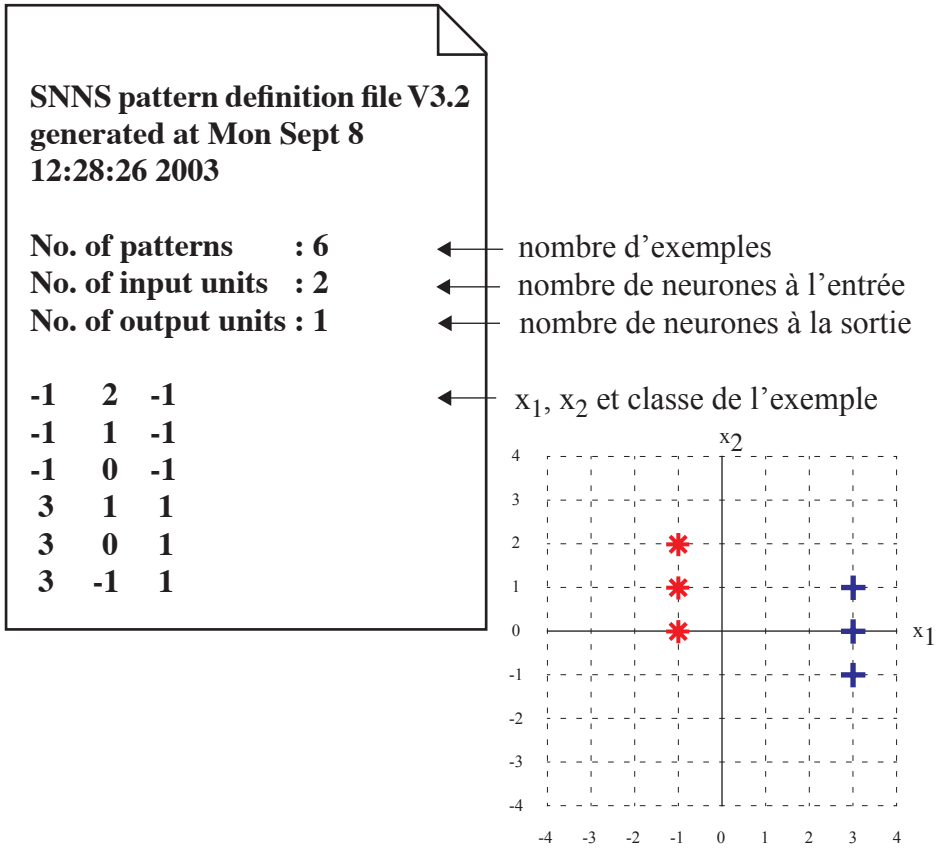

Figure 5.g *Base d'apprentissage.*
Les données d'exemple pour l'entraînement du réseau sont stockées dans un fichier texte de suffixe .pat. L'exemple ci-dessus est constitué des échantillons illustrées et montre la syntaxe qui doit être respectée afin d'être compatible avec JavaNNS.

9. Le fichier de données de la figure 5.g est disponible sur le site Internet du livre, dans le sous-répertoire *TP5/Donnees/exemple.pat*. Ouvrez ce fichier à l'aide d'un éditeur de texte, sauvegardez-le dans votre répertoire *SNNS* sous le nom figure5.a.pat, puis modifiez-le pour qu'il corresponde aux données de la figure 5.a.

Phase d'apprentissage du réseau avec *le panneau de commande « Control Panel »*

Le réseau doit maintenant apprendre à classifier les données, c'est-à-dire à placer correctement la droite de séparation au sein de la figure 5.a.

10. Pour charger les données en mémoire, sélectionnez le fichier *figure5.a.pat*.à l'aide du menu déroulant *File⇒Open*. À l'inverse, n'importe quelles données (apprentissage, test, résultats, etc.) peuvent être sauvegardées au moyen du menu déroulant *File⇒Save data*.

11. Le panneau de commande du simulateur, *Control Panel*, est de loin la fenêtre la plus importante du simulateur. C'est par ce panneau.que toutes les étapes d'une simulation sont initialisées et commandées. La panneau de commande est accessible par le menu déroulant *Tools⇒Control Panel*. Le panneau de commande offre plusieurs onglets pour spécifier les différentes étapes d'une simulation.

12. Pour confirmer que le simulateur lit correctement les données, cliquez sur l'onglet *Pattern* du panneau de commande. Le panneau de commande apparaît alors comme à la figure 5.h ci-dessous. Parcourez les données de la base d'apprentissage et vérifiez que les données d'entrée correspondent aux données emmagasinées. Dans ce mode de vérification, les données de la base d'apprentissage sont affichées sous forme de couleurs sur la couche d'entrée et sur la couche de sortie qui affiche alors la valeur désirée. Chacune des données de la base d'apprentissage peut aussi être propagée à travers le réseau grâce à l'onglet *Updating*, aussi illustré à la figure 5.h. Sélectionnez *Topological_Order*. En plus de l'affichage selon le code de couleur, ce mode permet de vérifier la valeur soumise à l'entrée du réseau, valeur qui est affichée sous chacun des neurones de la couche d'entrée.

    Remarquez qu'un neurone inhibiteur (sortie négative) devient rouge, alors qu'un neurone excitateur (sortie positive) devient vert.

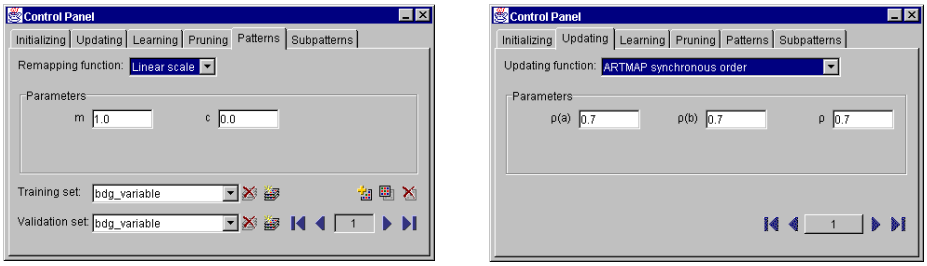

Figure 5.h Panneau de commande pour l'examen des formes soumises à l'entrée.

⇒ Pourquoi le neurone de sortie a-t-il toujours une sortie de -1.000 ?

13. Les paramètres de l'apprentissage sont ajustés au moyen de l'onglet *Learning* du panneau de commande, comme il est illustré à la figure 5.i ci-dessous.Sélectionnez la fonction *Backpropagation* comme fonction d'apprentissage. Entrez *0.1* dans la première case de cette ligne : c'est le taux d'apprentissage η. Entrez *0.1* dans la deuxième case : ceci correspond au seuil d_{max} de tolérance sur l'erreur, au-dessous duquel on n'exige pas de correction des poids[1].

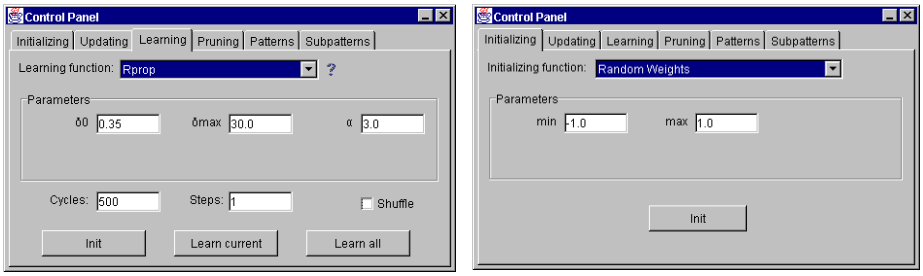

Figure 5.i Panneau de commande pour l'apprentissage. Un premier onglet permet de régler les paramètres de l'apprentissage tandis que l'onglet d'initialiser permet d'initialiser les poids de connexion du réseau.

Entrez *10* dans la case *Cycles*. L'apprentissage devra s'arrêter après 10 époques, ce qui est suffisant pour cet exemple simple. Pour des réseaux plus complexes, on limite plutôt le nombre initial d'époques aux alentours de 1000.
Les poids des connexions du réseau sont initialisés au moyen de l'onglet *Initializing*. Sélectionnez la fonction d'initialisation *Random Weights* dans

1. Cette fonctionnalité est sans effet pour le perceptron, dont la sortie est discrète. Vous devrez cependant ajuster d_{max} lors du TP sur le perceptron multicouche.

une plage de *-1* à *+1* afin d'initialiser tous les poids du réseau avec des valeurs aléatoires entre -1 et +1.

14. La direction de la propagation de la forme soumise à l'entrée à travers le réseau est spécifiée par l'onglet *Updating* du panneau de commande (figure 5.h à droite). Assurez-vous que la fonction *Topological_Order* est sélectionnée comme fonction de mise à jour.

15. Pour visualiser la courbe d'erreur lors de l'apprentissage, sélectionnez l'affichage de la courbe d'erreur dans le menu déroulant *View⇒Error Graph*. Trois fonctions d'erreur sont affichables, avec *n* le nombre de formes dans la base d'apprentissage, et *outputs* le nombre de neurones de sortie :

$$\sum e^2 = SSE = \sum_{k=1}^{n} \sum_{o=1}^{outputs} (d_n(k) - y_n(k))^2$$

$$\frac{1}{n}\sum e^2 = MSE = \frac{SSE}{n}$$

$$\frac{1}{outputs}\sum e^2 = SSE/\#\text{ out-unit} = \frac{SSE}{outputs}$$

La fonction *SSE* est proposée par défaut. Ajustez les échelles du graphique à l'aide des flèches sur les côtés de cette fenêtre pour voir 16 cycles en x, et une erreur *SSE* de 20 unités en y.
Le journal de bord des opérations réalisées par le simulateur peut être affiché par l'intermédiaire du menu déroulant *View⇒Log*.

⇒ De combien la fonction cumulative d'erreur *SSE* augmentera-t-elle à chaque fois que le réseau fait une erreur de classification?

16. Cliquez sur le bouton *INIT* dans l'onglet d'apprentissage du panneau de commande pour initialiser les poids du réseau, et notez ces valeurs initiales de poids. Lancez l'apprentissage en cliquant sur *Learn all*, et notez les poids finaux.

17. À partir des poids des connexions entre les neurones, calculez l'équation de la droite qui sépare les deux classes de données. Tracez cette droite sur la figure 5.a pour vous assurer que tous les éléments d'une même classe sont situés d'un seul côté de la droite.

En cas d'erreur :

a) Revérifiez vos données dans le fichier *figure5.a.pat* : correspondent-elles précisément aux données de la figure 5.a ?

b) Confirmez avec la fenêtre *Unit Properties* (souris droite ou menu déroulant *Edit*) que vos trois neurones utilisent les fonctions d'activation et de sortie mentionnées aux étapes 3. et 4.. Vérifiez que le neurone de sortie n'est pas configuré comme neurone caché.

Au besoin, refaites les étapes 16. et 17. après avoir corrigé le problème.

18. Faites un *snapshot* de votre courbe d'erreur pour le rapport. Pour ce faire, utilisez le programme de capture d'écran, disponible parmi les utilitaires de votre système d'exploitation : ceci vous permettra de « photographier » la fenêtre *Error Graph* et d'en créer une image lisible par votre logiciel de traitement de texte.

19. Sauvegardez votre réseau entraîné, comme à l'étape 8.

⇒ La coordonnée verticale x_2 était-elle vraiment nécessaire pour classifier les formes de la figure 5.a ? À la lumière de cette observation, commentez les poids finaux obtenus par le réseau.

Apprentissage du réseau, 2e version

Vous disposez maintenant d'un perceptron fonctionnel, prêt pour utilisation. Cependant, la méthode d'apprentissage que vous avez utilisée n'indique pas comment JavaNNS parvient à ce résultat : en effet, vous fournissez toutes les données d'apprentissage et le simulateur vous retourne d'un seul coup la solution finale. Pour mieux comprendre l'algorithme du perceptron, nous allons recommencer l'apprentissage en présentant les formes une à une, et nous calculerons nous-mêmes la correction à apporter aux poids.

Ne faites pas les opérations suivantes pour toutes vos données, ce serait inutilement long. Contentez-vous de 2 ou 3 formes mal classifiées, juste ce qu'il faut pour comprendre le mécanisme. Fournissez un exemple de calcul dans votre rapport.

20. Dans l'onglet d'apprentissage du panneau de commande, entrez 1 dans la case *Cycles*. Pesez sur le bouton *Init* pour initialiser vos poids à une valeur aléatoire, et notez les valeurs de poids pour usage ultérieur.

    Si vous le désirez, vous pouvez aussi repartir avec les mêmes poids initiaux qu'à l'apprentissage précédent (étape 16.) : dans ce cas, il vous suffit de sélectionner les deux neurones reliés par la connexion à modifier, puis de pointer un des deux neurones et sélectionner *Edit Links ...* en cliquant sur le bouton droit de la souris.

21. Faites afficher la fenêtre *Error graph* pour visualiser la courbe d'erreur. Au besoin, appuyez sur l'icone de nouvelle page pour effacer l'ancienne courbe.

22. Parcourez vos données avec les flèches de déplacement dans l'onglet *Updating* du panneau de commande. Pour chaque forme d'entrée, vérifiez la sortie du réseau dans la fenêtre qui affiche votre réseau.

23. Si la forme est correctement classifiée avec les poids actuels, passez à la forme suivante. Sinon :

    1. Tracez sommairement l'actuelle droite de séparation sur la figure 5.a
    2. Calculez les nouveaux poids avec la règle du perceptron (équ. 5.b) reproduite ci-dessous :

$$W(t+1) = W(t) + \eta(d(k) - classe(k))X_k$$

avec $W(t+1)$: nouveaux poids
$W(t)$: poids actuels $W(t) = [w_1 w_2]$
η : taux d'apprentissage
X_k : signal ou forme d'entrée $X_k = [x_1(k) x_2(k)]$
$d(k)$: valeur désirée en sortie pour X_k

$classe(k)$: valeur obtenue en sortie pour X_k

3. Calculez l'erreur $SSE = (d(k) - classe(k))^2$
4. Pressez le bouton *Learn current* pour indiquer à JavaNNS de corriger les poids pour cette seule forme. Les poids et l'erreur *SSE* apparaissant respectivement dans les fenêtres *Network* et *Error graph* doivent correspondre à ceux que vous avez calculés.
5. Tracez la nouvelle droite de séparation sur la fig. 5.a; observez dans quel sens elle s'est déplacée.

Continuez ce processus (étapes 22. et 23.) pour quelques autres formes.

Phase de généralisation du réseau

Il s'agit maintenant de vérifier si le perceptron peut classifier correctement des échantillons de forme qu'il n'a pas appris.

24. Préparez le fichier de données correspondant à la figure 5.j ci-dessous, de la même façon qu'à l'étape 9. Écrivez 0 sur la ligne « *No. of outputs units* », et entrez seulement les valeurs (x_1,,x_2) des élément de la figure 5.j à titre de données. Sauvegardez ce fichier sous le nom figure5.j.pat.

25. Chargez ce fichier en mémoire comme à l'étape 10..

26. Parcourez toutes les formes utilisées pour vérifier les capacités de généralisation (fig. 5.j) avec les flèches de déplacement dans l'onglet *Updating* du panneau de commande et observez la valeur de sortie du perceptron dans la fenêtre d'affichage du réseau. Comparez ces résultats avec la droite de séparation que vous avez obtenue à l'étape 17.. AVERTISSEMENT : ne touchez pas au bouton *Init* de l'onglet d'apprentissage, ce qui changerait les valeurs de poids.

⇒ À quelle classe est assignée la forme située à (0, 0), point qui appartient à la ligne de séparation, et surtout pourquoi? Aurait-elle dû être classée autrement?

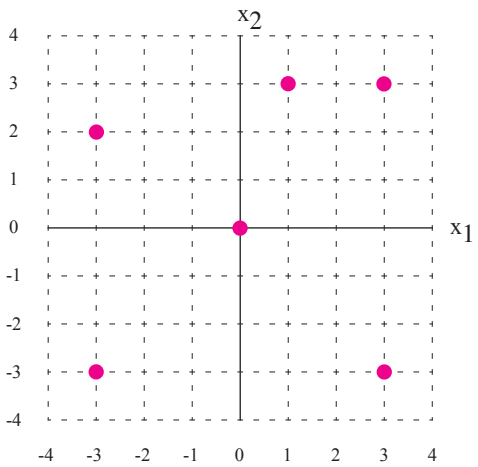

Figure 5.j *Généralisation.*
Échantillons de données qui n'ont pas été apprises par le réseau.

27. Sauvegardez les résultats de la généralisation : menu déroulant *File⇒Save data*, tapez le nom de fichier *figure5.j*, spécifiez le type de fichier *Result files *.res*, puis *Save*. JavaNNS ajoute automatiquement l'extension .res au nom de fichier. Vous pouvez consulter ce fichier avec un simple éditeur de texte.

    Lorsqu'il y a beaucoup de données (comme pour le TP sur le perceptron multicouche), il est généralement plus pratique d'écrire les résultats dans un fichier qui sera analysé ultérieurement, plutôt que de parcourir les formes une à une comme vous l'avez fait.

Un exemple de données non-séparables linéairement

On étudiera ici le comportement du perceptron avec des données non séparables linéairement.

28. Écrivez le fichier de données correspondant à la figure 5.k ci-dessous.

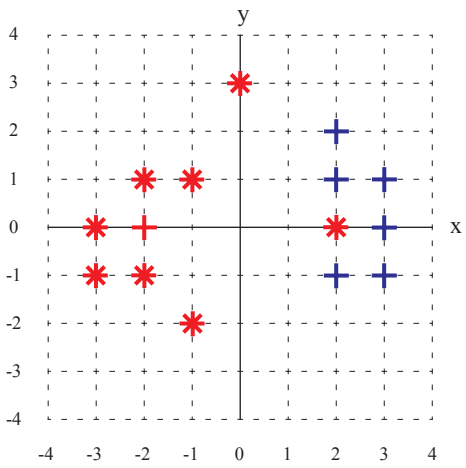

Figure 5.k Données non-séparables linéairement.

29. Chargez ce fichier en mémoire.

30. Réinitialisez votre réseau en cliquant sur le bouton *Init* de l'onglet d'apprentissage.

31. Recommencez l'apprentissage avec les données de la figure 5.k, en augmentant le nombre de cycles d'apprentissage. Expérimentez avec ou sans *SHUFFLE* (dans l'onglet d'apprentissage), qui indique au simulateur de parcourir les données d'apprentissage dans un ordre aléatoire ou séquentiel.

32. Faites une capture d'écran de la courbe d'erreur. Tracez la droite de séparation obtenue (à partir des poids finaux) sur la fig. 5.k.

⇒ Concluez sur la pertinence du perceptron pour classifier ce genre de données. À quoi pourrait servir le perceptron dans l'exploration de données inconnues?

Un exemple plus complexe de données séparables linéairement

Jusqu'ici, la droite de séparation passait obligatoirement par l'origine. Cette contrainte n'a pas empêché le réseau d'apprendre à séparer correctement les données de la fig. 5.a puisqu'il existe toute une famille de droites-solutions passant par l'origine. Mais cette contrainte empêcherait le réseau de converger vers une solution pour les données de la fig. 5.l ci-dessous. L'hyperplan de séparation

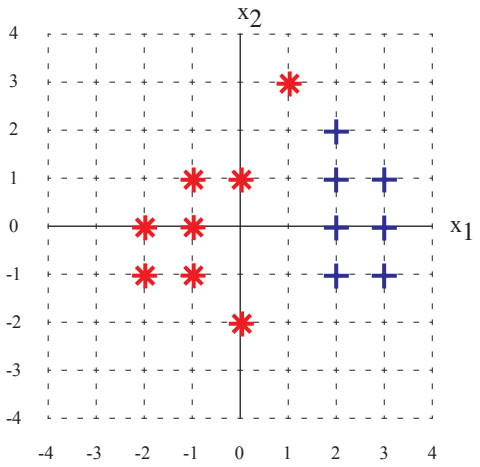

Figure 5.l *Données linéairement séparables. Un degré de liberté doit être ajouté parce que la droite de décision ne passe pas par l'origine.*

des deux classes doit donc pouvoir se déplacer dans l'espace des solutions et non pas être restreint à passer par l'origine. Ce déplacement est matérialisé en ajoutant une valeur de seuil θ au neurone. Cette valeur de seuil est variable et sera entraînée comme toutes les valeurs de poids des connexions parvenant au neurone. C'est pourquoi dans plusieurs simulateurs, cette valeur de seuil est implantée en rajoutant une connexion supplémentaire connectée à une entrée fixe égale à -1. Dans JavaNNS, l'utilisation d'un seuil s'effectue par le choix de la fonction d'activation, dont certaines incluent une valeur de seuil ou de polarisation.

33. Modifiez la fonction d'activation du neurone de classification en choisissant la fonction *Act_Perceptron*. Avec cette fonction d'activation, le neurone de classification émet une sortie binaire $\{0, 1\}$ et est muni d'une valeur de

seuil qui est entraînée tout comme les autres valeurs de poids des connexions qui parviennent à ce neurone.

34. Les données de la fig. 5.l sont disponibles sur le site Internet du livre à l'emplacement *TP5/Donnees/lineaire.pat*. Chargez ce fichier en mémoire.

35. Lorsque l'apprentissage de la nouvelle base de données sera terminée, notez la valeur atteinte des poids de connexion et la valeur de seuil du neurone de classification. Tracez la droite de séparation sur la fig. 5.l.

Rapport

Votre rapport devrait inclure les éléments suivants :

- Court exposé du problème à résoudre : problème de classification.
- Fonctionnement du perceptron, avec équations à l'appui.
- Analyse des résultats : droite de séparation, convergence de l'apprentissage, pertinence des classes obtenues en généralisation, etc.
- Conclusions et observations.

N'oubliez pas de répondre spécifiquement aux questions identifiées dans le texte par une flèche ⇒.

Incluez aussi dans votre rapport une illustration du réseau, vos courbes d'erreur, et la définition du réseau (contenu du fichier *perceptron.net*) après apprentissage.

TRAVAUX PRATIQUES *Les réseaux Adaline et Madaline*

Ce laboratoire reprend essentiellement la problématique de classification de données linéairement séparables du TP précédent, mais cette fois en évaluant les performances du réseau Adaline de Widrow-Hoff et en les comparant avec celles du perceptron de Rosenblatt. Nous examinons également la possibilité d'ajouter une couche cachée au réseau Adaline afin de résoudre des problèmes de classification de données qui ne sont pas linéairement séparables. Le Madaline constituait une première solution au problème de classification non-linéaire qui sera généralisé plus tard par le perceptron multicouche à rétropropagation d'erreur. Nous illustrons ici l'utilisation du Madaline pour résoudre le problème non linéaire classique du XOR.

L'ADALINE DE WIDROW-HOFF

Le réseau Adaline a été développé à la même période que le perceptron de Rosenblatt, et les deux réseaux présentent plusieurs similarités. La différence la plus marquée est que l'Adaline de Widrow-Hoff ajuste ses poids selon un critère de moindre carré en comparant la sortie désirée avec la valeur nette du neurone, et non avec la sortie quantifiée comme dans le cas du perceptron de Rosenblatt.

Nous allons comparer ces deux réseaux à l'aide des mêmes données à classifier que nous avons utilisées pour le perceptron. Les données sont celles de la fig. 5.l qui sont reprises ci-dessous.

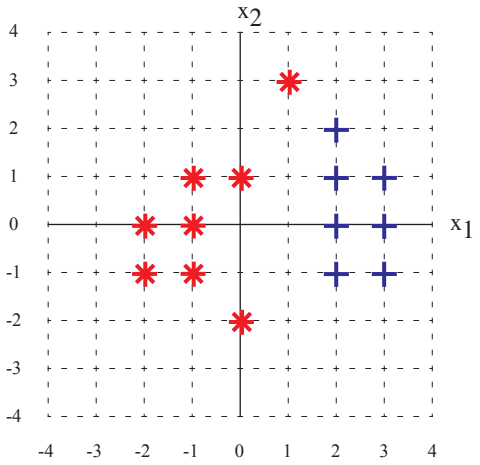

Figure 5.m Données linéairement séparables.

Le simulateur JavaNNS ne permet pas d'entraîner les connexions d'un neurone à partir de la valeur d'activation de ce dernier, comme l'exige l'algorithme de Widrow-Hoff. Nous remédions à cette lacune en construisant la couche de classification à l'aide de deux sous-couches en cascade : une première sous-couche avec fonction d'activation et de sortie toutes les deux linéaires, qui fournira la valeur nette pour l'apprentissage, suivie d'une seconde sous-couche à ***poids fixe*** unitaire et munie d'une fonction d'activation binaire afin de quantifier la sortie. Le réseau à construire est illustré à la figure 5.n.

Manipulations

1. Construisez le réseau Adaline selon l'architecture de deux couches de la figure 5.n. La première couche, celle du *combinateur linéaire adaptatif*, est la portion du réseau dont les poids seront entraînés. Les étapes de construction sont similaires aux étapes de construction du perceptron vues en détail au TP précédent (étapes 1 à 8.). Ces étapes sont résumées ci-dessous :
   1.1 Après avoir démarré JavaNNS, construisez le réseau à partir du menu déroulant *Tools⇒Create⇒Layers*.

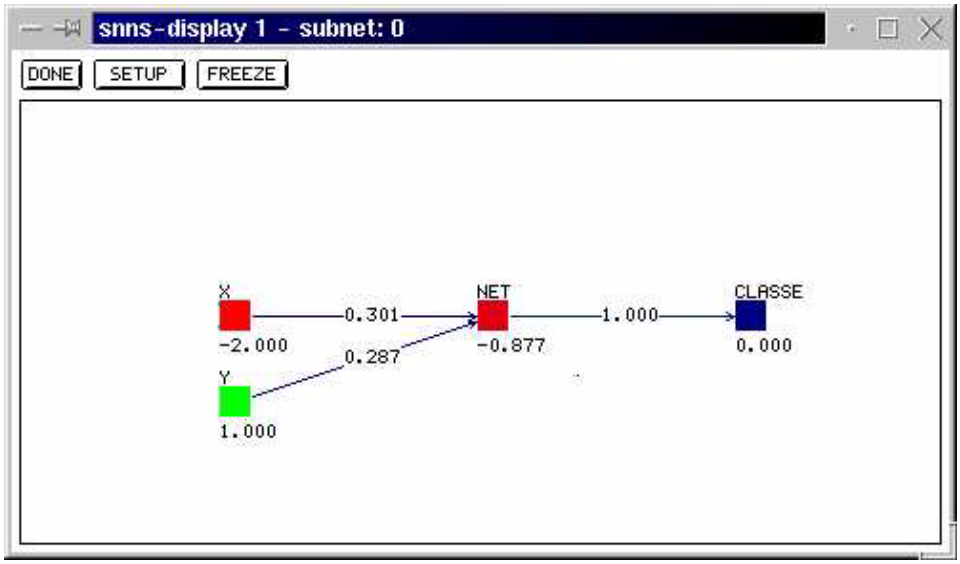

Figure 5.n *Architecture de l'Adaline à 2 entrées.*
La couche de classification est séparée en deux sous-réseaux, un premier qui constitue un combinateur linéaire adaptatif et un second qui ne participe pas à l'entraînement pour binariser la sortie du neurone de classification.

1.2 Construisez un réseau avec deux neurones à l'entrée, un neurone en couche cachée et un neurone en sortie. Les connexions sont de type *Connect feed-forward*.

1.3 Visualisez le réseau à l'aide du menu déroulant *View⇒Network* et affichez les liens, le nom des neurones, la valeur des poids ainsi que la valeur de polarisation ou de sortie, au choix, de chaque neurone comme à la figure 5.n.

1.4 Changez les fonctions d'activation et de sortie de chaque neurone à l'aide de la fenêtre *Unit Properties* ou, mieux, lors de la création du réseau.
Plus spécifiquement :

a) couche d'entrée :
- nom : x1 et x2
- fonction d'activation : Act_Identity
- fonction de sortie : Out_Identity

b) couche cachée (intermédiaire) :
- nom : net
- fonction d'activation : Act_IdentityPlusBias

- fonction de sortie : Out_Identity

c) couche de sortie - c'est la couche qui ne doit pas être entraînée :
- nom : classe y
- type de neurone : Special output
- fonction d'activation : Act_Signum
- fonction de sortie : Out_Identity
- poids fixe non entraîné : fixer à 1 la valeur de ce poids à l'aide du menu contextuel (bouton droit de la souris) *Edit Links*

1.5 Ajustez les paramètres de l'apprentissage au moyen de l'onglet d'apprentissage du panneau de commande :
- fonction d'apprentissage : Backpropagation
- taux d'apprentissage : 0.1
- erreur minimum : 0.01
- mise à jour : Topological_Order
- nombre maximum d'époques : 10
- initialisation des poids : Random Weights

1.6 Affichez l'erreur d'apprentissage au moyen de la fenêtre *Error graph*.

2. Chargez en mémoire du simulateur les données de la fig. 5.m disponibles sur le site Internet du livre à l'emplacement *TP5/Donnees/lineaire.pat*.

3. Procédez à l'entraînement du réseau en utilisant l'algorithme d'apprentissage *Backpropagation*. Tracez la courbe d'erreur et comparez avec la courbe d'erreur du perceptron.

4. Tracez sur la fig. 5.m la droite de séparation obtenue après l'apprentissage du réseau Adaline. Tracez également sur le même graphique la droite obtenue par l'entraînement du perceptron de Rosenblatt. Commentez les différences entre les deux droites de séparation.
   Étant donné que le réseau utilise une valeur de polarisation β au lieu d'une valeur de seuil θ pour la fonction d'activation, l'équation 5.a de la droite de décision devient : $x_2 = -\frac{w_1}{w_2}x_1 - \frac{\beta}{w_2}$. La seule différence réside dans le signe du second terme de l'équation, à cause des définitions de la polarisation et du seuil, qui sont à l'opposé l'une de l'autre.

⇒ À l'aide du tableau 5.a des fonctions d'activation et de sortie disponibles sous JavaNNS, démontrer les équations de droite de décision dans les deux

cas où une valeur de seuil et une valeur de polarisation sont utilisées. Discuter de l'équivalence fonctionnelle entre ces deux termes.

5. Vérifiez le bon fonctionnement du réseau Adaline d'abord avec les données qui ont servi à l'apprentissage (fichier *TP5/Donnees/lineaire.pat*, en écrivant 0 sur la ligne « *No. of outputs units* » et en supprimant la colonne de sortie désirée), puis avec les données de la fig. 5.j pour vérifier la capacité de généralisation du réseau. Vous pouvez avantageusement utiliser la méthode de parcours des données avec les flèches de défilement sous l'onglet *Updating* du panneau de commande dans JavaNNS.

6. Quel serait le comportement du réseau Adaline dans le cas où les données ne sont pas séparables linéairement, comme à la fig. 5.k ?

LE RÉSEAU MADALINE

Le réseau Madaline est une des premières réponses apportées au problème de données qui ne sont pas séparables linéairement. Le réseau est composé de deux couches : une première couche, entraînable, constituée de neurones Adaline, et une seconde couche de sortie constituée de neurones formels de McCulloch & Pitts à poids fixes et réalisant des fonctions logiques de base.

Nous allons utiliser le réseau Madaline pour résoudre le problème classique du OU EXCLUSIF qui avait montré à l'époque une limite fondamentale du perceptron de Rosenblatt. Il faut donc concevoir un réseau de neurones capable de classifier correctement les deux entrées à une fonction OU EXCLUSIF. Les deux catégories ne sont pas séparables par une seule ligne droite, comme le montre la fig. 5.o ci-dessous.

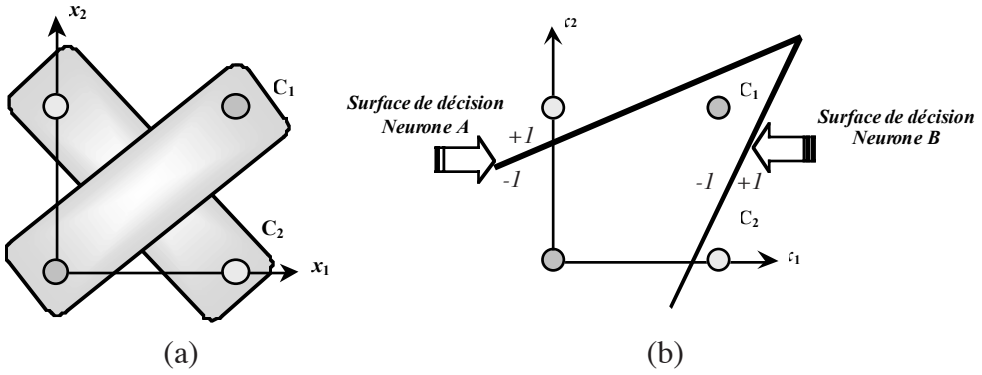

Figure 5.o *Problème de la fonction logique OU EXCLUSIF.*
a) la sortie «vraie» ne peut pas être isolée de la réponse «fausse» par une ligne droite.
b) une solution au problème consiste à utiliser une première couche de neurones configurée en réseau Adaline, qui génère les deux droites de séparation C_1 et C_2. Une seconde couche combine par une opération OU les deux catégories précédentes pour fournir la bonne réponse.

Le réseau Madaline à deux couches pour résoudre le problème du OU EXCLUSIF est illustré à la figure 5.p ci-dessous. Détaillez dans votre rapport les paramètres de chacun des neurones constituant le réseau Madaline.

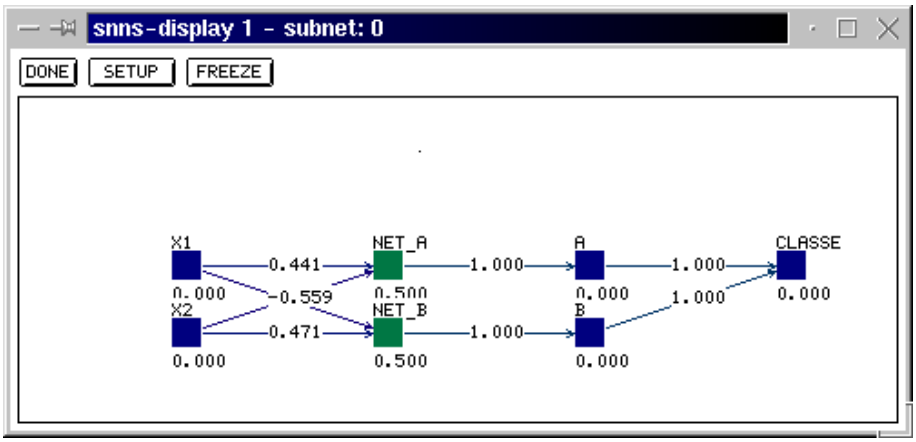

Figure 5.p *Réseau Madaline.*
Réseau à deux couches pour résoudre le problème du XOR.

Manipulations

1. Étudiez d'abord le réseau Adaline de la figure 5.q constitué des 6 premiers

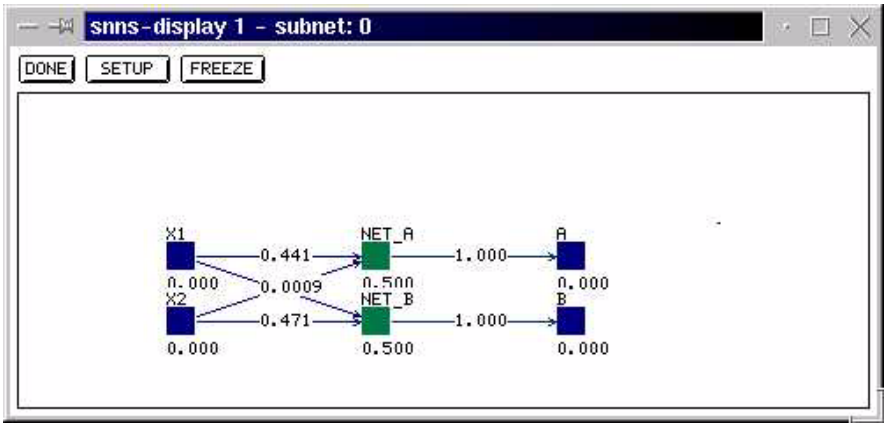

Figure 5.q *Première couche du réseau Madaline.*
Cette couche est constituée de deux neurones de type Adaline alimentés par les entrées x_1 et x_2. Cette couche est entraînée pour produire les deux lignes de décision C_1 et C_2.

neurones de la figure 5.p. La principale difficulté rencontrée dans la mise en oeuvre de ce réseau réside dans la connectivité entre les deux dernières

couches, qui sont de type 1-à-1 alors que la connectivité complète est le type de connectivité par défaut de JavaNNS. Il faut donc modifier la connectivité à l'aide de l'outil *Tools⇒Create⇒Connections* utilisé lors de la construction du réseau. Un neurone source est d'abord sélectionné, puis l'outil de connexion est exécuté (*Tools⇒Create⇒Connections*) et l'opération *Connect selected units* est choisie. Le bouton *Mark selected units as source* est pressé, puis le neurone de destination est sélectionné, et enfin le bouton *Connect source with selected units* est pressé. Le processus est recommencé pour connecter les deux derniers neurones. Pour les besoins de ce TP, vous trouverez la bonne configuration du réseau sur le site Internet du livre à l'emplacement *TP5/Traitements/madalineapp.net*. Configurez les neurones comme une couche Adaline, tel que vu à la première partie de ce TP.

2. Par la suite, entraînez le réseau à différencier les deux zones A et B définies à la figure 5.o ci-dessus. Entraînez le réseau avec les données du OU EXCLUSIF. Constituez la base d'apprentissage requise et joignez-la à votre rapport. Prenez bonne note de toutes les valeurs de poids de connexion et de polarisation afin de pouvoir les transférer au réseau complet du Madaline. Tracez dans votre rapport les droites de séparation apprises par votre réseau.

3. L'étage de sortie du Madaline est constitué d'un réseau à poids fixes qui réalise une fonction logique. Dans notre cas, le problème du OU EXCLUSIF sera résolu en soumettant les réponses obtenues des neurones A et B à une fonction logique OU. Complétez la construction du Madaline en ajoutant une porte OU réalisée par le neurone final de sortie. La configuration du réseau complet se trouve dans le fichier *TP5/Traitements/madaline.net* qui prend en charge la configuration des diverses connexions du réseau. Initialisez le réseau avec les valeurs de poids de toutes les connexions et les valeurs de polarisation des neurones appropriés que vous avez notées à la suite de l'entraînement de la section Adaline. Configurez le neurone de sortie afin qu'il réalise une opération logique OU.

4. Testez le fonctionnement de votre réseau.