Utilisation d'un réseau de neurones artificiels comme fonction d'évaluation d'un jeu

Didier Müller, EPFL-DMA, CH-1015 Lausanne
Novembre 1992

Résumé

Dans ce travail, nous présentons une méthode utilisant un réseau de neurones multicouche pour remplacer une fonction d'évaluation d'un jeu. Le jeu choisi est le push-over, mais la méthode s'applique à la plupart des jeux. Nous montrons comment créer une base de données de positions étiquetées et comment l'utiliser pour l'apprentissage. Les résultats obtenus sont séduisants: en effet, le réseau joue mieux que son maître; cependant, les réseaux multicouches ne sont pas sans défauts.

Mots Clés

Théorie des jeux, fonction d'évaluation, push-over, réseaux de neurones, classification.

---

1. Introduction

Rien ne nous plaît que le combat, mais non pas la victoire (...) Ainsi dans le jeu, ainsi dans la recherche de la vérité (...) De même, dans les passions (...) Nous ne cherchons jamais les choses, mais la recherche des choses.
PASCAL, Pensées, II, 135.

1.1. L'étude des jeux

La théorie des jeux a été lancée vers 1920 par Émile Borel, mais il fallut attendre 1926 pour que John von Neumann démontre le théorème du minimax autour duquel cette théorie s'est développée. Von Neumann publia avec l'économiste Oscar Morgenstern, en 1944, l'ouvrage désormais classique *Theory of games and economic behavior.* Depuis, la théorie des jeux s'est considérablement développée, mêlant l'algèbre, la géométrie, la recherche opérationnelle, la théorie des ensembles et la topologie. Elle s'applique aussi bien à l'économie qu'aux affaires, à la guerre qu'à la politique.

Naturellement, cette théorie s'applique aussi aux jeux tels que tout un chacun les conçoit: échecs, dames, backgammon, etc. Une question qui revient souvent est: pourquoi les mathématiciens s'intéressent-ils à un sujet aussi futile que les jeux ?

Les spécialistes de l'intelligence artificielle furent parmi les premiers à s'intéresser aux jeux. En effet, un jeu est composé d'un nombre limité d'éléments et d'un ensemble réduit de règles précises; son but est également clair. Un jeu forme ainsi un petit univers plus facile à maîtriser

que certains problèmes "réels" semblables, mais quand même suffisamment complexe pour que l'on puisse faire intervenir des notions typiquement humaines comme la réflexion et le raisonnement.

Les échecs en particulier ont très intéressé les mathématiciens. L'un d'eux a dit : « Les échecs sont la drosophile de l'intelligence artificielle », faisant référence à la mouche que Morgan a utilisée pour ses recherches en génétique. Cependant, les programmes actuellement performants font surtout appel à la *force brutale* (i.e. regarder tous les coups possibles) et à la puissance de calcul plutôt qu'au raisonnement.

En 1950, Claude Shannon publia l'article mythique *Programming a computer for playing chess* [Sha50] qui influença tous les programmes d'échecs jusqu'à nos jours. Il y explique notamment comment construire une *fonction d'évaluation*, qui est l'un des deux concepts fondamentaux dans tout programme de jeu, l'autre étant *l'algorithme de parcours de l'arbre du jeu.*

On représente en effet un jeu par un arbre où un nœud correspond à une configuration possible du jeu et un arc représente une transition légale entre deux configurations. Dans les jeux à deux joueurs, les nœuds d'un niveau n (impair) de l'arbre correspondent à toutes les positions possibles pour le premier joueur après n coups ; idem avec les nœuds des niveaux pairs pour le deuxième joueur. On se rend vite compte que même pour des jeux très simples, comme le tic-tac-toe (morpion), ces arbres deviennent rapidement gigantesques au fur et à mesure que l'on descend profond dans les niveaux. Il a donc fallu inventer des algorithmes performants et rapides qui évitent d'explorer l'arbre dans sa totalité. Nous ne nous attarderons pas davantage sur les arbres de jeux, le lecteur intéressé pourra se reporter à la revue *Artificial Intelligence*, où de nombreux algorithmes de parcours sont décrits, notamment le plus connu d'entre eux, l'algorithme alpha-bêta.

Si de nos jours on maîtrise relativement bien l'exploration des arbres de jeux, il en va tout autrement des fonctions d'évaluation. Comme leur nom l'indique, ces fonctions ont pour but d'évaluer une position : un nombre très grand sera l'indice d'une excellente position, tandis qu'un nombre très petit (généralement négatif, mais cela dépend de l'échelle choisie) traduira une situation catastrophique.

Comment obtient-on une « bonne » fonction, c'est-à-dire une fonction qui traduit fidèlement la réalité? Une fonction d'évaluation f est généralement une somme pondérée de la forme :

$$f(P) = a_1 \cdot c_1(P) + a_2 \cdot c_2(P) + \ldots + a_n \cdot c_n(P) \qquad (1.1)$$

où les a_i sont des *poids* (a_i R) et les c_i des *caractéristiques* d'une position P (par exemple, aux échecs, le nombre de pions doublés, les pions occupant le centre, le nombre de fous pris à l'adversaire, le nombre de fous perdus, etc.). Trouver les caractéristiques d'un jeu n'est généralement pas chose facile, et les programmeurs ont souvent recours à des experts pour les y aider. Quant aux poids, qui traduisent le fait qu'une caractéristique est plus importante qu'une autre, leur ajustement fait plus appel à l'empirisme et à l'expérience du programmeur qu'à une méthode rigoureuse.

Le but de ce travail est de remplacer la fonction d'évaluation d'un jeu par un réseau de neurones artificiels (RNA).

1.2. Brefs rappels sur les RNA

Les systèmes évolués mettent en jeu une quantité considérable de cellules nerveuses, liées par des connexions synaptiques en quantité gigantesque. Ils sont le siège d'une intense activité électrochimique et les capacités d'adaptation et d'action qu'ils démontrent en font un sujet d'étude digne d'intérêt. Ainsi, le principe du connexionnisme repose sur le fait qu'une activité collective d'unités simples qui s'influencent mutuellement peut engendrer un comportement varié et complexe.

Depuis les années 1960, plusieurs modèles ont été proposés sur ce principe. Ces modèles font généralement appel à deux phases de calcul : une phase d'apprentissage qui sert à établir la fonction que doit réaliser le réseau à partir d'une base d'exemples, puis une phase de reconnaissance qui exploite cette fonction sur un ensemble de données éventuellement plus vaste. Ces deux notions représentent deux aspects des réseaux connexionnistes : l'*apprentissage* par l'exemple et la capacité de *généralisation.*

Le modèle de McCulloch et Pitts est certainement l'un des plus anciens et constitue l'élément de base des réseaux connexionnistes. Il décrit un neurone comme un élément sommateur des entrées qui lui parviennent à travers des pondérations représentées par des éléments synaptiques (voir fig. 1.1).

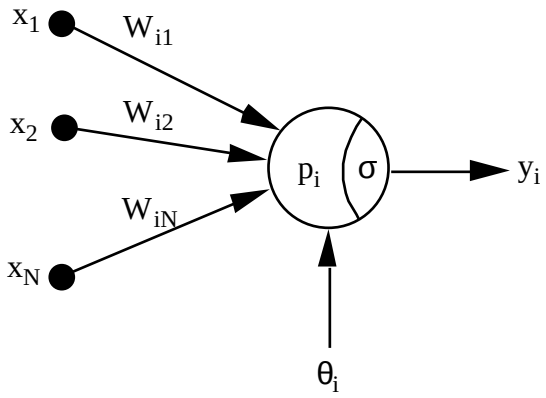

Fig. 1.1 : *modèle général du neurone formel*

Si nous appelons :

x_j un stimulus d'entrée,

W_{ij} la valeur du poids synaptique entre l'entrée j et le neurone i,

θ_i, une valeur appelée seuil du neurone i,

alors le potentiel du neurone i est égal à :

$$p_i = \sum_{j=1}^{N} W_{ij} x_j \qquad (1.2)$$

On calcule la sortie effective du neurone par exemple en comparant la valeur du potentiel à un seuil. Si la valeur dépasse le seuil, le neurone délivre une sortie +1, sinon, il délivre une sortie égale à 0.

$$y_i = \sigma(p_i) = \begin{cases} 1 : p_i > \theta_i \\ 0 : p_i \le \theta_i \end{cases} \qquad (1.3)$$

La fonction de transition σ ci-dessus, appelée fonction seuil, délivre des valeurs binaires.

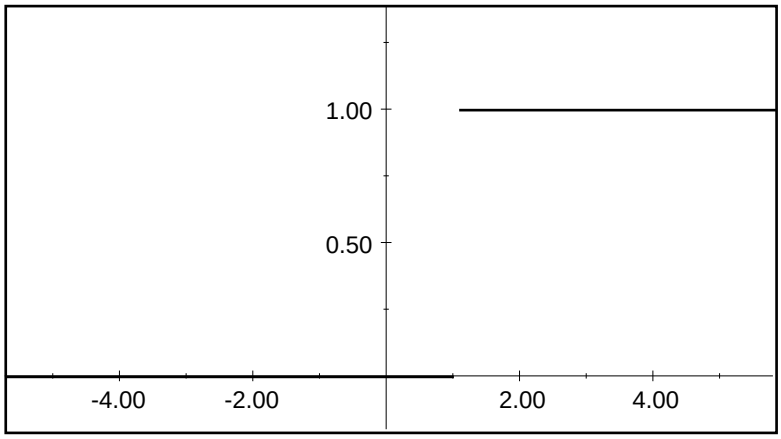

Fig. 1.2 : *fonction seuil (ici avec θ=1.0)*

On peut utiliser d'autres fonctions pour σ, par exemple la fonction sigmoïde. On a alors :

$$y_i = \sigma(p_i) = \frac{1}{1 + e^{\frac{-(p_i - \theta)}{T}}} \qquad (1.4)$$

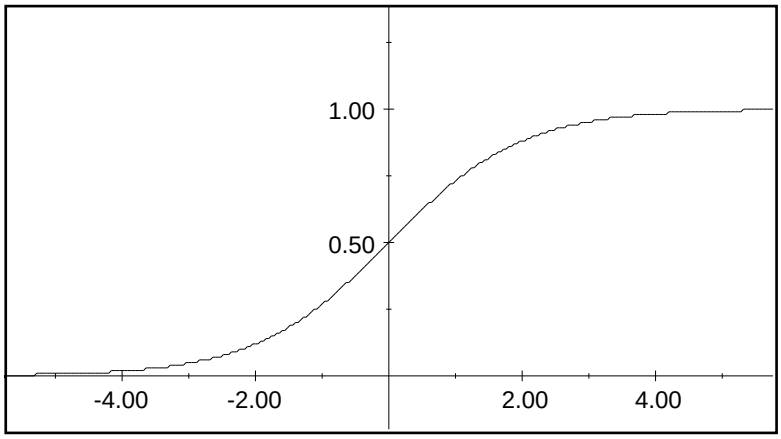

Fig. 1.3 : *fonction sigmoïde*

En connectant de tels neurones entre eux d'une certaine manière, on forme des réseaux d'un certain type.

Un type de réseau fréquemment utilisé est le *réseau multicouche*. Un exemple d'un tel réseau est donné à la fig. 1.4. Tous les neurones de la couche n sont connectés à tous les neurones de la couche n+1 ; il n'y a pas de d'autres connexions. Les couches situées entre la couche d'entrée et la couche de sortie sont appelées les *couches cachées*.

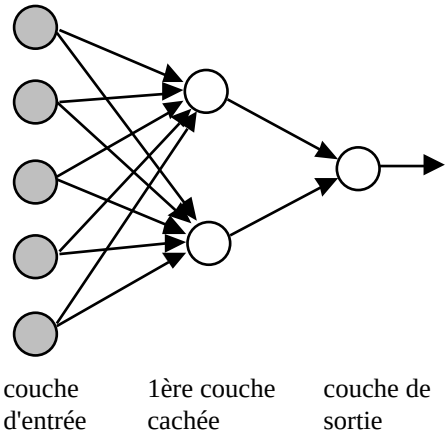

Fig. 1.4 : *réseau multicouche avec une seule couche cachée*

1.3. Travaux effectués dans le domaine des jeux et de l'apprentissage

Un des premiers articles concernant l'apprentissage d'un jeu est dû à A. Samuel [Sam59]. Ce travail a influencé de nombreuses autres recherches. Voici une liste non exhaustive d'articles concernant l'apprentissage des jeux : [Sam67], [Elc67], [Kof68], [Gri74], [DeJ87], [Lee88], [Put89], [Sej89], [Cor90].

Les réseaux de neurones ont été jusqu'à présent très peu utilisés dans la théorie des jeux. En fait, nous ne connaissons que deux articles dans ce domaine : [Sej89] et [Cor90].

Dans [Sej89], traitant du backgammon, le concept suivi par Sejnowski et Tesauro était d'entraîner le réseau à sélectionner des coups (parmi un ensemble prédéterminé de coups

permis), plutôt qu'à les générer. Équipé d'un préprocesseur qui génère les coups légaux étant donnés une position et un jet de dé, le réseau était entraîné à évaluer tous ces coups, leur donnant une valeur entre -100 et +100 (-100 étant le pire coup possible, +100 le meilleur). La position et le changement causé par le coup étaient codés sur 459 neurones d'entrée. Le réseau multicouche avait un seul neurone de sortie qui représentait l'évaluation. De nombreuses architectures furent testées ; le réseau le plus performant avait deux couches cachées de 24 neurones chacune. Cependant, les différences de performance, qu'il y ait moins ou même pas de couches cachées, n'étaient pas très grandes.

La base de données pour l'apprentissage contenait environ 3000 positions de jeu (sur un total estimé à environ 10), avec un jet de dé particulier pour chaque position, et un ensemble de coups légaux. Un petit sous-ensemble de ces coups a été estimé par un expert humain ; pour la grande majorité des coups, l'évaluation a été tirée au sort dans l'intervalle [-65,35]. Cette technique quelque peu étrange a été introduite à cause du grand nombre de coups possibles. Le réseau fut d'abord entraîné sur les coups explicitement évalués, mais ensuite aussi sur des coups aléatoirement évalués, afin d'éviter un biais exagéré dans l'ensemble d'apprentissage.

Les performances du réseau entraîné furent mesurées de différentes manières, la plus intéressante d'entre elles étant son affrontement avec le programme commercial GAMMONTOOL[1]. Dans sa meilleure configuration, le réseau battit GAMMONTOOL dans 60% des parties, ce qui constitue une réussite remarquable.

Dans [Cor90], traitant du jeu de l'awale, les auteurs utilisent un réseau multicouche avec apprentissage par rétropropagation du gradient et un programme « classique » comme expert. En résumé, ils procèdent ainsi : ils créent d'abord une base d'exemples où ils font correspondre à une situation donnée le coup que jouerait le programme classique. Après apprentissage, ils obtiennent un premier réseau, qui joue assez mal.

Ensuite, ils font jouer le réseau obtenu ci-dessus contre un adversaire donné plusieurs parties. Si le réseau a gagné la partie, ils bonifient les configurations rencontrées lors de la partie (réponse attendue : 1) ; si le réseau a perdu, ils pénalisent (réponse attendue : -1). L'adversaire est le réseau lui-même, auquel a été ajouté un léger bruit gaussien centré de faible variance à la sortie.

Avec la nouvelle base d'exemples obtenue, ils réalisent l'apprentissage sur un nouveau réseau de neurones multicouche ayant un seul neurone en sortie : l'évaluation de la solution.

La situation est donnée par le nombre de graines dans chacun des 12 trous du jeu et par le nombre de graines prises. Ce sont les seules indications dont le réseau dispose.

Le réseau a 180 neurones en entrée. Le nombre de graines dans chaque trou est codé sur 13 neurones. Le nombre de graines prises est codé sur 12 neurones. On a donc bien (12 trous) * (13 neurones) + (2 joueurs) * (12 neurones) = 180 neurones.

Après apprentissage, le réseau joue mieux que le programme qui fut son « maître », en réussissant notamment la prouesse que gagner plus d'une partie sur deux contre un niveau du programme anticipant 7 coups, alors que lui n'en anticipe qu'un seul !

2. Jeu du push-over

Ce jeu est décrit dans *Jeux & Stratégie no 2* (Avril-mai 1980). Nous l'avons choisi parce qu'il est relativement simple et que nous avons déjà à disposition un programme qui y joue « bien » (en tout cas aussi bien sinon mieux que l'auteur et les personnes qui l'ont testé !).

2.1. Règles du jeu

Le jeu du push-over se joue sur un damier 4x4 (ou 5x5). Le but est d'aligner 4 (5) pions de sa couleur en vertical, en horizontal ou en diagonal. On entre les pions par les bords du damier et on pousse le pion qui éventuellement s'y trouve d'une case dans la direction du mouvement du nouveau pion ; tous les pions qui gênent sont décalés eux aussi d'une case et éventuellement

[1] Un produit de Sun Microsystems Inc.

sortis du damier. Il est interdit pour un joueur de jouer un coup qui conduirait à une configuration identique à celle consécutive à son coup précédent. Les o commencent.

La fig. 2.1 montre un milieu de partie.

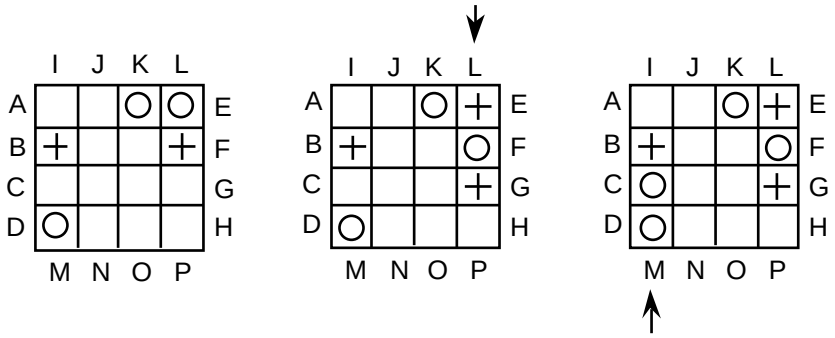

Fig. 2.*1 : 2 coups d'un milieu de partie*

Nous avons ajouté deux règles supplémentaires :

1. Le nombre de coups pour une partie est limité à 50 sur un damier 4x4; si cette limite est dépassée, la partie est déclarée nulle. En effet, il est possible de créer des cycles de coups comme le montre la fig. 2.2. Cette règle est la plus simple pour éviter des parties infinies.
2. Il est possible qu'à la fin de la partie, les deux joueurs soient en position gagnante. Dans ce cas, c'est le dernier à avoir joué qui a gagné.

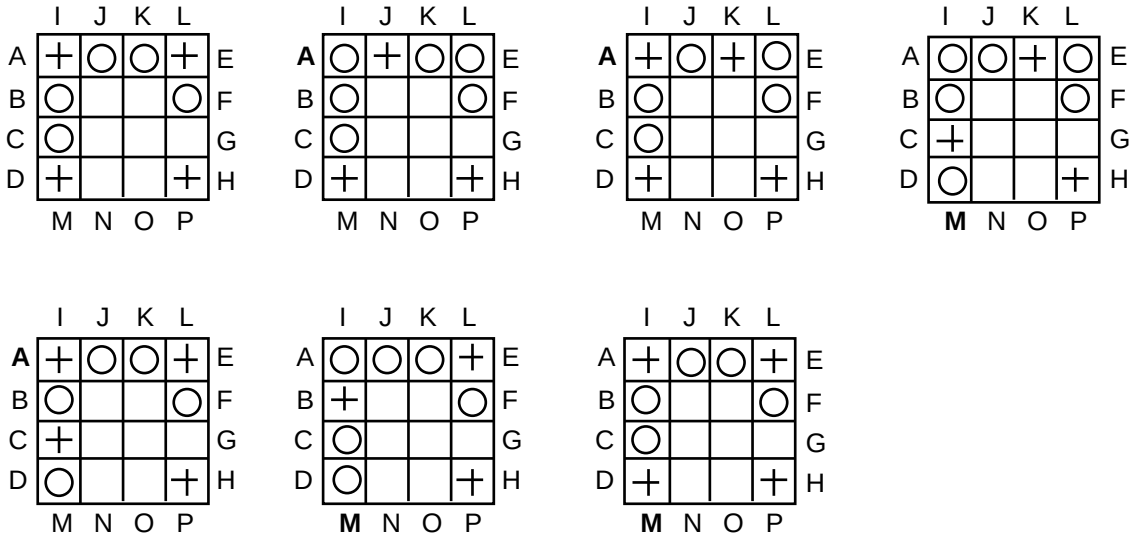

Fig. 2.2 : *cycle de longueur 6 (la lettre en gras indique le coup qui vient d'être joué)*

Remarque : nous n'avons pas pu retrouver les règles originales du jeu et nous nous sommes donc fiés aux règles décrites dans [LeN82]. Nous ne savons pas si les deux cas ci-dessus avaient été prévus.

2.2. Premier programme qui joue au push-over

À notre connaissance, le premier article s'intéressant à la programmation de stratégies pour ce jeu fut [LeN82]. Les procédures principales d'un programme pour ce jeu y sont écrites et elles ont été reprises telles quelles. Cependant, la fonction d'évaluation que les auteurs proposaient était mauvaise ; en effet, si l'on appliquait leur fonction à une position, puis que l'on tournait le damier d'un demi-tour et que l'on réappliquait cette fonction, on obtenait parfois deux résultats

différents pour en fait deux configurations identiques. Nous avons donc réécrit la procédure en cause, dont voici le nouveau code :

```
procedure setval (i,j:integer; piece:char; board:matrix;
                  var value, weight: integer);
  (* tient a jour value et weight *)
begin
  if board[i,j] = piece then
    begin
      if weight = 0 then
        weight := 1;
      weight := weight * 5
    end
  else
    begin
      if board[i,j] = ' ' then  (* case vide *)
        weight := 1
      else                      (* case occupee par l'adversaire *)
        weight := 0
    end;
  value := value + weight
end (* setval *);
```

Cette procédure est appelée par une autre qui parcourt séquentiellement toutes les cases par colonne, par ligne et par diagonal : `i`, `j` sont les coordonnées d'une case, `piece` est le type de pion (+ ou o) sur cette case, `board` est la configuration courante, `value` et `weight` sont les valeurs renvoyées. Avant de parcourir chacune des 10 rangées, on initialise `value` à 0 et `weight` à 1. On somme ensuite les 10 résultats obtenus pour obtenir l'évaluation de la position.

Nous avons aussi légèrement modifié la fonction d'évaluation elle-même (nommée `compute` dans [LeN82]) pour tenir compte du fait qu'il est mauvais d'avoir un groupe de trois pions alignés verticalement ou horizontalement suivi d'un pion ennemi. En effet, l'adversaire peut repousser facilement l'attaque et même prendre l'avantage sur cette rangée comme on peut le voir sur la fig. 2.3.

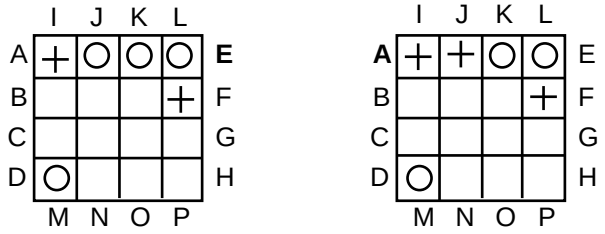

Fig. 2.3 : *les o ont perdu l'initiative sur la première ligne (ils n'ont pas le droit de jouer en E, car on retrouverait la position de gauche).*

Nous reportons le lecteur à [LeN82] pour plus de détails sur ce programme, car nous n'avons mentionné ici que les points essentiels pour la lecture de ce rapport.

3. Démarche

Pour entraîner le réseau de neurones, on utilisera une base de données de positions, où chaque position sera étiquetée gagnante ou perdante. Pour construire cette base de données, on a fait jouer le programme décrit au chapitre 2 contre lui-même 3000 parties. Pour chaque partie, le gagnant a été mémorisé, puis toutes les positions de la partie où le gagnant jouait ont été étiquetées gagnantes, les autres perdantes. Cette méthode est simple, mais il y a un sérieux problème : une position gagnante pourrait devenir perdante à la suite d'un mauvais coup

ultérieur, et serait donc mal étiquetée. Une telle méthode nécessite donc un expert fiable, ce que nous supposons être le cas.

La fig. 3.1 schématise le processus suivi.

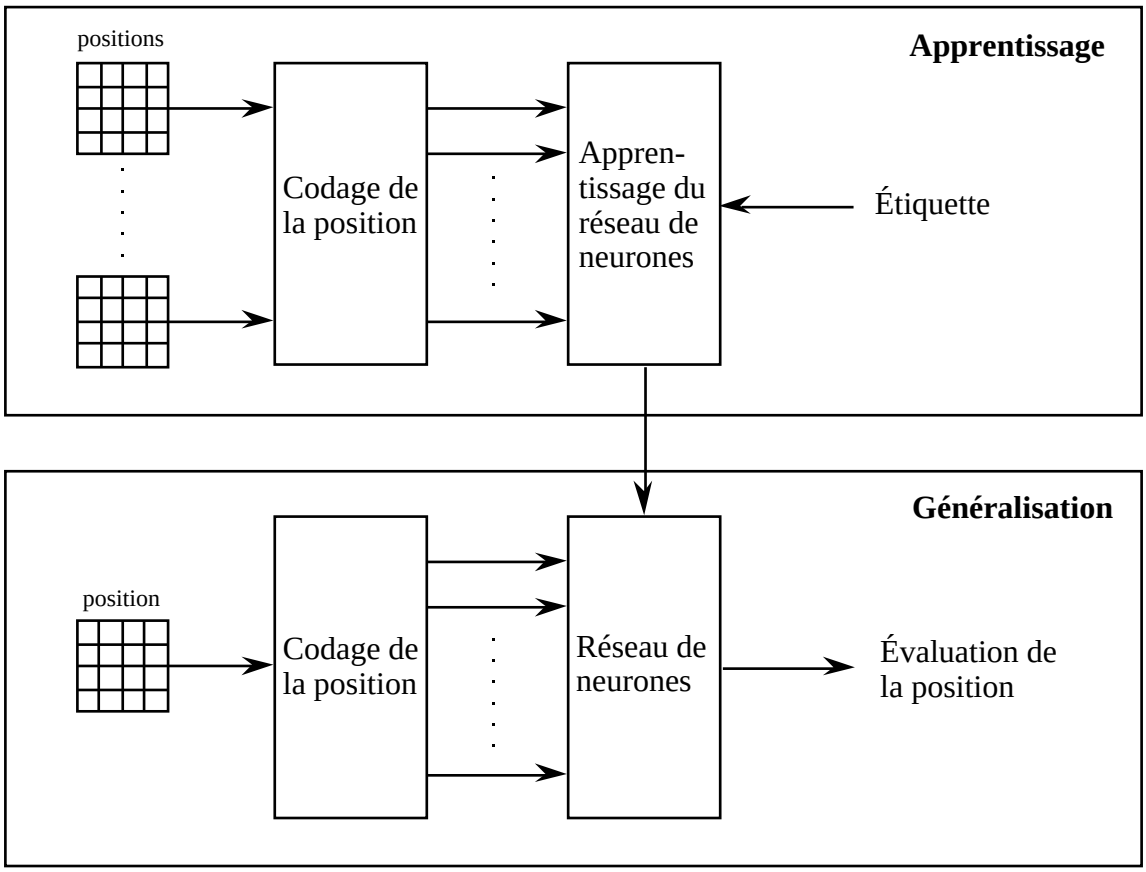

Fig. 3.1 : *démarche*

3.1. Création d'une base de données des positions (BDP)

Nous avons fait jouer notre programme de push-over 3000 parties contre lui-même, les 6 premiers coups de la partie (3 pour chaque couleur) étant joués au hasard, afin d'éviter trop de parties identiques. Sur ces 3000 parties, 103 ont été nulles, les o (premier joueur) ont gagné 64.4% des autres parties et les + 35.6%. 52'590 positions, pas forcément toutes différentes, ont été enregistrées dans la base de données. Une borne supérieure du nombre de positions possibles est 3^{16} = 43'046'721. Si l'on tient compte des symétries et des rotations du damier pour éliminer des positions équivalentes, on peut diviser ce nombre par huit. On obtient alors : 5'380'840 positions comme borne supérieure.

Le base de données a le format suivant (chaque ligne représente une position étiquetée) :

```
+ooo...+....o...Eo -0.400
++oo...+....o...A+  0.466
```

Cette base de données contient les deux positions de la fig. 2.3. Les 16 premiers caractères représentent la position du damier ligne par ligne (« . » = case vide), la lettre suivante le dernier coup joué, le dernier caractère dit qui vient de jouer. Le nombre associé à cette chaîne de caractères donne une estimation de la position. Ce nombre, appelons-le e, est calculé ainsi :

$$e = \frac{j}{t} \qquad (2.1)$$

où *j* est le nombre de coups joués jusque-là et *t* la longueur de la partie.

Reprenons l'exemple de la fig. 2.3. Supposons que la partie a duré 15 coups et que la position de gauche résulte du 6e coup. Disons que les ronds ont perdu la partie. L'estimation de cette position est - 6/15 = - 0.4. Cette manière de faire nous semble plus réaliste que de simplement étiqueter gagnant ou perdant. En effet, plus on se rapproche de la fin de la partie, plus l'étiquetage est censé être correct. De plus, cela relativise un peu le fait que l'expert doive être fiable.

L'estimation de la position forme ce que nous avons appelé *l'étiquette.*

4. Utilisation d'un réseau de neurones

Dans ce chapitre, nous décrivons le réseau de neurones, la méthode et les échantillons d'apprentissage, ainsi que les résultats obtenus sous diverses conditions.

4.1. Idée générale

À partir de la base de données des positions (BDP), nous créons une base de données des caractéristiques (BDC). Pour chaque ligne de la BDP, on crée une ligne de la BDC en procédant ainsi :

a. On inverse si nécessaire les rôles des o et des + de façon à ce que, pour chaque position, ce soient toujours comme si les o venaient de jouer. Un o sera représenté dans la BDC par la valeur 1.0, un + par la valeur 0.0 et une case vide par la valeur 0.5.
b. On recopie l'estimation de l'évaluation de la position.

C'est à partir de cette BDC que l'on va effectuer l'apprentissage du réseau, l'idée étant que le réseau associe à une position une valeur (l'estimation de la position).

4.2. Description du réseau de neurones

Nous avons utilisé un réseau multicouche avec 16 unités d'entrée et 1 neurone en sortie. Les entrées correspondent aux états des cases de la position donnée. Les entrées peuvent donc valoir 0, 0.5, ou 1. La valeur donnée par le neurone de sortie est un nombre réel pouvant prendre toutes les valeurs comprises dans l'intervalle [0,1]. L'échelle est la suivante : 0.5 correspond à une position nulle, 0 à une position perdue et 1 à une position gagnante.

La fonction de transition utilisée pour chaque neurone est la sigmoïde donnée par la formule (1.4).

Dans les réseaux multicouches, le grand problème est de déterminer le nombre de couches cachées et le nombre de neurones pour chacune de ces couches, afin d'obtenir le meilleur rapport précision/rapidité. En effet, un grand nombre de neurones augmente exagérément le temps de calcul, mais donne généralement (pas toujours !) de meilleurs résultats. Il n'existe actuellement pas de méthodes pour trouver la configuration optimale. Nous avons donc essayé plusieurs réseaux et gardé le meilleur.

La configuration retenue est un réseau multicouche avec une seule couche cachée de 16 neurones.

La méthode d'apprentissage est une méthode quasi-newtonienne connue sous le nom de BFGS. Cet algorithme a été développé indépendamment par Broyden, Fletcher, Goldfarb et Shanno en 1970. Elle est décrite entre autres dans [Min83] pp. 121-123. Nous avons effectué l'apprentissage avec 3000 positions.

Nous avons utilisé le logiciel du domaine public GRADSIM [Res90] sur des stations Silicon Graphics pour réaliser l'apprentissage du réseau.

4.3. Choix de l'échantillon d'apprentissage

Une autre difficulté est de choisir les positions que l'on va faire apprendre au réseau. On a constaté en effet que si l'on montrait toutes les positions (du début d'une partie jusqu'à la fin),

le réseau s'embrouillait et pensait, après apprentissage, que toutes les positions étaient gagnantes.

Après plusieurs tentatives décevantes, on a trouvé la « formule magique » : on montre au réseau les positions ayant une évaluation supérieure à 0.5 en valeur absolue, et l'on fait correspondre à cette position soit gagnant (1), soit perdant (0), suivant le signe de l'évaluation.

4.4. Premiers résultats obtenus

On a fait jouer le réseau de neurones contre le programme classique afin de tester s'il joue mieux que ce dernier, ce qui est notre but. Nous avons donc organisé des tournois, chaque tournoi comptant 1000 parties. Dans chaque partie, les 4 premiers coups furent joués au hasard, afin d'éviter trop de parties identiques.

Dans les tableaux de résultats ci-après, on indique le pourcentage de gain des o (rappelons que les o commencent). La signification des symboles est la suivante :

r : réseau, e : expert, 1 : un niveau de recherche dans l'arbre du jeu, 2 : deux niveaux.

o \ +	r 1	r 2	e 1	e 2
r 1	60.6	42.1	50.3	28.5
r 2	74.7	56.4	41.4	41.5
e 1	64.0	66.3	73.4	7.8
e 2	87.5	83.1	97.4	61.8

Tableau 4.1: *pourcentage de gains des o*

On constate que le réseau joue de manière correcte, sans plus. Il est encore plus faible que l'expert dans tous les cas, sauf r1 e1.

4.5. Correction d'une certaine « myopie »

Ces premiers résultats sont présentés sous forme de chiffres. Or, quand on regarde d'un peu plus près les parties qui se sont jouées, on constate que le réseau est un peu "myope", dans le sens où il a parfois du mal à reconnaître une situation gagnante, i.e. une situation où 4 pions identiques sont alignés sur une rangée. Il lui arrive ainsi de rater un coup victorieux. En l'aidant à reconnaître ces situations par un simple test supplémentaire dans le programme, on arrive à améliorer de manière spectaculaire les performances, comme le montre le tableau 4.2. Pour différencier le réseau tel quel du réseau aidé, nous désignerons ce dernier par « r* ».

o \ +	r* 1	r* 2	e 1	e 2
r* 1	73.1	52.4	68.5	47.8
r* 2	88.9	72.6	93.3	83.7
e 1	62.8	24.2	73.4	7.8
e 2	81.8	45.7	97.4	61.8

Tableau 4.2 : *pourcentage de gains des o avec le réseau « aidé »*

On voit qu'à niveau de recherche égal, le réseau est toujours meilleur que l'expert, sauf quand il joue en second avec 1 niveau de recherche (e1 r*1). Comme l'expert, le réseau jouant avec les + semble défavorisé.

Ainsi, on a pu trouver une évaluation de la position meilleure que celle de l'expert à partir de renseignements (l'étiquette) pourtant donnés par la fonction d'évaluation de l'expert. On aurait pu s'attendre à ce que le réseau joue au mieux aussi bien que l'expert, mais en fait, l'élève a dépassé le maître.

4.6. Influence de la BDP sur les résultats

Dans ce paragraphe, nous allons tester l'influence du contenu de la base de données des positions sur les performances des o et des +.

En effet, nous avions remarqué que les o semblaient favorisés à ce jeu, ce qui se traduisait dans la BDP par environ 65% de positions provenant de parties gagnées par les o contre 35% gagnées par les +. Ceci a peut-être une influence sur les résultats.

Pour vérifier cela, nous avons effectué plusieurs apprentissages avec des BDP différentes.

Dans la première BDP, la moitié des positions résultent de parties gagnées par les o, et l'autre moitié de parties gagnées par les +. Les résultats sont exposés dans le tableau 4.3.

o \ +	r* 1	r* 2	e 1	e 2
r* 1	63.4	40.5	63.2	41.3
r* 2	89.0	70.3	93.4	82.5
e 1	60.6	19.3	73.4	7.8
e 2	83.2	46.9	97.4	61.8

Tableau 4.3: *pourcentage de gains des o avec une BDP où les o et les + ont gagné le même nombre de parties*

On voit que le réseau joue un peu mieux avec les + qu'avec l'autre base d'apprentissage, surtout contre le réseau. En contrepartie, le réseau joue moins bien avec les o contre l'expert.

Nous avons réalisé deux autres apprentissages avec des BDP où ne se trouvaient que des positions apparues au cours de parties où les o, respectivement les +, avaient gagné. Les résultats sont exposés dans les tableaux 4.4 et 4.5.

o \ +	r* 1	r* 2	e 1	e 2
r* 1	61.5	44.2	47.1	19.1
r* 2	93.4	78.3	88.7	63.2
e 1	64.9	44.4	73.4	7.8
e 2	92.1	72.1	97.4	61.8

Tableau 4.4: *pourcentage de gains des o avec une BDP constituée de positions où les o ont gagné toutes les parties*

o \ +	r* 1	r* 2	e 1	e 2
r* 1	53.3	39.3	32.1	6.2
r* 2	83.8	66.2	80.5	43.3
e 1	82.5	53.3	73.4	7.8
e 2	96.6	79.3	97.4	61.8

Tableau 4.5: *pourcentage de gains des o avec une BDP constituée de positions où les + ont gagné toutes les parties*

On voit que l'échantillon d'apprentissage a une grande influence sur les résultats. En particulier, les deux échantillons utilisés dans les tableaux 4.4 et 4.5 font jouer le réseau bien plus mal qu'avec ceux utilisés dans les tableaux 4.2 et 4.3. On aurait pu s'attendre à ce que le réseau joue encore mieux les o (resp. les +) en utilisant l'échantillon 4.4 (resp. 4.5), mais ce n'est pas le cas.

4.7. Rôle de l'expert

L'expert est indispensable avec cette méthode. En effet, si l'expert n'était pas fiable, l'étiquette serait sans doute fausse, ce qui nuirait à l'apprentissage. Si on avait pris comme BDP des positions résultant de coups aléatoires, on aurait observé deux « anomalies » :

1. Les parties ne seraient pas représentatives de parties réelles : elles seraient plus longues et le vainqueur ne serait pas toujours « logique » (l'adversaire aura peut-être laissé passé une victoire).
2. Certaines positions seraient mal étiquetées; celles justement où le joueur aurait gagné la partie s'il avait été intelligent et où il a finalement perdu.

5. Autre méthode

Dans ce chapitre, nous allons essayer une autre approche. Plutôt que de coder la position directement, on va essayer d'utiliser des caractéristiques de la position. Cette approche est plus dans l'esprit des programmes de jeu classiques.

La fig. 5.1 schématise le processus suivi.

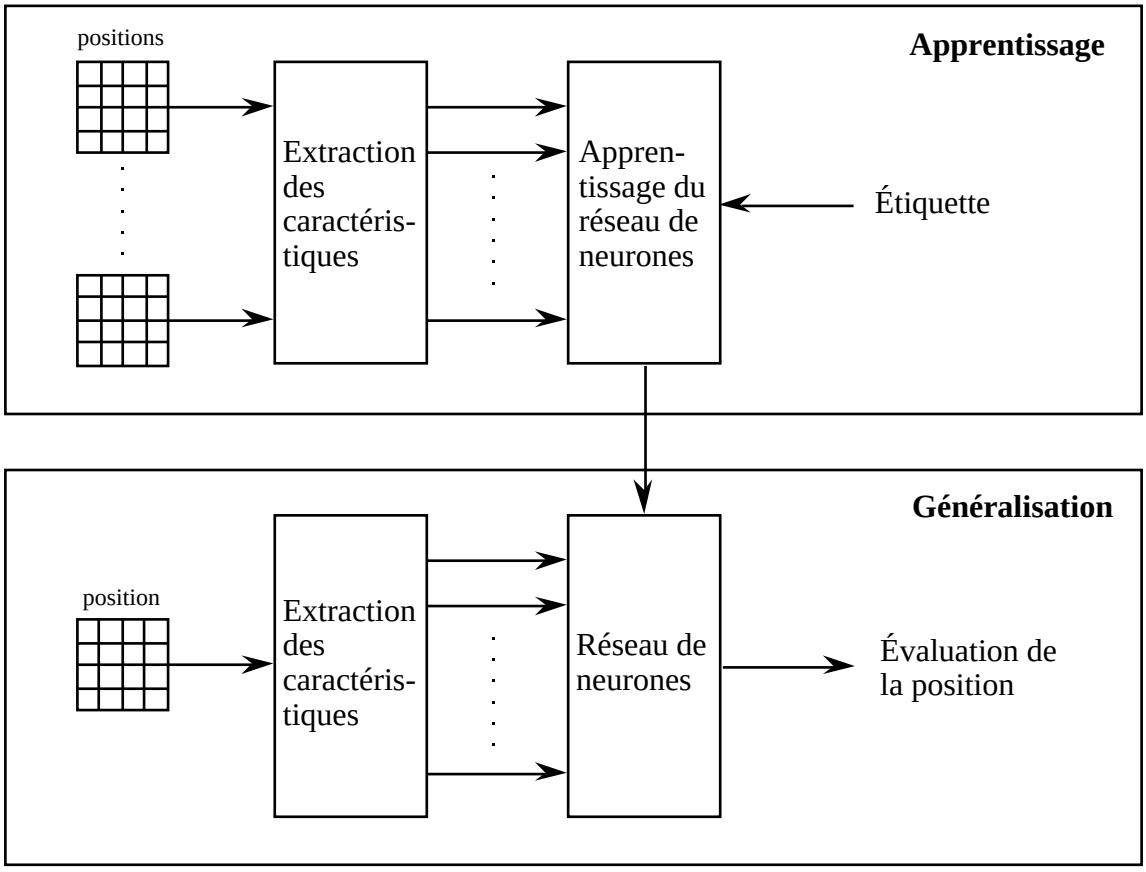

Fig. 5.1: *deuxième approche*

5.1. Extraction des caractéristiques d'une position

Comme on l'a dit dans l'introduction de cet article, on utilise généralement des caractéristiques d'une position plutôt que la position elle-même pour construire la fonction d'évaluation d'un jeu. On guide ainsi l'ordinateur en lui indiquant ce qui est important et ce qui l'est moins. Nous allons tenter d'appliquer cette technique. Un inconvénient de cette méthode est que l'on ne sait pas *a priori* quelles sont les « bonnes » caractéristiques.

Introduisons d'abord deux définitions :

DÉFINITIONS

Déf. 5.1. *rangée* : ligne, colonne ou diagonale.
Déf. 5.2. *n-grappe* : n pions de même couleur adjacents deux à deux sur une même rangée (voir fig. 5.2).

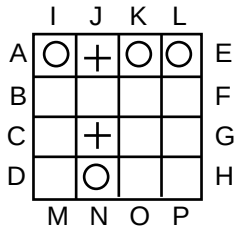

Fig. 5.2: *Exemples de n-grappes: une 1-grappe et une 2-grappe de o sur la première ligne. Deux 1-grappes de + et une 1-grappe de o sur la deuxième colonne.*

Un des avantages du jeu du push-over est que les caractéristiques importantes d'une position sont assez intuitives, même pour un joueur débutant. Les 44 caractéristiques que nous avons choisies sont :

Pour chacune des 10 rangées (pour les groupes de caractéristiques a & b) :

a1) le nombre de pièces amies
a2) le nombre de pièces ennemies
b1) la plus grande n-grappe de pièces amies
b2) la plus grande n-grappe de pièces ennemies

c1) le nombre de pièces amies dans les cases centrales
c2) le nombre de pièces ennemies dans les cases centrales
d1) le nombre de pièces amies dans les 4 coins
d2) le nombre de pièces ennemies dans les 4 coins.

Nous justifions ci-après le choix de ces groupes de caractéristiques en reprenant la même numérotation que ci-dessus :

a) Plus on a de pions dans une rangée, mieux c'est, puisque le but du jeu est d'en aligner quatre.
b) D'après [LeN82], il est préférable d'avoir des grappes plutôt que des pions éparpillés.
c) Il faut au moins deux coups pour faire sortir ces pions du damier et il existe trois possibilités de compléter une rangée.
d) Il existe trois rangées adjacentes au lieu de deux, mais ces pions peuvent être sortis de deux manières au lieu d'une.

Les valeurs possibles de chacune des caractéristiques sont comprises dans l'intervalle [0,4].
Il est intéressant de constater qu'on ne peut pas reconstituer la position en n'utilisant qu'un seul des quatre groupes de caractéristiques ci-dessus. Par contre, on peut, dans un damier 4x4, reconstituer le damier à une symétrie près en utilisant a) et b).
On aurait aussi pu prendre comme caractéristique le nombre total de pions de chaque joueur, mais cette information est redondante avec a).

5.2. Réseau de neurones

Nous avons utilisé un réseau multicouche avec 44 unités d'entrée, 30 neurones en couche cachée et 1 neurone en sortie. Ici aussi, cette configuration a été choisie après de nombreux essais. La fonction de transition utilisée pour chaque neurone est la sigmoïde donnée par la formule (1.4).

Les entrées correspondent aux caractéristiques de la position donnée; leurs valeurs sont comprises entre 0 et 1 (on a simplement divisé chaque caractéristique par 4). Les entrées peuvent donc valoir 0, 0.25, 0.5, 0.75, ou 1.

La valeur donnée par le neurone de sortie est un nombre réel pouvant prendre toutes les valeurs comprises dans l'intervalle [0,1]. L'échelle est la suivante: 0.5 correspond à une position nulle, 0 à une position perdue et 1 à une position gagnante.

Remarquons au passage que ce réseau est beaucoup plus grand que celui utilisé avec la première méthode.

5.3. Résultats obtenus

Les résultats obtenus furent décevants. En effet, malgré de nombreux essais, on n'a jamais pu obtenir des résultats comparables à ceux obtenus avec la première méthode, bien que l'apprentissage se soit bien passé.

Cela vient sans doute du fait que l'on donne moins d'informations au réseau en entrant des caractéristiques qu'en codant directement la position. De plus, les caractéristiques choisies ne sont peut-être pas les plus pertinentes. Remarquons enfin que bien que l'on ait moins d'informations, paradoxalement le nombre d'entrées du réseau est plus grand.

Bien que cette méthode n'ait pas bien fonctionné dans le contexte du push-over, il est possible qu'elle donne des résultats avec des jeux plus compliqués ou à information incomplète comme les jeux de cartes par exemple.

6. Conclusion

Nous tirons ici les conclusions de notre travail.

6.1. Critique de la méthode

Ce travail a montré que l'on pouvait utiliser les réseaux de neurones dans la théorie des jeux. En effet, on peut utiliser la méthode décrite pour d'autres jeux comme par exemple l'Othello, les dames, go-moku, etc. Une propriété intéressante est que l'on ne peut pas prévoir le coup de l'ordinateur si l'on ne connaît pas les poids synaptiques du réseau; en effet rien n'indique qu'il jouera le meilleur coup possible (celui de l'expert): cela dépendra de sa manière de généraliser.

Un autre intérêt de la méthode décrite est qu'elle peut aussi s'appliquer à d'autres domaines, notamment à la reconnaissance des formes et à la classification.

La contrainte majeure est la nécessité d'un expert, soit un humain, soit un ordinateur. Il faut dire aussi qu'il faut beaucoup de temps pour déterminer un « bon » réseau multicouche et de « bons » échantillons d'apprentissage.

Un autre inconvénient est que le temps de calcul de la fonction d'évaluation est beaucoup plus important que dans un programme classique, même en étant futé (en tabulant la fonction sigmoïde par exemple), et même en utilisant des réseaux aussi petits que possible. Dans un jeu, où l'interaction humain-ordinateur est très importante, cela constitue un défaut majeur: il est en effet très désagréable d'attendre que l'ordinateur daigne jouer son coup. Il est d'ailleurs curieux que les autres articles sur le sujet ne parlent absolument pas du temps de calcul...

6.2. Travail futur

Il pourrait être intéressant de voir dans quelle mesure les réseaux de neurones peuvent aider à diminuer la profondeur de la recherche dans les arbres de jeux. Nous avons en effet constaté, comme Corsi et Noël dans [Cor90], qu'un réseau de neurones jouait presque aussi bien et parfois mieux qu'un programme classique, tout en utilisant une profondeur de recherche plus faible.

Le rôle de l'expert est très important, comme on l'a vu au paragraphe 4.7. Il serait intéressant de développer une méthode où aucun expert n'intervient. Mais rien n'indique que cela soit faisable: si l'on observe le comportement humain, on se rend compte que le rôle du maître est

tout à fait indispensable pour apprendre, que ce soit dans le domaine des jeux où dans la vie en général.

Références

[Ama92] Amaldi E., "Apprentissage supervisé dans les réseaux en couches", XIII^{e} cours postgrade en informatique technique, EPFL, 1992

[Ber79] Berliner H., "On the construction of evaluation functions for large domains", International joint conference on artificial intelligence, IJCAI-79, Tokyo, pp. 53-55

[Cor90] Corsi P., Noël H., "Validation d'une approche neuronale dans le contexte de la théorie des jeux", Proceedings of Neuro-Nîmes 1990, pp. 287-300

[DeJ88] De Jong K. A., Schultz A. C., "Using experience-based learning in game playing", Proceedings of the 5th international conference on machine learning, 1988, pp. 284-290

[Elc67] Elcock E. W., Murray A. M., "Experiments with a learning component in a go-moku playing program", in Collins, Dale, Michie, "Machine Intelligence 1", Edinburgh University Press, 1967, pp. 87-103

[Gri74] Griffith A. K., "A comparison and evaluation of three machine learning procedures as applied to the game of checkers", Artificial Intelligence 5, 1974, pp. 137-148

[Kof68] Koffman E. B., "Learning games through pattern recognition", IEEE transactions on systems science and cybernetics, Vol. SSC-4, No 1, March 1968

[Lee88] Lee K.-F., Mahajan S., "A pattern classification approach to evaluation function learning", Artificial Intelligence 26, 1988, pp. 1-25

[LeN82] Le-Ngoc T., Vroomen L. C., "Programming strategies in the game of push-over", IEEE Micro, August 1982, pp. 58-68. Aussi in Levy D. N. L, "Computer games II", Spriger-Verlag, 1988, pp.461-474

[Min83] Minoux M., "Programmation mathématique: Théorie et algorithmes", Tome 1, Dunod, Paris, 1983

[Put89] Putter G., Smeets J. J., "Some experience with a self-learning computer program for playing draughts", in Levy, Beal, "Heuristic programming in artificial intelligence: the first computer olympiad", Ellis Horwood Limited, Chichester, 1989, pp. 176-194

[Res90] Resnik Ph., "A simplified version of the GRADSIM neural network simulator for feed-forward learning problems"

[Rum86] Rumelhart D. E., Hinton G. E., Williams R. J., "Learning representations by backpropagation errors", Nature 393, 1986, pp. 533-536

[Sam59] Samuel A. L., "Some studies in machine learning using the game of checkers", IBM Journal of research and development 3, July 1959, pp. 211-229

[Sam67] Samuel A. L., "Some studies in machine learning using the game of checkers II: recent progress", IBM Journal of research and development 11, November 1967, pp. 601-617

[Sej89] Sejnowski T. J., Tesauro G., "A parallel network that learns to play backgammon", Artificial Intelligence 39, 1989, pp. 357-390

[Sha50] Shannon C. E., "Programming a computer for playing chess", Philosophical magazine, Serie 7, Vol. 41, Number 314, March 1950, pp. 256-275