Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association Suisse des Electriciens, de l'Association des Entreprises électriques suisses

Band: 83 (1992)

Heft: 5

Artikel: Simulation de réseaux de neurones artificiels

Autor: Blayo, François / Demartines, Pierre

DOI: https://doi.org/10.5169/seals-902802

Simulation de réseaux de neurones artificiels

François Blayo, Pierre Demartines

La pratique des réseaux neuronaux requiert une maîtrise des paramètres et des architectures des modèles qui ne peut être acquise par une seule approche théorique. Une simulation sur ordinateur permet d'avoir facilement cette connaissance intuitive qui permet de décider de l'applicabilité d'un modèle. Une méthode de conception de simulateurs didactiques de réseaux neuronaux est présentée dans cet article ainsi que les éléments de base nécessaires pour les construire.

Die Arbeit mit neuronalen Netzwerken verlangt eine Beherrschung der Parameter und Architekturen der Netzwerk-Modelle, welche nicht durch einen theoretischen Ansatz allein zu erreichen ist. Rechner-Simulationen erlauben, auf einfache Weise jene intuitiven Kenntnisse zu erlangen, welche man für den Entscheid über die Anwendbarkeit eines Modells benötigt. Der vorliegende Artikel beschreibt eine Entwurfsmethode für didaktische Simulatoren von neuronalen Netzwerken sowie die zu deren Konstruktion nötigen Basis-Elemente.

Adresse des auteurs

Dr. *François Blayo* et *Pierre Demartines*, ing. dipl. EPFL, Ecole Polytechnique Fédérale de Lausanne, Laboratoire de Microinformatique, INF – Ecublens, 1015 Lausanne.

Dans le cadre de toute formation, l'approche pratique de la matière passe par le développement d'outils dédiés à l'expérimentation. Le domaine des réseaux de neurones, encore jeune, réclame d'autant plus ces outils que les théories ne sont pas encore connues. On privilégie donc un enseignement intuitif, basé sur des simulateurs logiciels, qui peut servir à toute personne désireuse de développer des applications à base de réseaux neuronaux. Cela n'exclut évidemment pas une démarche plus théorique, qui doit compléter cette première approche.

Dans cet article, on présente un jeu de simulateurs de modèles connexionnistes considérés aujourd'hui comme des modèles de base. Les deux volets, enseignement et expérimentation, sont décrits ainsi que les principes qui ont guidé le développement logiciel. On décrit enfin les problèmes spécifiques liés à la simulation numérique des modèles connexionnistes.

Fonction des simulateurs

Les multiples raisons qui ont conduit au développement de simulateurs de réseaux de neurones peuvent être classées en deux catégories: les raisons liées à l'enseignement et celles liées à l'expérimentation. La démarche générale de construction réside en deux points: développement d'un cœur algorithmique qui simule le modèle retenu et d'un environnement graphique qui permet la visualisation de tous les constituants du modèle. Cet environnement graphique permet également de gérer les interactions entre l'utilisateur et le simulateur.

Enseignement

Les mécanismes fondamentaux des réseaux de neurones [1] sont difficilement accessibles sans un bagage théorique important. La matière n'est d'ailleurs enseignée que dans les années terminales des écoles d'ingénieurs et universités. Malgré le niveau en général élevé des auditeurs de cours, on a constaté que les démonstrations formelles ne permettaient pas de comprendre en profondeur les applications possibles des modèles présentés. Cela est dû à plusieurs facteurs, parmi lesquels on retiendra les suivants:

- la structure massivement parallèle des réseaux ainsi que l'évolution asynchrone des états des neurones sont des mécanismes complexes difficilement accessibles théoriquement,
- la notion d'apprentissage et non de programmation rompt avec les concepts traditionnels de l'informatique,
- la nouveauté des modèles qui ne sont pas encore consolidés par une théorie unique, cela se traduit par une multitude de paramètres associés à chaque modèle, qui sont parfois dépendants du temps.

Pour essayer de contourner ces difficultés, on a choisi de construire les simulateurs sur trois principes: interactivité, spécialisation et souplesse.

Interactivité

Le principe d'interactivité repose sur la possibilité d'agir à tout moment sur l'environnement de simulation. Le fonctionnement du modèle simulé doit se faire en temps réel, et toute modification des paramètres doit avoir une répercussion visuelle immédiate. On a donc éliminé tout langage de commande pour dialoguer avec le simulateur [2; 3]. L'entrée des données est assurée par l'action sur des potentiomètres (fig. 1) et sur des boutons

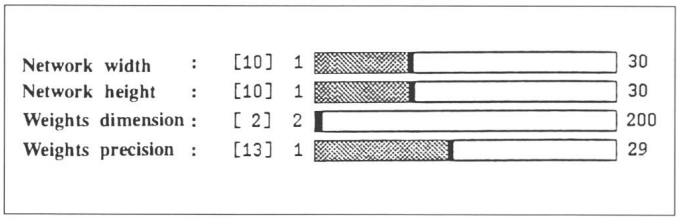


Figure 1
Exemples de potentiomètres

(fig. 2). Les valeurs associées sont affichées instantanément en regard des potentiomètres. Pour les paramètres propres à l'adaptation on peut ainsi connaître, par simple déplacement aux extrêmes de la course possible des potentiomètres, les valeurs minimales et maximales admissibles pour le paramètre concerné. Cette fonction est essentielle pour assurer un fonctionnement toujours correct de la simulation. Elle donne une information quantitative qui guide l'utilisateur en cours d'expérimentation. Elle permet également de connaître les ordres de grandeur de tous les paramètres, information souvent omise dans les présentations théoriques.

Les valeurs associées aux éléments du modèle (potentiel et valeur de sortie du neurone, poids synaptique, etc.) sont représentées par des couleurs, permettant de mettre en évidence leurs variations, même rapides. Ce mode de représentation renforce une compréhension qualitative du fonctionnement du modèle simulé. Les valeurs exactes peuvent être connues en cliquant sur la représentation des éléments. Elle est donc ressentie comme une donnée facultative, accessible pour une analyse profonde du modèle.

La grande liberté d'action laissée à l'utilisateur dans l'exploration des simulateurs engendre des contraintes de cohérence pour la gestion du contexte de simulation. Les paramètres n'ont pas tous la même importance sur l'état de la simulation. Par exemple, la modification manuelle du paramètre d'adaptation α n'a qu'une influence numérique sur le déroulement de la simulation. Par contre, la modification du nombre de neurones doit entraîner une réinitialisation du contexte de simulation. Il y a donc une classification systématique à faire afin d'évaluer les liens entre le contexte de simulation et les paramètres du modèle simulé.

Spécialisation

Contrairement aux produits commerciaux, d'usage général, ces simulateurs sont dédiés à une application simple: quantification d'un espace d'entrée et voyageur de commerce pour Kohonen, reconnaissance de motifs pour Hopfield, approximation de fonctions pour le Perceptron multicouches, coloriage d'une carte pour le modèle à coopération/compétition et séparation de sources pour le modèle de Hérault-Jutten.

Ce choix est justifié par la difficulté rencontrée par les utilisateurs à dissocier structure de réseau et règle d'adaptation. En premier lieu, on définit la structure (réseau bouclé ou non rebouclé, interactions locales, interactions aléatoires, etc.) puis on choisit parmi les règles d'adaptation admissibles sur cette structure, celle qui convient au problème. Ainsi, en liant l'application à la structure, sans donner le choix à l'utilisateur, on souligne l'effet des règles d'adaptation sur le comportement général du modèle. Pour chaque structure il y aura donc, quand cela est possible, plusieurs règles d'adaptation. Pour le réseau de Hopfield (structure rebouclée) par exemple, il y a trois règles d'adaptation: Hebb, Projection et Delta projection et cela pour une seule application.

Souplesse

La compréhension intuitive qui découle en général des manipulations avec ces simulateurs doit être complétée par une étude théorique qui permet d'intégrer le modèle. A cet effet, on a retenu la solution de la programmation du noyau des algorithmes. Ainsi, tout utilisateur peut programmer par lui-même le corps de l'algorithme sans se préoccuper de la gestion des événements, des fenêtres, des

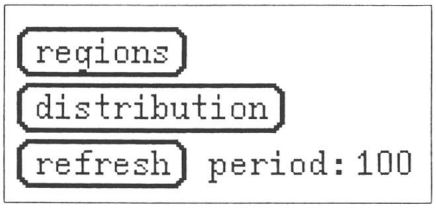


Figure 2 Exemples de boutons

entrées-sorties. Il suffit de remplir une boîte noire qui reçoit et délivre des valeurs à travers des variables déclarées à cet effet. Le simulateur permet ensuite de substituer cette boîte noire au programme standard qui réalise la simulation du modèle. Il est donc possible de vérifier immédiatement l'exactitude du programme écrit et, en cas de mauvais fonctionnement, de modifier sa propre compréhension du modèle.

Cette possibilité est due à la méthode de développement des programmes. On a ainsi choisi de bâtir trois bibliothèques (matricielle, graphique et neuronale) qui sont appelées par le programme qui gère les interactions entre objets du simulateur. Le corps de simulation n'est donc qu'un objet quelconque auquel on délivre un contexte et qui renvoie des valeurs. La description des bibliothèques sera reprise dans le paragraphe sur les outils de développement.

Expérimentation

L'analyse des modèles, par un utilisateur expert, est une fonction essentielle pour la validation d'hypothèses de recherche. L'environnement de simulation présenté est enrichi de fonctionnalités supplémentaires qui permettent une étude quantitative des fonctions des modèles. Le logiciel Mathematica [4] a été retenu comme instrument d'analyse. A cet effet, une interface logicielle appropriée est intégrée dans tous les simulateurs afin d'exporter les résultats vers Mathematica.

Pour simplifier la désignation des objets ou groupes d'objets dont on souhaite exporter les valeurs, il a été décidé d'utiliser encore l'interface graphique, sans langage de commande. Il suffit de cliquer, avec le bouton de droite de la souris, sur l'objet ou le groupe d'objets concerné. Cela génère automatiquement un fichier au format de liste admissible par Mathematica, à partir duquel on peut faire les traitements numériques et les représentations graphiques nécessaires à une analyse quantitative approfondie (fig. 3).

Plus généralement, des mécanismes d'entrées-sorties par fichiers sont mis à disposition de l'utilisateur expert, pour aborder des traitements plus élaborés que ceux proposés sur les cas d'école. Le simulateur de Kohonen, par exemple, permet d'entrer des vec-

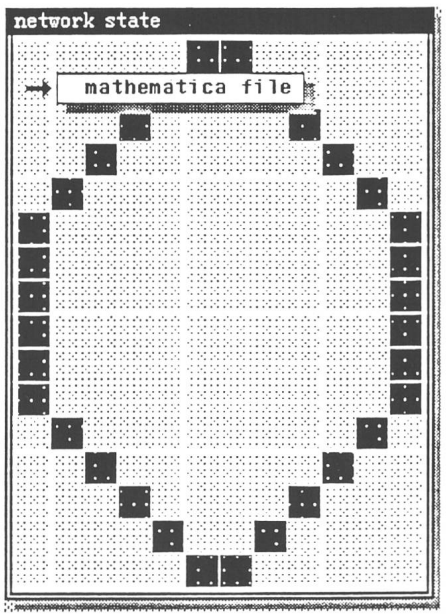


Figure 3 Exemple de sortie vers Mathematica

teurs de données issus de mesures ou de bases de données. Il a été utilisé ainsi pour des applications en traitement d'images, traitement du son et analyses géopolitiques de données économiques. Un tel environnement autorise ainsi une expérimentation immédiate, à condition que les données soient correctement conditionnées (selon les modèles: centrage, normalisation, etc.).

Enfin, l'appel à des librairies matricielles et graphiques permet une intervention rapide sur l'algorithme lui-même pour en étudier les variantes possibles. La préoccupation de l'utilisateur se borne ainsi à une simple modification du code correspondant au modèle, sans avoir à intervenir dans l'environnement de visualisation. A terme, cette notion d'indépendance entre le processus simulé et les outils de visualisation sera étendue pour mettre à disposition de l'utilisateur une boîte à outils qui lui permettra de se construire un environnement de simulation ad hoc.

Outils de développement

Ces outils sont répartis en trois librairies: neuronale, graphique et matricielle. Elles sont orientées objet, afin d'augmenter la souplesse et la facilité de développement. Elles sont écrites en C [5] sur Sun Sparcstation en utilisant le gestionnaire de fenêtres Sunview [6].

Librairie neuronale

D'après notre expérience, le corps des algorithmes permettant l'implantation des modèles de base sur des machines séquentielles est d'expression généralement assez simple (même si le calcul qui en résulte est parfois coûteux). En utilisant un formalisme matriciel, ces modèles de base sont exprimés par des algorithmes de quelques lignes. En revanche, la représentation des données (entrées et résultats) qui sont traitées par ces réseaux de neurones simulés est souvent délicate. D'autre part, plusieurs modèles permettent le traitement d'une même application; par exemple, les réseaux de Kohonen, de Hopfield et Perceptrons multi-couches peuvent tous remplir la fonction de mémoire associative, alors que les problèmes d'optimisation peuvent être confiés à un réseau de Hopfield ou de Kohonen.

Dès lors, il semble naturel, plutôt que de développer un simulateur universel de réseaux de neurones, de s'orienter pour l'enseignement vers la définition d'applications représentatives des différentes classes de problèmes traités par les réseaux de neurones formels. Les différents modèles sont donc vus au travers d'une application spécifique. Il s'agit là d'une approche duale des techniques d'enseignement habituelles où, généralement, après avoir présenté la théorie mathématique justifiant un modèle, on présente succinctement quelques applications de ce modèle.

Le tableau I donne un aperçu des classes de problèmes envisagées, de l'application spécifique choisie pour chaque classe et, dans la version actuelle, du modèle utilisé. Ces différents modèles sont rassemblés dans une librairie neuronale. Bien que dans la version actuelle des logiciels, chaque application soit dédiée à un modèle de réseau, ce concept sera dans l'avenir affiné en dissociant complètement les applications des modèles. On pourra alors utiliser aussi bien un réseau de Kohonen qu'un réseau de Hopfield pour effectuer de la reconnaissance de caractères, pour donner un exemple concret. Il sera également possible de mettre en série plusieurs réseaux (pré-traitement) entre l'entrée et la sortie.

Classe	Application spécifique	Modèle
Mémoire associative	Reconnaissance de motifs	Hopfield
Discrétisation d'espaces, classification	Quantification de distribution particulières	Kohonen
Réglage automatique	Pendule inversé	Perceptron multicouches
Optimisation	Voyageur de commerce	Kohonen
Problèmes np complets	Coloriage de cartes en 4 couleurs	Coopération-Compétition
Analyse en composantes indépendantes	Séparation de signaux temporels mélangés	Hérault & Jutten
Apprentissage supervisé	Approximation de fonction	Perceptron multicouches

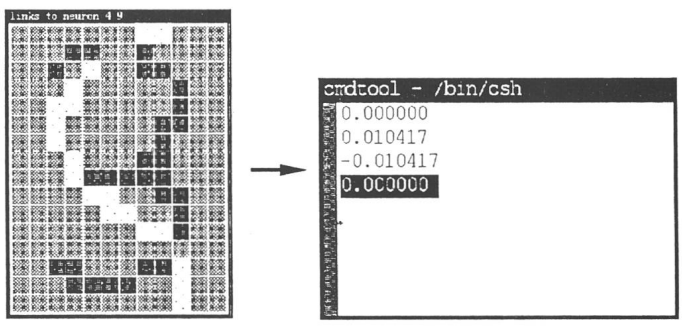


Figure 4 Sortie numérique

Librairie graphique

En poursuivant cette démarche d'unification, on constate qu'il est possible de dégager un nombre restreint de moyens de visualisation qui rendront intelligibles des données a priori abstraites. Il s'agit de:

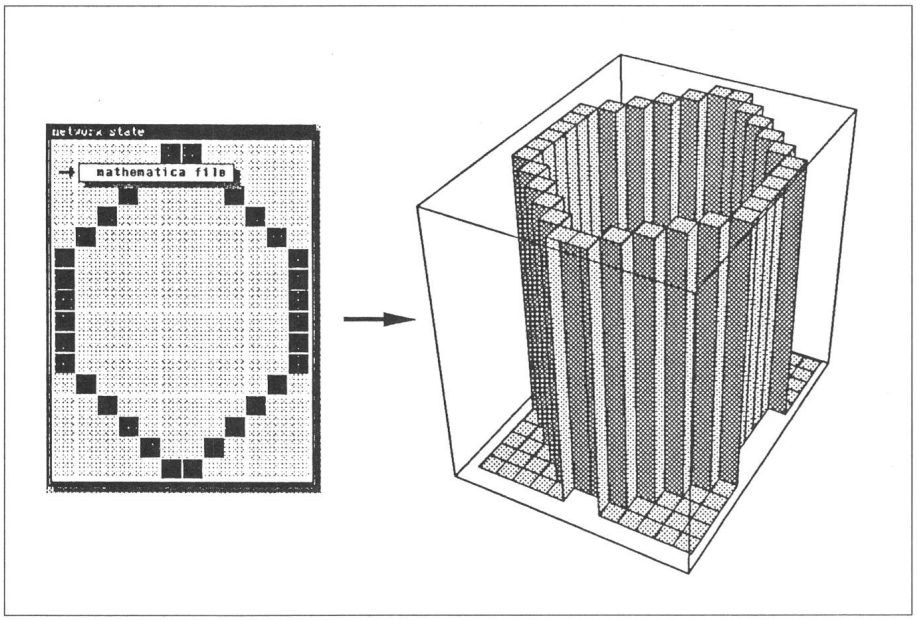


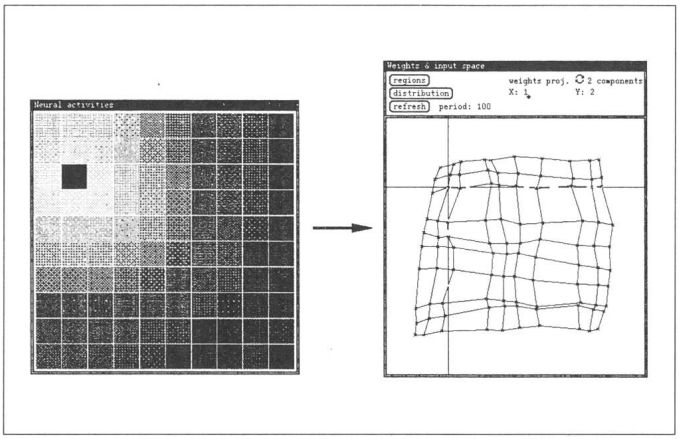
Figure 5 Edition de motifs et sortie vers Mathematica

– Visualisation sous forme de tableaux colorés, utilisables pour la représentation des matrices de poids synaptiques, de l'activité ou de l'état des neurones dans un réseau rectangulaire. Ces tableaux sont interactifs (ils sont sensibles aux actions de la souris), ce qui permet par exemple de connaître la valeur numérique d'une case (fig. 4), d'éditer des motifs (Hopfield, fig. 5), de générer des fichiers Mathematica, de désigner un neurone (Kohonen, fig. 6), d'ouvrir des fenêtres de visualisation de poids (Hérault-Jutten, fig. 7) ou encore d'éditer la palette des couleurs de l'écran du Sun (fig. 8).

– Visualisation de signaux temporels (semblables à des traces d'oscilloscope, fig. 9), utilisables en traitement du signal (dans l'algorithme de Hérault-Jutten), mais aussi pour la représentation de termes d'erreur, etc. Cet objet est également interactif et permet par exemple (dans Hérault-Jutten) de modifier des amplitudes ou de mettre en phase des signaux.

– Visualisation de signaux en X et Y (également comme avec un oscilloscope, fig. 10), pour la représentation de l'évolution temporelle de couples de poids. Ce dernier mode de représentation est lui aussi un objet interactif. Il permet (dans Hérault-Jutten) de fixer les points de mélange et les couples de poids.

Figure 6 Relations entre fenêtres de visualisation



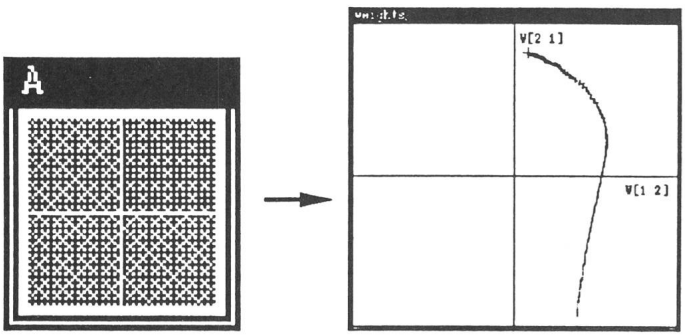


Figure 7 Visualisation des poids

Ces outils et toute l'infrastructure logicielle utilisée pour leur définition sont rassemblés dans la librairie graphique.

Librairie matricielle

Afin d'apporter un maximum de souplesse à la mise en œuvre des algorithmes, ainsi qu'à notre propre expérimentation sur de nouveaux algorithmes, une librairie matricielle gérant tous les problèmes d'utilisation de la mémoire (auto-allocation et garbage collecting) a été développée. Les objets matrice et vecteur sont à la base de la définition de tout algorithme neuronal. La librairie neuronale fait donc un large usage de cette librairie de calcul matriciel. Un avantage sensible de cette démarche est de pouvoir, sans autre effort particulier, tester l'effet de la quantification (c'est-à-dire un nombre de bits limité pour la représentation des données) sur les différents algorithmes. En effet, dans la définition des matrices et vecteurs, on pourra inclure le type et la taille du stockage des composantes (par exemple, composantes en virgule fixe sur 5 bits), ainsi que le type et la taille des opérateurs élémentaires.

Un exemple du codage en C à l'aide de cette librairie d'une partie d'algorithme est donné dans la figure 11 ci-après (il s'agit de l'algorithme de Jacobi utilisé pour trouver les sorties $\boldsymbol{y}$ en fonction des entrées $\boldsymbol{e}$ et de la matrice des poids synaptiques $\boldsymbol{W}$ dans le modèle de Hérault-Jutten). Remarquons que sans la facilité apportée par la prise en charge de la gestion de la mémoire nécessaire pour les différents objets, le codage d'un tel algorithme serait beaucoup plus fastidieux (le programmeur devrait passer par plusieurs tableaux temporaires explicite-

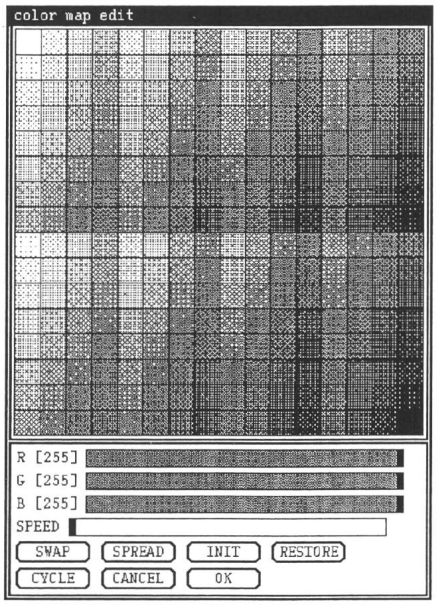


Figure 8 Exemples d'opérations engendrées par l'action de l'utilisateur

ment déclarés et gérer lui-même les allocations mémoire). Le code qui gagne ainsi en lisibilité et en souplesse est plus facilement modifiable.

Passage de paramètres en nombre variable, orientation objet

L'écriture de ces librairies s'inspire en partie des règles de programmation définies dans les langages-objets [7]. Lors de la création d'une instance d'un objet, le programmeur indique les routines que cet objet va utiliser par la suite. Par exemple, lors de la création d'un tableau coloré (pattern), l'adresse d'une routine fournissant un numéro de couleur pour une position dans le tableau est transmise, et c'est cette routine qui sera automatiquement appelée par la suite chaque fois qu'une portion du tableau doit être redessinée (dans le jargon des langages-objets, on dit que la couleur en fonction de la position dans le tableau est une méthode de l'objet «pattern», fournie lors de la création d'instances de cet objet). Dans l'exemple donné ci-dessous, cette routine s'appelle my_color_proc, et la librairie graphique l'utilise sous la forme color = my_color_proc (pattern, x, y), où <x, y> indique la case dont on demande la couleur.

En outre, certaines routines de ces trois librairies tolèrent un nombre variable de paramètres. Cette possibilité, inspirée du format d'appel des routines de création d'instances d'objets graphiques dans Sunview, engendre une grande souplesse dans l'utilisation, mais également dans le développement des librairies. Pour l'utilisation, la souplesse vient du fait que, d'une part, on n'est pas obligé de spécifier la totalité des paramètres (ce qui est fort fastidieux lorsque leur nombre est grand), et que, d'autre part, l'ordre dans lequel ces paramètres sont fournis est sans importance (fig. 12). Pour le développement, ce passage de paramètres par couple <clé, valeur> est également très intéressant, parce qu'il permet l'extension des possibilités d'un objet par l'adjonction de nouveaux paramètres sans avoir besoin de modifier tous les programmes qui utilisent cet objet.

Problèmes liés à la simulation

Cohérence de présentation

Il est souhaitable de présenter sur l'écran un ensemble non-exhaustif des différents paramètres, afin de ne pas déconcerter un utilisateur peu familier du modèle simulé. Les paramètres sont donc regroupés en classes (par exemple paramètres d'initialisation, paramètres d'adaptation, etc.). Lorsque le nombre total des paramètres est trop important, chaque groupe est accessible sur demande dans des fenêtres séparées (ex.: simulateur de réseau de Kohonen). Malgré ce découpage, des actions sur certains paramètres engendrent parfois un changement d'autres paramètres (comme dans Kohonen, où le changement de dimension du réseau va engendrer une réinitialisation des paramètres de contrôle des fonctions alpha et voisinage). De toute manière, qu'ils soient affichés ou non, il est indispensable de maintenir la cohérence la plus rigou-

1) $y_0 = e$
2) $y_n = y_{n-1} - We$

calculer y_{10} est codé:

```
Vector    e, y;
Matrix    W;
...
ASSIGN(y, e);
for (i = 0; i < 10; i++)
ASSIGN(y, sub(y, mult(W, e)));
```

Figure 11 Exemple de codage

reuse entre l'état des différents paramètres et l'état du simulateur (dans certaines conditions, des boutons sont inutilisables et sont donc soit cachés soit en grisé). Cette cohérence doit également être observée dans les différents affichages, pour ne pas troubler la compréhension de l'utilisateur, ce qui implique souvent l'utilisation de listes de dépendance (liste de tous les objets sur lesquels une action doit se répercuter).

Pseudo-parallélisme, vitesse de simulation

Un des facteurs de confort dans l'utilisation de ces simulateurs est de pouvoir changer des paramètres lorsqu'une action est en cours (par exemple dans Kohonen, contrôle manuel du paramètre alpha en cours d'adaptation). Dans l'environnement retenu (Sun Sparcstation et Sunview), la solution paraissant la plus satisfaisante a été de confier la gestion du pseudo-parallélisme au système de fenêtrage (Sunview), lequel dispose de fonctions pour lancer des actions à intervalles réguliers (polling). La philosophie

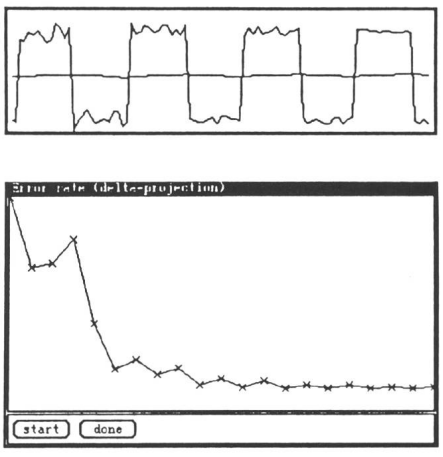


Figure 9 Exemples d'afficheurs de signaux temporels

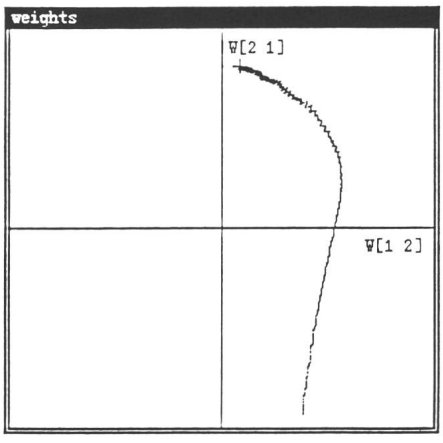


Figure 10 Exemple d'afficheur

a

```
Pattern   my_pat;
...
my_pat =  pattern_create(parent_frame,
                PAT_COLUMNS,          10,
                PAT_LINES,                15,
                PAT_COLOR_PROC,           my_color_proc,
                0);
```

b

```
Pattern   my_pat;
...
my_pat =  pattern_create(parent_frame,
                PAT_SELECT_DOWN_PROC,
          my_notification_proc,
                PAT_X,                500,
                PAT_Y,                300,
                PAT_COLOR_PROC,           my_color_proc,
                PAT_COLUMNS,          10,
                PAT_LINES,                15,
                PAT_BLOCK_MIN_SIZE,   3,
                PAT_BLOCK_SIZE,           10,
                PAT_CLIENT_DATA,          &anything,
                0);
```

Figure 12 **Exemple de codage**
a Création d'un tableau coloré, appelé pattern
b Même exemple, avec plus de spécifications: position x, y, routine de notification en cas de sélection par l'utilisateur, contraintes sur la taille des carrés colorés, etc.

d'emploi de cette fonctionnalité est la même que pour les actions déclenchées par l'utilisateur (boutons, etc.): une procédure définie par l'utilisateur est appelée chaque fois que c'est nécessaire (en l'occurrence à intervalles de temps définis dans le programme). Comme il s'agit de pseudo-parallélisme, le contrôle n'est rendu au gestionnaire de fenêtre qu'après l'exécution complète de la procédure. Pour cette raison, il n'est pas acceptable que ces procédures répétitives aient un temps d'exécution trop long. En effet, lorsque c'est le cas (rencontré dans certaines versions en cours de développement), le programme perd son interactivité et toutes les actions de l'utilisateur (emploi de boutons, modifications de paramètres) sont retardées d'une façon inadmissible (plusieurs secondes). Il a donc fallu découper toutes les actions trop longues en plusieurs petites pouvant être effectuées en plusieurs étapes. De même, on a souvent choisi des algorithmes simplifiés pour garantir une vitesse de simulation acceptable.

Dans Hérault-Jutten, les signaux de sorties sont calculés par itérations de l'algorithme de Jacobi, au lieu de calculer l'inverse de la matrice des poids synaptiques qui permet d'obtenir la solution analytique. Si l'on considère d'ailleurs un réseau de Hérault-Jutten physique (électronique ou biologique), c'est en fait cet algorithme itératif de Jacobi qui sera implicitement employé (les opérateurs n'ayant pas une bande passante infinie). Dans le simulateur de Kohonen, c'est l'algorithme du «winner take all» qui est utilisé, au lieu de l'algorithme original et fortement coûteux en temps de calcul sur machine séquentielle (émergence d'une bulle d'activité par relaxation d'un réseau à connexions latérales). La vraisemblance biologique de l'algorithme simplifié de Kohonen est, contrairement à l'algorithme simplifié de Hérault-Jutten, relativement faible.

Domaine de variation des paramètres

Le choix de ce domaine de variation numérique des paramètres, ainsi que leur valeur par défaut (valeur à l'initialisation) est un point délicat. En effet, ce sujet est relativement peu abordé dans les présentations théoriques des modèles de réseaux de neurones, et de nombreux tâtonnements ont été nécessaires pour fixer les valeurs adéquates. Ces simulateurs constituent un apport intéressant pour les utilisateurs désirant exploiter certains modèles, et qui pourront ainsi appréhender les ordres de grandeur des valeurs numériques utilisées.

Conclusion

Dans cet article, on a présenté une méthodologie de développement de simulateurs de réseaux de neurones. L'approche retenue, qui privilégie l'interactivité et la souplesse, présente de nombreux avantages pour la formation et l'expérimentation rapide d'hypothèses algorithmiques appliquées à des cas concrets.

Les simulateurs développés ont été exploités pour l'enseignement destiné aux années terminales des écoles d'ingénieurs, ainsi que pour la formation industrielle dans le cadre d'un projet Comett. De futurs développements sont à l'étude afin d'enrichir la famille de modèles simulés (Algorithmes à structure évolutive, Time-Delay Neural Networks, etc.) et d'étendre la notion d'outil de visualisation pour la rendre, à terme, indépendante des modules de simulation.

Remerciements

Les auteurs tiennent à remercier L. Tettoni et G. Baratoff pour le développement des librairies et des simulateurs. Le développement des librairies a été supporté par le Fonds National Suisse de la Recherche Scientifique, dans le cadre du projet Esprit-BRA Nerves n° 3049. Le développement des simulateurs a été partiellement supporté par le projet Comett Neural n° 90/1/3116/Cb.

Bibliographie

[1] *F. Blayo*: Cours de base en réseaux de neurones artificiels, EPF-Lausanne, 1991.
[2] *P. Demartines*: Manuels d'utilisation du simulateur de réseau de Kohonen, du simulateur de réseau de Hopfield, du simulateur de réseau de Hérault-Jutten, rapports internes, novembre 1990.
[3] *L. Tettoni*: Manuel d'utilisation du simulateur de Perceptron avec rétro-propagation de gradient, rapport interne, novembre 1990.
[4] *S. Wolfram*: Mathematica, A system for Doing Mathematics by Computer, Addison-Wesley, 1988.
[5] *B.W. Kernighan, D.M. Ritchie*: The C Programming Language, Prentice Hall, 1988.
[6] *Sun microsystems*: Sunview, Sun View 1 Programmer's guide, 1988.
[7] *B. Meyer*: Object-oriented Software Construction, Prentice Hall, C.A.R. Hoare ed., 1988.