

Michelangelo
Chapela Sistina, Vatican

Sommaire

1	Introduction	6
1	But	6
2	Historique	7
3	Les principales applications de l'IA	10
4	Problématique de l'IA	11
5	Plan du cours	11
2	Recherche dans un espace d'états	13
1	Introduction	13
2	Définitions	14
3	Structures de données	15
4	Résolution d'un problème par recherche dans l'espace des états	16
4.1	Algorithmes sur les graphes	18
4.1.1	Parcours en largeur d'abord	18
4.1.2	Parcours en profondeur d'abord	20
5	Algorithme A^* [Nilson 1971]	22
6	Formalisme ET-OU	25
6.1	Structures de données	25
6.2	Recherche OU	26
6.3	Recherche ET	26
6.3.1	Exemple	27
6.4	Problème du formalisme des graphes ET-OU	27
7	Algorithmes de jeu	30
7.1	Algorithme minimax	30
7.2	Algorithme NEGMAX [Knuth & Moore 1975]	31
7.3	Algorithme Alpha-Béata	31
7.4	Améliorations possibles des algorithmes de jeu	32

3	Systèmes experts	33
1	Introduction aux systèmes experts	33
2	Exemple de système expert	35
2.1	Utilisation des systèmes experts	35
3	Problèmes rencontrés par les systèmes experts	36
4	Logiques formelles	37
1	Logique des propositions (logique d'ordre 0)	38
1.1	Exemples de démonstration	39
1.1.1	Propriétés	40
1.2	Le calcul des prédicats (logique du premier ordre)	40
1.3	Propriétés	41
1.4	Méthodes de résolution	42
1.4.1	Herbrand [Herbrand 1930, Robinson 1965]	42
1.4.2	Unification	42
1.4.3	Diminution	42
1.4.4	Stratégie de saturation simple	43
1.4.5	Stratégie de saturation avec simplification	43
1.4.6	Arbre de dérivation	43
1.4.7	Extraction de réponses	44
1.5	Autres logiques	45
5	Logique floue	46
1	Ensembles flous	46
1.1	Définitions	47
1.1.1	Support	47
1.1.2	Hauteur	47
1.1.3	Noyau	47
1.1.4	Cardinal	47
1.1.5	Égalité	47
1.1.6	Inclusion $A \subseteq B$	47
1.1.7	Intersection $A \cap B$	48
1.1.8	Union $A \cup B$	48
1.1.9	Complément	48
1.2	Propriétés	48
2	Logique floue	49

2.1	Proposition floue	50
2.1.1	Proposition floue simple	50
2.1.2	Proposition floue composée	50
2.1.3	Implications : règles floues	51
2.1.4	Modus ponens généralisé	51
2.1.5	Applications	51
2.1.6	Mise en forme	52
3	Exemple : système d'arrosage	52
6	Neurone biologique et formalisme	56
1	Le neurone biologique	56
1.1	Modèle du neurone formel	57
7	Réseaux de neurones supervisés	60
1	Mémoire associative linéaire	60
1.1	Apprentissage	60
1.2	Rappel	61
2	Réseaux de Hopfield	61
2.1	Dynamique du réseau	61
2.2	Evolution de l'énergie	62
2.3	Application au cas des MA	63
3	Limitations des MA	64
4	Perceptron	64
5	Adaline	65
6	Limitations	65
7	Rétropropagation du gradient	65
7.1	Une couche	66
7.2	2 couches	66
7.2.1	Remarques sur l'utilisation de la rétropropagation	68
8	Réseaux de neurones non-supervisés	70
1	Winner Take All (WTA)	70
2	Carte de Kohonen	72
3	Adaptive Resonance Theory (ART)	74
9	Renforcement	77
1	Retour sur la règle de Hebb et le LMS	77

2	Conditionnement instrumental	77
2.1	Formalisme des méthodes de renforcement	78
2.2	Les techniques de renforcement	81
2.2.1	Schéma de fonctionnement général	81
2.2.2	L'algorithme TD(λ)	82
2.2.3	Le Q-learning	83
2.2.4	L'algorithme Dyna-Q	83
10	Multi-agents	85
1	Introduction	85
1.1	Pourquoi distribuer l'intelligence ?	85
1.2	Définition d'un agent (physique ou virtuel)	86
1.3	Définition d'un système multi-agents (SMA)	87
1.4	Agent logiciel/agent physique	87
1.5	Niveaux d'organisation	88
2	Conception d'un SMA	88
2.1	Pourquoi utiliser des SMA ?	88
2.2	Quelle est la nécessité d'utiliser des SMA ?	89
2.3	Inspiration	89
2.4	Quand utiliser un SMA ?	89
2.5	Conception	90
2.6	Applications	91
3	Agents mobiles	91
3.1	Intérêt	91
3.2	Avantages	91
3.3	Inconvénients	92
4	Nouvelle architecture ISO	92
5	Application : Projet Netmarché	92
5.1	Présentation	92
5.2	Implantation : FIPA-OS	92
5.3	Autre stratégie possible	93
5.4	Développement	93
6	Applications	93
6.1	Tours de Hanoï	93
A	Filtrage de Wiener	94

B	Algorithmes d'optimisation	96
0.2	Le Gradient Déterministe (GD)	96
0.3	Le Gradient Stochastique (GS) et le LMS (Least Mean Square) . .	96

Chapitre 1

Introduction

Depuis la nuit des temps, l’imaginaire collectif semble peuplé de créatures intelligentes créées de toutes pièces par l’être humain (le golem juif, la statue de pygmalion...). Ces contes semblent traduire un désir profond de l’homme de reproduire les mécanismes à la base de son intelligence. Outre ces visions fantasmatiques, l’histoire dénombre de multiples tentatives concrétisées, pour construire des machines capables de se substituer à l’homme pour effectuer à sa place certaines tâches “intellectuelles” répétitives ou fastidieuses (le boulier, la machine de Pascal, les automates de Vaucanson, etc.). L’apparition des ordinateurs a permis de cristalliser autour d’un ensemble de paradigmes, la compréhension et la reproduction des mécanismes de l’intelligence.

1 But

Le but de ce cours est de vous faire appréhender les techniques utilisées pour résoudre des problèmes liés à l’analyse, au traitement et à l’apprentissage de connaissances¹. Cette problématique s’inscrit dans le cadre général des sciences cognitives, dont le souci est l’étude de l’intelligence. Ces disciplines regroupent des domaines aussi variés que philosophie, psychologie, biologie ainsi que des sciences pour l’ingénieur telles qu’informatique et robotique. Depuis les premiers ordinateurs, il y a en effet eu une interaction forte entre les sciences qui essayaient de comprendre le fonctionnement de l’intelligence, et celles qui tâchaient d’en reproduire le fonctionnement. Pour comprendre cette interaction, commençons par un petit historique.

¹Le dictionnaire définit la connaissance comme “Ce que l’on a acquis par l’étude ou par la pratique”.

2 Historique

PHILOSOPHIE :

- 1596-1650 : Descartes suppose que les animaux sont des sortes de machines vivantes. Les hommes sont dotés d'une âme échappant à la matière.
- 1646-1716 : Leibniz introduit le matérialisme. Selon lui, tout est régi par des lois physiques.
- 1561-1626 : Bacon et l'empirisme : tout ce qui atteint la compréhension l'est par l'intermédiaire des sens
- 1711-1776 : Hume construit son principe d'induction : les règles générales sont acquises par exposition à des associations répétées de leurs éléments
- 1872-1970 : Russel formalise les travaux de Hume et introduit le positivisme logique : le savoir est caractérisé par des liens logiques, finalement reliés aux sens.

MATHS :

- 9ème siècle : al-Khowarazmi introduit l'algorithmique, l'algèbre et la notation arabe
- 1815-1864 : Boole construit l'algèbre binaire et la logique formel
- 1848-1925 : Frege établit la logique du premier ordre
- 1906-1978 : Gödel établit son théorème d'incomplétude et d'indécidabilité (il n'existe pas de procédure capable de décider qu'un non-théorème en est un).
- 1912-1954 : Turing montre que toute fonction calculable l'est par une machine de Turing (et donc par un ordinateur). Attention ! il existe des fonctions non-calculables.

PSYCHOLOGIE :

- 1821-1894 Helmholtz, 1832-1920 Wundt : origine de la psychologie scientifique
- 1878-1958 Watson, 1874-1949 Thorndike : introduction du behaviorism : le comportement d'un individu est le produit de conditionnements multiples.
- 1948 Tolman : Le cerveau contient des représentations internes sous forme de cartes cognitives

AUTOMATES :

- 1748 : Julien Offroy de la Mettrie assimile l'homme à une machine complexe
- 1709-1782 : Vaucanson construit des automates (canard, joueur de flute traversière) qui donnent l'illusion d'être vivants.
- 1592-1635 : Schickard créé la première machine à calculer à l'aide d'engrenages (addition, soustraction, multiplications, mémorisation de résultats et dépassement de capacité).
- 1623-1662 : Pascal (1642) réalise la "Pascaline" ? Leibniz construit une machine effectuant les 4 opérations élémentaire.
- 1785-1870 : Thomas de Colmar construit "l'arithmomètre"
- 1792-1871 : Babage réalise une "machine analytique" programmable grâce à des cartes de variables et des cartes d'opérations (il est à noter que Ada de Lovelace, fille de Lord Byron, fut la première personne à concevoir des programmes. C'est en son honneur que le langage Ada porte son nom).

Il faut attendre les travaux de Turing (1912-1954) pour voir apparaître la première tentative délibérée pour comprendre et reproduire l'intelligence humaine. Turing construit son action autour du paradigme fondateur consistant à affirmer que "toute l'intelligence cognitive humaine est modélisable dans le cadre formel de la machine de Turing". La machine de Turing est une machine abstraite capable d'effectuer des calculs. Elle est composée d'une unité de traitement et d'une mémoire qui n'est autre qu'un ruban à partir duquel la machine peut lire données et programme, stocker des informations temporaires et imprimer les résultats. Un ordinateur est donc une machine de Turing améliorée. Turing propose un test, capable, selon lui, de déceler si un système a reproduit l'intelligence humaine. Ce test consiste à faire dialoguer un humain et le système et déterminer si l'humain peut déceler si le système n'est pas humain.

INFORMATIQUE :

- 1940 : Heath Robinson crée par l'équipe de Turing pour décoder les messages allemands (technologie à base de relais)
- 1943 : Colossus, encore créé par l'équipe de Turing (technologie à base de lampes à vide)
- 1940-42 :
 - Z3; ordinateur programmable doté du premier langage évolué
 - ABC; Iowa state university

- Mark I, II et III; Harvard
- ENIAC : University of Pennsylvania
- EDVAC : Programme mémorisable (sur les conseils de Von Neumann²)
- IBM701
- 1965-1980 Mise au point d’algorithmes efficaces. Prise de conscience de la grande complexité du problème
- 1982 ordinateurs de la 5eme génération (Japon) pour 1992 une machine parallèle capable de raisonner.

Avec l’apparition des machines massivement parallèles le niveau de représentation de l’information utilisant des “symboles” est il toujours pertinent ?

INTELLIGENCE ARTIFICIELLE :

- 1943 : McCulloch et Pitts créent le modèle du neurone formel
- 1948 : Création de la cybernétique (science des systèmes) par Norbert Wiener.
- 1949 : Hebb établit la première règle d’apprentissage neuronal
- 1950 Shannon, 1952 Samuel, 1953 Turing : machine pour jouer aux échecs
- 1956 Workshop où est né le terme “intelligence artificielle”
- 1959 Rochester : Geometry Theorem Prover
- 1958 McCarthy au MIT crée le LISP et le “time sharing”. Créé DIGITAL.
- 1960 John McCarthy, Allen Bewell & Herbert Simon: L’ordinateur peut être utilisé pour autre chose que des calculs “manipuler des symboles” (idée proposée par Ada Lovelace amie de Babbage 1842)
- 1969 arrêt des RNs (Minsky & Paper 1969) limitations des perceptrons
- 1969-1979 : systèmes experts
- Depuis 1986 : retour des réseaux de neurones

Les langages d’IA :

- Lisp 1956/58 John McCarthy au MIT

²Von Neumann (1903-1957) est un mathématicien qui a travaillé sur de nombreux projets liés aux balbutiements de l’informatique.

- Les moteurs d'inférences³: l'importance de l'organisation des données (structure, liste...) Programmer à partir des données
- Prolog 1973 Colmerauer : Générateurs de Systèmes Experts

Le développement de l'intelligence artificielle a été conditionné d'une part par le cloisonnement des sciences du vivants et des sciences pour l'ingénieur (l'IA se réclamant des sciences exactes), et d'autre part par une orientation principalement exclusivement basée sur le paradigme de Turing. C'est pourquoi, malgré des débuts ancrés dans la biologie et la psychologie, l'IA est rapidement passé à un formalisme mathématique très poussé et s'est attaquée à des problèmes plutôt symboliques. Elle s'est d'autant plus confortée dans ces idées que les débuts de la cybernétique⁴ (Widrow60 : adaline, Rosenblatt62 : perceptron), ont conduit à une impasse du fait de la démonstration de Minsky69 de la limitation des perceptrons.

3 Les principales applications de l'IA

- la traduction automatique (initialisé pendant la guerre)
- Les systèmes d'aide (au diagnostic, à la programmation)
- Les système de résolutions de problèmes — GPS (general problem solver) A. Newell & H. Simon 1978
- Les jeux La simulation
- ELIZA J. Weizenbaum 1960 (simulation d'un dialogue avec un psy)

Cependant, cette approche s'est trouvée confrontée à de nombreux problèmes :

- Problème de la traduction automatique (Symbol Grounding Problem)
 - Problèmes de la traduction syntaxique:
 - anglais → russe → anglais
 - “l'esprit est fort, mais la chair est faible” → “La vodka est forte, mais la viande est pourrie”
- Problème sur les applications à taille réelle
- Modélisation du sens commun

³sorte de programme universel (machine à raisonner)

⁴étude des processus de commande et de communication dans le vivant, les machines, les systèmes économiques et sociologiques

- Explosion combinatoire
- L'apprentissage
- Niveau de représentation
- Symbol Grounding Problem (ex. de la chambre chinoise).
- Complexité/chaos

4 Problématique de l'IA

Le but d'un système d'intelligence artificielle est d'aider à la résolution d'un problème posé par l'utilisateur. Mais qu'est-ce qu'un problèmes (comment décrire le problème, connaissances a priori) ? Quel est le niveau d'autonomie de l'IA (peut-il y avoir interaction entre l'utilisateur et le système et de quelle manière) ? Le système doit-il gérer un problème particulier ou le spectre de résolution est-il large (peut-on généraliser le problème, quelle est l'influence du changement d'échelle, y'a-t-il plusieurs solutions, veut-on la meilleure solution ou une bonne solution) ? Quel est le degré d'évolutivité du système (peut-il assimiler de nouvelles données/apprendre) ?

Répondre à ces questions nous oriente vers un type de résolution de problèmes adapté (optimisation, recherche dans un espace d'états, logique, réseaux de neurones supervisés ou non). Toutefois, il est parfois encore possible de résoudre le problème de différentes manières. Le choix se fait alors en fonction de 2 types d'approches :

- Une approche analytique et formelle
- Une approche holiste essayant d'intégrer la complexité

Ces 2 approches sont souvent contradictoires car elle amène, pour la première, à décomposer le problème en partant d'une description abstraite puis en la raffinant petit à petit (approche "top-down") alors que la deuxième tente plutôt de construire un système en intégrant hiérarchiquement des mécanismes de plus bas niveau (approche "bottom-up")

La décision finale est alors laissée à la discrétion du programmeur en fonction de ses habitudes, de ses goûts ou de ses convictions !

5 Plan du cours

- Recherche dans des espaces d'états
 - Recherche avec heuristique : A^*

- Algorithme pour les jeux
- Présentation d'un système expert
- Présentation d'un langage d'IA : Prolog
- Logique formelle
- Logique floue
- Réseaux de neurones
 - supervisés
 - non-supervisés
- Techniques d'optimisation
 - Techniques de renforcement
 - Recuit simulé
 - Algorithmes génétiques
- Systèmes multi-agents

Chapitre 2

Recherche dans un espace d'états

1 Introduction

Supposons le problème qui consiste à trouver un nombre choisi entre 1 et 10 par un utilisateur. Comment représenter ce problème ?

il s'agit en fait de trouver **UN** état particulier parmi les **10** possibles. Par ailleurs, dans ce cas, l'étiquetage de chaque état est aisé puisque on peut leur associer 1 numéro unique. *L'ensemble des états possibles d'un problème est appelé "espace d'états".*

Résoudre ce problème revient alors à proposer **une stratégie** permettant de parcourir l'espace d'états de manière à trouver l'état unique solution du problème.

Cette stratégie conditionne les opérateurs qui pourront être utilisés pour passer d'un état à un autre mais aussi la représentation de l'espace d'états.

Exemple : Stratégie aléatoire

On donne un nombre au hasard compris entre 1 et 10.

Connaissances a priori : aucune !

Interaction : information succès/échec.

Représentation de l'espace d'état : aucune (ou presque : contrainte $n \in [0, 10]$!)

Succès : assuré ! (au bout d'un temps au maximum infini).

Généralisation : à tout problème.

Exemple : Stratégie ad-hoc \Rightarrow dichotomie

On commence avec 5 et on partitionne l'espace de recherche en fonction des réponses.

Connaissances a priori : dans la stratégie !

Interaction : information supérieur/inférieur.

Représentation de l'espace d'état : arbre binaire de recherche

Succès : assuré ! (au bout de $\log(10)$ essais).

Généralisation : aucune.

Pour rendre un système généralisable il faudrait pouvoir séparer plus clairement la stratégie des connaissances.

2 Définitions

Considérons un système interagissant avec son milieu. On représente l'ensemble des situations que le système peut rencontrer par un ensemble d'états accessibles S . Dans chaque état, le système est susceptible d'utiliser un certain nombre d'opérateurs appartenant à l'ensemble O . Après la réalisation d'une certaine action depuis l'état $s \in S$, le système se retrouve dans l'état $s' \in S$ (voir figure 1). Le problème est de savoir quel action effectuer dans tel ou tel état pour atteindre un objectif donné.

État : c'est une description symbolique à un moment donné:

- des objets
- de leurs propriétés
- de leurs relations

Objectif : (résolution du problème)

Opérateurs: il sont définis par

- leur domaine de validité
- la manière dont il permette de passer d'un état vers un autre

Système de production: il peut être défini par

- des règles de production (si Conditions alors Actions)

- une base de données
- une stratégie de commande (heuristiques, méta règles...)

Un système de production est dit :

- monotone : si tous les opérateurs sont toujours applicables
- partiellement commutatif : si $r_1 r_2$ et $r_2 r_1$ sont toujours possibles (pas besoin de “backtracking” en cas d’échec)
- commutatif : s’il est monotone et partiellement commutatif

Exemple : le jeu du Taquin (puzzle)

Représentation de l’espace d’état : graphe d’états

Un état : un tableau à 9 cases.

Objectif : L’état [012345678].

Règles de production : opérateurs \leftarrow , \rightarrow , \uparrow , \downarrow .

Contraintes : effets de bords des règles de production.

3 Structures de données

Les structures de données utilisées pour les recherches sont classiquement des structures de graphe orienté ou d’arbre si le modèle s’y prête.

On a alors :

- Univers à exploré : graphe orienté
- État : noeud
- Opérateur : arc

Les questions importantes a se poser lors de l’élaboration du graphe sont les suivantes :

- La création du graphe complet est elle nécessaire?
- Le graphe doit il être explicite?

4 Résolution d'un problème par recherche dans l'espace des états

Les méthodes d'énumération sont liées à :

- La direction de recherche
 - chaînage avant : on part de l'état initial pour aller vers l'objectif
 - chaînage arrière : on part de l'objectif
- L'ordre d'énumération:
 - largeur d'abord (file)
 - profondeur d'abord (pile)
- La sélection des règles, fonction heuristique...

Parcours en largeur d'abord pour un arbre

Le principe de cet algorithme est très simple ; il consiste à partir de la racine et d'énumérer l'ensemble de ses descendants directs puis réitérer le processus sur les descendants eux-mêmes. On effectue ainsi un parcours “de gauche à droite et de haut en bas”.

On utilise 2 listes appelées OUVERT et FERME.

- La liste OUVERT est une file d'attente qui contient les sommets en attente d'être traités
- La liste FERME contient les sommet dont les successeurs, s'ils existent, ont déjà été énumérés.

L'algorithme est le suivant :

Largeur-D-abord

1. *Placer le sommet initial s_0 dans OUVERT*
2. *tant qu'OUVERT n'est pas vide*
 - soit n le premier élément d'OUVERT*
 - mettre les fils de n en queue d'OUVERT*
 - {effectuer le traitement pour n }*
 - mettre n dans FERME*
3. *fin tant que*

Parcours en profondeur d'abord

Le principe de cet algorithme est de descendre le plus profondément dans l'arbre avant de se déplacer en largeur. On effectue ainsi un parcours "de haut en bas et de gauche à droite".

La encore, on utilise 2 listes appelées OUVERT et FERME.

L'algorithme est le suivant :

Profondeur-D-abord

1. *Placer le sommet initial s_0 dans OUVERT*
2. *tant qu'OUVERT n'est pas vide et que la profondeur n'est pas la profondeur limite*
soit n le premier élément d'OUVERT
mettre les fils de n en tête d'OUVERT
{effectuer le traitement pour n }
mettre n dans FERME
3. *fin tant que*

Problème : Il se peut que l'on s'enfonce indéfiniment dans l'arbre. **Solution :** Tester une profondeur maximum \Rightarrow Dans ce cas, le parcours en largeur d'abord n'est pas complet.

4.1 Algorithmes sur les graphes

Il est tout à fait possible, moyennant quelques modifications, d'utiliser les parcours en largeur et profondeur d'abord. En fait, comme il y a des cycles dans le graphe, il ne faut pas reparcourir des sommets qui ont déjà été visités.

En largeur d'abord:

- Ne pas remettre dans OUVERT un sommet déjà traité

En profondeur d'abord:

- (définir la profondeur)
- Garder le sommet de plus faible profondeur
- Attention dans fermer il faut remettre à jour les profondeurs!

4.1.1 Parcours en largeur d'abord

On peut appliquer l'algorithme en considérant que l'on parcourt des arborescences associées aux sommets du graphe (l'arborescence est celle constituée des descendants qui n'ont pas encore été parcourus). Les arbres sous-jacents forment une forêt recouvrante du graphe (contient tous les sommets du graphe).

L'algorithme est le suivant :

Largeur-D-abord-graphes

1. *Visiter tous les sommets*

2. VISITER-SOMMET(S)

3. *Si s n'est pas dans OUVERT, ni dans FERME*

placer le sommet initial s dans OUVERT

tant qu'OUVERT n'est pas vide

soit n le premier élément d'OUVERT

*mettre les successeurs de n qui ne sont pas déjà dans OUVERT ou FERME
en queue d'OUVERT*

{effectuer le traitement pour n}

mettre n dans FERME

fin tant que

Remarque : le parcours en largeur d'abord est une stratégie COMPLÈTE.

4.1.2 Parcours en profondeur d'abord

L'algorithme est le suivant :

Profondeur-D-abord-graphes

1. *Visiter tous les sommets*

2. *VISITER-SOMMET(S)*

3. *Si s n'est pas dans OUVERT, ni dans FERME*

placer le sommet initial s dans OUVERT

tant qu'OUVERT n'est pas vide

soit n le premier élément d'OUVERT

*mettre les successeurs de n qui ne sont pas déjà dans OUVERT ou FERME
en tête d'OUVERT*

{effectuer le traitement pour n}

mettre n dans FERME

fin tant que

Remarque : le parcours en largeur d'abord est une stratégie COMPLÈTE.

Exemple 1 Recherche d'un élément situé entre 1 et 10 à partir des opérations +1 et -1 en partant de 5.

Algorithme itératif (en profondeur d'abord)

```
int prof_max=10;
int p[prof_max];

i=0
p[i]= situation_initiale

do
  {
    if(existe_fils(p[i]) && i<prof_max)
      {
        i++;
        p[i]=fils(p[i-1]);
      }
    else
      {
        while(i!=0 && existe_frere(p[i])==FALSE) i--;
        if(i!=0) p[i]=frere(p[i]);
      }
    fini=(p[i]==but);
  }
while(fini || (i==0));
```

5 Algorithme A^* [Nilson 1971]

Le principe est de trouver la solution présentant le moindre coût. On décompose en fait le problème en 2 :

Soit $f(n)$ le coût d'une solution passant par n : $f(n) = g(n) + h(n)$

Avec :

- $g(n)$ le coût effectif pour aller jusqu'à n (ne doit pas sous estimer le coût)
- $h(n)$ le coût pour atteindre la solution (heuristique – ne doit pas surestimer le coût)

On appelle $c(n, n_i)$ le coût pour passer de n à n_i .

On cherche en fait une estimation du coup: $f^*(n) = g^*(n) + h^*(n)$.

A^* pour un arbre

2 listes: *OUVERT* (sommets en attente) et *FERME* (sommets traités) ou piles (largeur ou profondeur d'abord)

1. placer le sommet initial n_0 dans *OUVERT*, $f^*(n_0) = 0$, $n = 0$
2. si *OUVERT* vide \Rightarrow échec
3. $n = n + 1$ le sommet de *OUVERT* ayant le plus faible f^* est mis dans *FERME*, l'appeler n
4. énumérer les successeurs du sommet mis dans *FERME* les mettre à la fin de *OUVERT* et initialiser les pointeurs vers n
Si l'objectif atteint sortir (pointeurs pour avoir une trace)
5. aller en 2

A* pour un graphe

1. placer le sommet initial n_0 dans OUVERT, $n = 0$, calculer $f^*(n_0)$
2. si OUVERT vide \Rightarrow échec
3. $n = n + 1$ le sommet de OUVERT ayant le plus faible f^* est mis dans FERME, l'appeler n
4. énumérer les successeurs m_i du sommet n mis dans FERME. Si l'objectif atteint sortir (pointeurs pour avoir une trace)
Pour chaque successeur :
 - (a) m_i pointe vers n (pointeur arrière)
 - (b) $f^*(m_i) = g^*(m_i) + h^*(m_i)$ avec $g^*(m_i) = g^*(n) + c(n, m_i)$
 - (c) si m_i n'est ni dans OUVERT ni dans FERME - l'ajouter à OUVERT, calculer $f^*(m_i)$
 - (d) si m_i est déjà dans OUVERT ou FERME l'appeler k si $f^*(k) > f^*(m_i)$:
 - actualiser k pointe vers n et $f^*(k)$
 - si m_i est dans FERME, le remettre dans OUVERT
5. aller en 2

Théorème: Il a été prouvé que l'algorithme A* trouve le chemin de coût minimum si $h^*(n) < h(n)$ et si $c(n, m) > 0$ (condition d'admissibilité de l'heuristique).

Remarques:

- si $h^*=h$ on a la solution optimale directement
- si $h^*=0$ on a un coût uniforme aveugle
- A1 est dit mieux "informé" que A2 si $h1^*(n) > h2^*(n)$
- On peut préférer $h^*(n) > h(n)$ pour trouver une bonne solution rapidement (pas optimale).

Problèmes:

- Coût de recherche / optimalité
- Limitations de calcul et de mémoire \Rightarrow compromis
- Trouver une bonne fonction h^*

Minimiser l'effort de recherche : $f^* = h^*$ ($g^* = 0$)

Compromis : $f^* = (1 - w) \cdot g^* + w \cdot h^*$ $0 < w < 1$

En général $0,5 < w < 1$ donne de bons résultats. w peut varier dans le temps (approche dynamique)

6 Formalisme ET-OU

Les graphes ET/OU servent dans le cas où l'on doit résoudre des sous problèmes pour résoudre un problème général.

Exemple:

Pour résoudre P, il faut :

Résoudre P1 et P2 ou P3 et P4 ou P5...

Mettre un problème sous la forme d'un arbre ET/OU revient à séparer en 2 niveaux les cas ET et OU :

- Sommet OU : il suffit de résoudre 1 des fils
- Sommet ET : il faut résoudre tous les fils

6.1 Structures de données

```
typedef struct type_regle{
    char tete[100];
    type_cond *queue;
    int active;
} type_regle;

typedef struct type_cond{
    char cond[100];
    type_cond *suivante;
} type_cond;
```

6.2 Recherche OU

```
int recherche_ou( type_regle * les_regles, char *probleme )
{
    type_regle une_regle;
    int succes=FALSE;

    if(appartient( base_de_fait, probleme)) return TRUE;

    while((vide_regle(les_regles)==FALSE )
    {
        une_regle=selection_regle(les_regles);

        if(strcmp(le_probleme,une_regle.tete)==0)
            if(recherche_et(une_regle.queue))
            {
                inhibe(une_regle);
                ajouter(base_de_fait, le_probleme);
                return TRUE;
            }
    }
    return FALSE;
}
```

6.3 Recherche ET

```
int recherche_et(type_cond des_problemes)
{
    int reussite;
    char *un_probleme;

    if(des_problemes==NULL) return TRUE;

    un_probleme=prend_probleme(des_problemes);
    reussite=recherche_ou(un_probleme);
    if(reussite==FALSE) return FALSE;
    return recherche_et(des_problemes);
}
```

6.3.1 Exemple

R1: $F \rightarrow B, D, E$

R2: $A \rightarrow D, G$

R3: $A \rightarrow C, F$

R4: $X \rightarrow B$

R5: $E \rightarrow D$

R6: $H \rightarrow A, X$

R7: $D \rightarrow C$

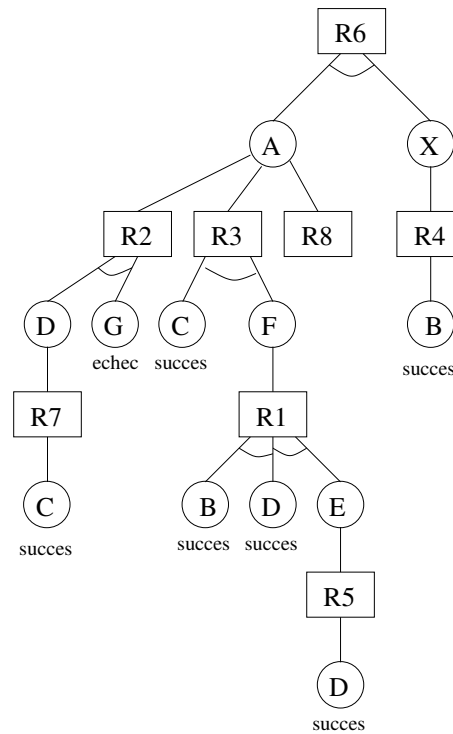
R8: $A \rightarrow X, C$

R9: $D \rightarrow X, B$

Base de faits : $\{B, C\}$

But : on cherche à résoudre H

Le problème se pose alors sous la forme :



```
sprintf(le_probleme, 'H');
succes=recherche_ou(base_de_regle, le_probleme);
```

6.4 Problème du formalisme des graphes ET-OU

Les sous-problèmes peuvent-ils être résolus de manière indépendante ? [Levi & Sirovitch 1975]

Difficultés si :

- nécessité de consistance entre les variables
- utilisation de ressources rares

Exemples :

- Un fait non intéressant peut être choisi comme solution d'un sous graphe et empêcher la résolution globale du problème (on ne peut plus revenir en arrière).
- Un fait peut être utilisé indéfiniment pour la démonstration alors qu'il est en fait disponible en quantité limitée.

Solution: Prendre 1 ou plusieurs noeuds en entrée \Rightarrow reformuler les règles

exemple: si R est une ressource unique

si le pb est d'atteindre P1 et P2 alors le probleme doit s'écrire $P1 \& P2 \& R$

1. S \rightarrow P1 & P2 & R
2. P1 & R \rightarrow T
3. P1 \rightarrow T
4. P2&R \rightarrow P3
5. P3 \rightarrow T
6. R \rightarrow T

Cas 1 : Problème de résolution hâtive

La base de faits est la suivante :

F1: Turing est humain

F2: Socrates est humain

F3: Socrates est grec

La base de règle est :

R1: Trouver un grec faillible \rightarrow

- trouver quelque chose faillible
- trouver quelque chose grec

R2: Trouver quelque chose faillible \rightarrow trouver quelque chose humain

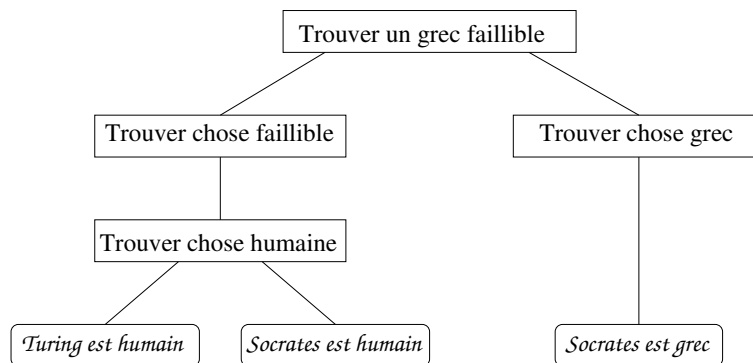


Figure 1. Résolution hâtive d'un problème présenté sous forme ET-OU

Cas 2 : Problème de pénurie

La base de faits est la suivante :

F1: Une voiture coute 5000\$

F2: Un yacht coute 5000\$

F3: John a 5000\$

La base de règle est :

R1: Pour séduire une actrice →

- acheter une voiture
- acheter un yacht

Montrer que John peut séduire une actrice

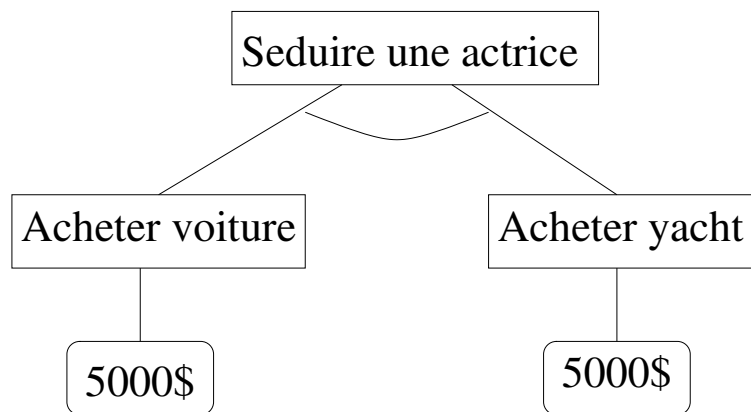


Figure 2. Résolution hâtive d'un problème présenté sous forme ET-OU

7 Algorithmes de jeu

On peut dire que les algorithmes de jeu sont le domaine de prédilection de l'IA.

On veut générer de “bons” mouvements, pour cela, on dispose d'une fonction d'évaluation, et on est limité par une profondeur limite de recherche.

Pour les jeux à 1 joueur, on utilise A^* .

Pour les jeux à 2 joueurs, on modélise le problème sous la forme d'un graphe ET-OU.

Exemple:

Soient JE (l'ordinateur) et LUI (l'adversaire).

- un sommet terminal correspond à la configuration gagnante
- tout autre sommet extrême correspond à une configuration perdante
- un sommet OU correspond à une configuration à partir de laquelle JE va jouer
- un sommet ET correspond à une configuration à partir de laquelle LUI va jouer

7.1 Algorithme minimax

La fonction d'évaluation définie pour JE. JE cherche à Maximiser la fonction d'évaluation (MAX). LUI est supposé chercher à la Minimiser (MIN) (coups défavorables à JE)

$f(P)$: valeur de la fct d'évaluation si P est un noeud extrême

$F(P)$: valeur de retour associée à P si P est n'est pas extrémité

si P_1, P_2, \dots sont les successeurs directs de P (noeuds fils):

- $F(P) = f(P)$ si P est un noeud extrémité
- $F(P) = \max_i F(P_i)$ si P est un noeud MAX
- $F(P) = \min_i F(P_i)$ si P est un noeud MIN

7.2 Algorithme NEGMAX [Knuth & Moore 1975]

Le principe de cet algorithme consiste à éviter de traiter différemment les noeuds MIN et MAX.

- $F(n) = f(n)$ si n extrémité
- $F(n) = \max(-F(n_1), -F(n_2), \dots)$ sinon

7.3 Algorithme Alpha-Béta

Le gain de cet algorithme consiste à éviter de faire une recherche exhaustive jusqu'à la profondeur limite en trouvant des "coupures" dans l'arbre de recherche.

- $\alpha = -\infty$;
- $\beta = +\infty$;

```
int valeur(p, alpha, beta)
position p;
int alpha, beta;
{
    int i, t, nbre;

    /* trouver les coups possibles à partir de P: P1, P2... */

    trouver_succeurs(...);
    nbre=nbre_succeurs(...);

    if(nbre==0) return f(p);
    for(i=0;i<nbre;i++)
    {
        t= - valeurs(Pi, -beta, -alpha);
        if(t>alpha) alpha=t;
        if(alpha>=beta) return alpha;
    }

    return alpha;
}
```


7.4 Améliorations possibles des algorithmes de jeu

- Mettre dans le “bon” ordre les successeurs (heuristique)
- Bibliothèque pour le début de partie
- L'évaluation peut aussi tenir compte du nombre de bons coups possibles à partir de la position courante \Rightarrow Evaluation confirmée

ATTENTION : problèmes liés au grands facteurs de branchement: (jeu de GO, facteur = 200)

Chapitre 3

Systemes experts

1 Introduction aux systemes experts

L'idée d'un systeme expert est de "reproduire un raisonnement".

Phases de conception:

- Analyse: comprendre les mecanismes de raisonnement
- Synthèse

Les composants d'un systeme experts sont :

- la base de regles
- la base de faits
- le moteur d'inférence

Phase d'évaluation:

- restriction (choix des faits pour une regle)
- filtrage (pattern matching)
- résolution de conflits (choix des regles déclenchées)

Un moteur d'inférence possède plusieurs modes de fonctionnement:

- Irrévocable
- Tentatives (back tracking)
- chaînage avant

- chaînage arrière (exploration d'un arbre ET-OU)
- chaînage mixte

Fonctionnement monotone d'un moteur d'inférence :

Un système expert est monotone si et seulement si :

- aucun fait ne peut être supprimé de la Base de Faits
- aucun fait ajouté n'introduit de contradictions dans la BdF

Système non monotones :: langages IA avec des instructions pour supprimer ou inhiber des faits...

Lors de l'analyse d'un problème les questions à se poser sont les suivantes ::

- Le problème est-il décomposable ?
- Les règles sont-elles réversibles ?
- Cherche-t-on une "bonne solution" ou la "meilleure solution" ?
- La base de données est-elle cohérente ?
- Toutes les règles sont-elles nécessaires tout le temps ?
- L'aide d'un humain est-elle disponible ?

2 Exemple de système expert

- a) si fleur \wedge graine alors phanérogame
- b) si phanérogame \wedge graine nue alors sapin
- c) si phanérogame \wedge 1-cotylédone alors monocotylédone
- d) si phanérogame \wedge 2-cotylédone alors dicotylédone
- e) si monocotylédone \wedge rhizome alors muguet
- f) si dicotylédone alors anémone
- g) si monocotylédone \wedge \neg rhizome alors lilas
- h) si feuille \wedge fleur alors cryptogame
- i) si cryptogame \wedge \neg racine alors mousse
- j) si cryptogame \wedge racine alors fougère
- k) si \neg feuille \wedge plante alors thallophyte
- l) si thallophyte \wedge chlorophylle alors algue
- m) si thallophyte \wedge \neg chlorophylle alors champignon
- n) si \neg feuille \wedge \neg fleur \wedge \neg plante alors colibacille

déterminer la plante ayant les caractéristiques suivantes: rhizome, fleur, graine, 1-cotylédone

2.1 Utilisation des systèmes experts

Système “d’aide à”:

- Programmation
- Diagnostic MYCIN (1976): infections bactériennes du sang
- Conception ou Fabrication R1 (1981): configuration d’un système informatique
- Planification (grands projets, emplois du temps...)

Résolution de problèmes:

- DENDRAL (Feigenbaum 1971): donne la formule développée d'un cors organique à partir de sa formule brute et de son spectrographe de masse.
- AM (Lena 1979): "Découvre" des concepts mathématiques

Enseignement Assisté par Ordinateur (LOGO...)

Jeux:

- Echecs, poker, bridge, dames...

Simulation:

- Physique qualitative

3 Problèmes rencontrés par les systèmes experts

- Problèmes sur les applications de taille "réelle":
 - Incomplétude de l'expertise
 - Erreurs dans les règles
 - Inconsistance des règles entre elles
- Comment modéliser le sens commun ?
- Peut on réduire l'expertise à des règles?
 - tour de main, savoir faire...
- explosion combinatoire: problèmes NP-complets
 - nécessité d'heuristiques

Chapitre 4

Logiques formelles

Un système formel est constitué :

1. D'un alphabet fini de symboles
2. D'un procédé de construction des mots du système formel
3. D'un ensemble d'axiomes (qui sont des mots)
4. D'un ensemble fini de règles de déduction

Une preuve est une suite finie de mots M_1, M_2, \dots, M_r dans laquelle chaque M_i est, soit un axiome, soit se déduit des mots précédents.

Un théorème est un mot t , tel qu'il existe une preuve de $M_r \equiv t$.

Tout axiome est un théorème.

Il a été montré que tout théorème est une tautologie (et vice versa), c.a.d. que toute interprétation¹ de ce théorème est valide.

¹modification de la valeur de vérité de ses variables

1 Logique des propositions (logique d'ordre 0)

1. Alphabet :

- lettres propositionnelles = p,q,r,s,t,...;
- opérateurs logiques = \neg , \rightarrow
- parenthèses

2. Formation des mots :

- une lettre propositionnelle est un mot
- si m est un mot alors (m) est un mot
- si m est un mot alors $\neg m$ est un mot
- si m_1 et m_2 sont des mots alors $m_1 \rightarrow m_2$ est un mot

3. Axiomes :

- $\alpha : m_1 \rightarrow (m_2 \rightarrow m_1)$
- $\beta : (m_1 \rightarrow (m_2 \rightarrow m_3)) \rightarrow ((m_1 \rightarrow m_2) \rightarrow (m_1 \rightarrow m_3))$
- $\gamma : (\neg m_2 \rightarrow \neg m_1) \rightarrow (m_1 \rightarrow m_2)$

4. Règle de déduction unique : (modus ponens)

- Si m_1 et $(m_1 \rightarrow m_2)$ sont des théorèmes, alors on en déduit (m_2) . Soit formellement : (m_1) et $(m_1 \rightarrow m_2) \rightarrow m_2$

1.1 Exemples de démonstration

$F \rightarrow F$:

1. $F \rightarrow (F \rightarrow F)$ (axiome α)
 2. $F \rightarrow ((F \rightarrow F) \rightarrow F)$ (axiome α)
 3. $(F \rightarrow ((F \rightarrow F) \rightarrow F)) \rightarrow ((F \rightarrow (F \rightarrow F)) \rightarrow (F \rightarrow F))$ (axiome β)
 4. $(F \rightarrow (F \rightarrow F)) \rightarrow (F \rightarrow F)$ (MP(2,3))
 5. $F \rightarrow F$ (MP(1,4))
-

$F \rightarrow G$ et $G \rightarrow H$:

1. $F \rightarrow G$
 2. $G \rightarrow H$
 3. $(F \rightarrow (G \rightarrow H)) \rightarrow ((F \rightarrow G) \rightarrow (F \rightarrow H))$ (axiome β)
 4. $(G \rightarrow H) \rightarrow (F \rightarrow (G \rightarrow H))$ (axiome α)
 5. $F \rightarrow (G \rightarrow H)$ (MP(2,4))
 6. $(F \rightarrow G) \rightarrow (F \rightarrow H)$ (MP(5,3))
 7. $F \rightarrow H$ (MP(1,6))
-

forme conjonctive:

$$(a \vee b \vee c) \wedge (d \vee e \vee f) \dots \wedge (\dots \vee \dots)$$

Forme disjonctive:

$$(a \wedge b \wedge c) \vee (d \wedge e \wedge f) \dots \vee (\dots \wedge \dots)$$

Passer d'une expression à une autre:

$$\neg \neg p = p$$

$$p \equiv q = \neg p \vee q$$

$$p \vee q = \neg (\neg p \wedge \neg q)$$

$$p \wedge q = \neg (\neg p \vee \neg q)$$

$$p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$$

$$p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$$

1.1.1 Propriétés

La logique des propositions est :

- *non-contradictoire* : t et $\neg t$ ne sont pas simultanément dérivables
- *complète* : les théorèmes sont des tautologies
- *décidable* : il est possible de dire en un temps fini si une phrase est un théorème ou non.

En fait, il suffit de choisir une interprétation donnée et démontrer la véracité d'un théorème selon cette interprétation. L'une des interprétations les plus simples est celle de l'algèbre de Boole. Pour démontrer un théorème il suffit donc que sa table de vérité soit toujours vraie.

1.2 Le calcul des prédicats (logique du premier ordre)

On aimerait pouvoir exprimer le fameux syllogisme de Socrates²

Tous les hommes sont mortels	majeure
Or Socrates est un homme	mineure
Donc Socrates est mortel	conclusion

²A ne pas confondre avec le faux syllogisme de Prevert :

Tous les hommes sont mortels
Or les chats sont mortels
Donc Socrates est un chat

Or, avec la logique des propositions, on ne peut malheureusement pas exprimer une assertion du type :

Tous les hommes sont mortels

C'est pourquoi la logique du premier ordre a été introduite par Frege. Elle reprend l'ensemble des éléments de la logique propositionnelle et ajoute des nouveautés :

1. Alphabet :

- constantes = A,B,C,D,E,...;
- variables = t,u,v,w,x,y,z,...;
- prédicats = Π, Λ, Γ
- opérateurs logiques = \neg, \rightarrow
- quantificateurs = \forall, \exists

2. Formation des mots : (LP) +

- On affect à chaque prédicat un poids k
- $\Lambda(x_1, x_2, \dots, x_k)$ est un mot ssi le poids de Λ est k
- $(\forall x_1) \Lambda(x_1, x_2, \dots, x_k)$ est un mot pour lequel x_1 est une variable liée et x_i sont des variables libres

3. Axiomes : (LP) +

- $\delta : (\forall t) \Gamma(t) \rightarrow \Gamma(A)$ (particularisation)
- $\epsilon : (\forall t) (m_1 \rightarrow m_2) \rightarrow (m_1 \rightarrow (\forall t) m_2)$
 . (attention : t ne doit pas être une variable libre dans m_1)

4. Règles de dérivation :

- (m_1) et $(m_1 \rightarrow m_2) \rightarrow m_2$ (modus ponens)
- $m_1 \rightarrow (\forall t) m_1$ (t est une variable libre dans m_1) (généralisation)

1.3 Propriétés

- Gödel a montré la *complétude* : les théorèmes sont vrais dans toutes les interprétations
- Malheureusement, Church a montré que le calcul des prédicats est *indécidable* c.a.d. qu'il est impossible de construire une procédure permettant de séparer les théorèmes des non-théorèmes (non-énumérables³).

³*Je ne suis pas un théorème*

1.4 Méthodes de résolution

1.4.1 Herbrand [Herbrand 1930, Robinson 1965]

Théorème 1 *Pour montrer un théorème T*

$$H_1 \cap H_2 \cap \dots \cap H_n \rightarrow C$$

il peut être plus facile de démontrer que

$$H_1 \cap H_2 \cap \dots \cap H_n \cap \neg C$$

est insatisfiable ou contradictoire (preuve par l'absurde)

Théorème 2 Herbrand (1930) : *Une C.N.S. pour qu'un ensemble \mathcal{N} de clauses soit insatisfiable est qu'il existe un sous-ensemble de formules contradictoires de \mathcal{N} complètement instanciées.*

L'univers de Herbrand est l'ensemble $G(\mathcal{N}) = \bigcup G_i$ avec G_0 l'ensemble des constantes de \mathcal{N} , G_i est obtenu en appliquant toutes les fonctions $f_k(t_1, \dots, t_k)$ qui sont des formules appartenant au vocabulaire de \mathcal{N} et les t_i sont des termes de G_0 .

Le principe de résolution est de faire croître l'univers d'Herbrand jusqu'à ce qu'il soit insatisfiable.

1.4.2 Unification

Les règles :

$$P(A)$$

$$P(x) \rightarrow Q(x)$$

Peuvent être mises en correspondance en substituant x par A : $P(A) \rightarrow Q(A)$.

1.4.3 Diminution

La règle de diminution s'applique quand l'application d'un unificateur σ sur une clause de la forme $\{l, m, n\}$ permet de réduire l'ensemble des clauses à $\{[l]\sigma, [n]\sigma\}$.

exemple 1 :

$$\text{Résoudre: } P(x, C), R(x), \neg P(C, C), Q(y) = (C|x)$$

$$\text{clause résolvante: } R(C), Q(y)$$

exemple 2:

$$\text{Résoudre: } P(x, f(x)), \neg P(A, y), R(f(y)) = (A|x), (f(A)|y)$$

$$\text{clause résolvante: } R(f(f(A)))$$

1.4.4 Stratégie de saturation simple

M est conséquence de d'un ens de clauses N si $\neg M \wedge N$ débouche sur la clause NIL. $\neg M \wedge N$ est appelé ensemble de base.

Résolution :

Clause \leftarrow ensemble de base

Répéter

sélection de 2 clauses dans CLAUSE

calculer le résolvant M de ces 2 clauses

CLAUSE \leftarrow CLAUSE $\wedge M$

Jusqu'à NIL appartient à CLAUSE

En s'y prenant de cette manière, on est rapidement confronté à un problème d'explosion combinatoire !

1.4.5 Stratégie de saturation avec simplification

- éliminer de CLAUSE les tautologies
- éliminer les clauses incluses dans d'autres clauses (subsumer)

1.4.6 Arbre de dérivation

résolution 2 par 2

stratégies générales

- Largeur d'abord : stratégie complète
- Profondeur d'abord : non complète à cause de la limite sur la profondeur

Stratégie linéaire

Chaque résolvant à un de ses parents qui appartient à l'ensemble de base. Si on peut la mettre sous la forme $M \wedge N$ où M est une clause qui ne comporte que des littéraux négatifs et N que des clauses de Horn (1 seul littéral positif). On montre que la stratégie est complète si le premier résolvant à M dans l'un de ses parents.

Stratégie utilisant un ensemble de support

Prouver M (clause objectif) est conséquence de N

Le 1er résolvant F_0 a ses 2 parents dans l'ensemble de base $\neg M \wedge N$ et l'un de ses parents est $\neg M$. Tout résolvant F_i est obtenu à partir de clauses dont l'une est F_{i-1} .

L'ensemble des descendants de la clause objectif est appelé ensemble de support. Cet algorithme est complet avec un mécanisme d'exploration en largeur d'abord

1.4.7 Extraction de réponses

- M (clause objectif) est conséquence de N ?
- On utilise la stratégie avec ensemble de support.
- On cherche à obtenir NIL avec $\neg M \wedge N$.
- On associe M à la clause objectif (tautologie)
- On applique la règle de résolution comme avant (substitutions)
- On obtient une clause qui est la réponse (qui découle logiquement de N)

Exemple :

Si Médor suit Jean partout,

si Jean est à l'école

où est Médor ?

Traduction : $\forall X [lieu(jean, X) \Rightarrow lieu(medor, X)]$

Si lieu(jean, ecole)

? $(\forall X) [lieu(medor, X)]$

suppression de l'implication : $\neg lieu(jean, X) \wedge lieu(medor, X)$

lieu(jean, ecole)

Résolution de la négation de la question :

$\neg(\forall Y) [lieu(medor, Y)] = (\exists Y) \neg lieu(medor, Y)$

Clause objectif : $\neg lieu(medor, Y)$

1.5 Autres logiques

On peut citer les logiques :

- déontique, flou, des impératifs, enductive, des interrogatifs, modale, non-monotones, de la pertinence, temporell,...

Chapitre 5

Logique floue

La logique floue a été introduite pour pallier certaines difficultés liées à l'utilisation d'interprétation binaires dans la logique des prédicats. Par ailleurs elle permet de s'affranchir en partie des incohérences ou inconsistance de l'expertise.

La logique floue permet d'exprimer des concepts du type :

Si la température est très élevée, alors refroidir beaucoup

1 Ensembles flous

Dans un ensemble classique, un élément appartient à un ensemble si fonction d'appartenance binaire vaut 1.

$$\text{Ex : } E = \{x \in \mathbb{N} \mid x = 2 \times k, k \in \mathbb{N}\}$$

Dans un ensemble flou, la fonction d'appartenance prend ses valeurs dans $[0, 1]$. Ainsi, un élément appartient plus ou moins au "concept" que représente l'ensemble. Un ensemble flou est ainsi défini par l'ensemble des valeurs de la fonction d'appartenance sur le domaine de définition (discret ou continu).

Ex : Le concept de "jeune" pourrait être défini par l'ensemble flou discret $A = \{1/10, 0.8/20, 0.6/30, 0.2/40, 0.1/50, 0/60, 0/70, 0/80\}$.

Ex : Le concept "chaud" peut être défini sur \mathbb{R} comme l'ensemble $C = \{x \in \mathbb{R} \mid f_C(x) = x\}$

1.1 Définitions

1.1.1 Support

Le support est défini comme l'ensemble des valeurs du domaine X pour lesquelles la fonction d'appartenance n'est pas nulle.

$$\text{supp}(A) = \{x \in X \mid f_A(x) \neq 0\}$$

1.1.2 Hauteur

La hauteur d'un ensemble flou A est la valeur max de la fonction d'appartenance sur le domaine X .

$$h(A) = \max_{x \in X} f_A(x)$$

1.1.3 Noyau

Le noyau d'un ensemble flou A est l'ensemble des valeurs pour lesquelles la fonction d'appartenance est égale à 1.

$$\text{kern}(A) = \{x \in X \mid f_A(x) = 1\}$$

1.1.4 Cardinal

Le cardinal d'un ensemble flou A est la somme des fonctions d'appartenance (cf. ensemble classique).

$$|A| = \text{Card}(A) = \sum_{x \in X} f_A(x)$$

1.1.5 Égalité

Deux ensembles flous A et B sont égaux si leurs fonctions d'équivalences sont égales pour toutes les valeurs du domaine X .

$$\forall x \in X : f_A(x) = f_B(x)$$

1.1.6 Inclusion $A \subseteq B$

Un ensemble flou A est inclus dans un ensemble flou B si toutes ses valeurs de fonction d'appartenance sont inférieures à celles de B sur tout le domaine X .

$$\forall x \in X : f_A(x) \leq f_B(x)$$

(Relation d'ordre partielle).

1.1.7 Intersection $A \cap B$

L'intersection C de 2 ensembles flous A et B est définie comme l'ensemble flou dont la fonction d'appartenance est le min des fonctions d'appartenance des ensembles A et B.

$$\forall x \in X : f_C(x) = \min(f_A(x), f_B(x))$$

1.1.8 Union $A \cup B$

L'union C de 2 ensembles flous A et B est définie comme l'ensemble flou dont la fonction d'appartenance est le max des fonctions d'appartenance des ensembles A et B.

$$\forall x \in X : f_C(x) = \max(f_A(x), f_B(x))$$

1.1.9 Complément

Le complément A^C d'un ensemble flou A est défini comme :

$$\forall x \in X : f_{A^C}(x) = 1 - f_A(x)$$

1.2 Propriétés

Fort de ces définitions, on retrouve les propriétés ensemblistes classiques :

- associativité de \cap et \cup
- commutativité de \cap et \cup
- $A \cap X = A, A \cap \emptyset = \emptyset$
- $A \cup \emptyset = A, A \cup X = X$
- $A \cup B \supseteq A \supseteq A \cap B$
- $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
- $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
- $|A| + |B| = |A \cap B| + |A \cup B|$
- Lois de DeMorgan :

$$(A \cap B)^C = A^C \cup B^C$$

$$(A \cup B)^C = A^C \cap B^C$$

- $(A^C)^C = A$
- $X^C = \emptyset$
- $\emptyset^C = X$
- $|A| + |A^C| = |X|$
- $kern(A^C)^C = supp(A)$
- $supp(A^C)^C = kern(A)$
- idempotence

$$A \cap A = A$$

$$A \cup A = A$$

2 Logique floue

La logique floue est aussi appelée “logique linguistique” car ses valeurs de vérité sont des mots du langage courant : “plutôt vrai, presque faux...”. Cette valeur de vérité dépend du prédicat !

Quantificateurs

\forall, \exists , beaucoup, quelques, rarement...

Variables floues

C'est un ensemble flou : “Pierre est jeune, Thomas est vieux...”

Variables linguistiques

Elles sont constituées par un triplet (X, U, T_X) .

X: désignation (nom de variable)

U: référentiel

T_X : valeurs linguistiques

Ex : taille d'un être humain.

$U = \{80, 90, 100, 110, 120, 130, \dots, 200\}$

T_X : très petit, petit, moyen, grand, très grand.

X="très petit" = $A_1 = \{1/80, 1/90, 0.8/100, 0/110\dots\}$

Les modificateurs

Il permettent de déterminer l'ensemble flou correspondant à une modification sur l'ensemble flou de départ :

très : $\mu_A^2(x) = (\mu_A(x))^2$ (concentration).

plus ou moins : $\mu_A^{\frac{1}{2}}(x) = \sqrt{\mu_A(x)}$

non : $\mu_{1-A}(x) = 1 - \mu_A(x)$

hautement : $\mu_{h(A)}(x) = (\mu_A(x))^3$

intensément, plutôt...

Ex : A="jeune", A^2 ="très jeune", $A^{\frac{1}{2}}$ ="plus ou moins jeune", $1-A$ ="non-jeune".

- $A = \{1/10, 0.81/20, 0.36/30, 0.25/40, 0.16/50, 0/60, 0/70\}$
- $A^2 = \{1/10, 0.65/20, 0.13/30, 0.07/40, 0.02/50, 0/60, 0/70\}$
- $A^{\frac{1}{2}} = \{1/10, 0.9/20, 0.6/30, 0.5/40, 0.4/50, 0/60, 0/70\}$
- $1 - A = \{0/10, 0.19/20, 0.64/30, 0.75/40, 0.84/50, 1/60, 1/70\}$

2.1 Proposition floue

Les propositions floues sont des valeurs de vérités de la logique floue.

Ex : "La charge est lourde", "Il fera probablement beau demain"

2.1.1 Proposition floue simple

Du type "X est A".

2.1.2 Proposition floue composée

Ce sont 2 propositions floues simples connectées par un "ET" ou un "OU".

2 cas sont alors possibles :

- 1 Le référentiel est le même : "X est A ET B" ou "X est A OU B". "Jim n'est pas grand et John est de taille moyenne"
- 2 Les 2 sujets sont différents : "(X,Y) est $A \times B$ " ou "(X,Y) est $(A \times V) OU (U \times B)$ ". "La voiture est grande et la chaussée est étroite".

2.1.3 Implications : règles floues

C'est implications représentent la base du raisonnement flou. Elles sont du type :

“X est A” \rightarrow “Y est B”.

Ex : “Si l’homme est jeune alors il est fort”.

Le problèmes est de savoir comment inférer la règle si “X est A” à un certain degré seulement. En effet, dans ce cas, la valeur de vérité $\mu_A(x)$ correspondant à “X est A” n’exprime qu’un degré de vérité de cette proposition.

On aimerait pouvoir disposer en logique floue des mêmes relations d’inférence qu’en logique formelle.

Ainsi, si la règle est “si X est A alors Y et B” et que X est $A' \simeq A$ on aimerait déduire que Y est $B' \simeq B$.

On définit la valeur de vérité de l’implication comme :

$$\mu_{A \rightarrow B}(x, y) = \min(\mu_A(x), \mu_B(y))$$

NB : toute autre co-norme triangulaire est utilisable (ex: $\min(1, 1 - \mu_A(x) + \mu_B(y))$, $\mu_A(x) \cdot \mu_B(x)$).

2.1.4 Modus ponens généralisé

On définit donc le modus ponens généralisé.

$$\mu_{B'} = \max_{x \in U, y \in V} (\mu_{A'}, \mu_{A \rightarrow B}(x, y))$$

2.1.5 Applications

- Automatisme
- Robotique
- Informatique
- Prise de décision
- Gestion
- Médecine
- Reconnaissance des formes...

2.1.6 Mise en forme

La mise en forme d'un problème en logique floue nécessite l'identification d'un univers intérieur dans lequel peuvent s'appliquer les règles d'inférences floues et un univers extérieur lié aux entrées sorties du systèmes.

On définit ainsi :

- Les Entrées/Sorties “vulgaires” (capteurs et effecteurs du système)
- Les caractéristiques floues (valeurs de vérité : “chaud, grand...”)
- L'univers du discours (intervalles de ces valeurs)
- Les fonctions d'appartenances
- Les règles d'inférences floues

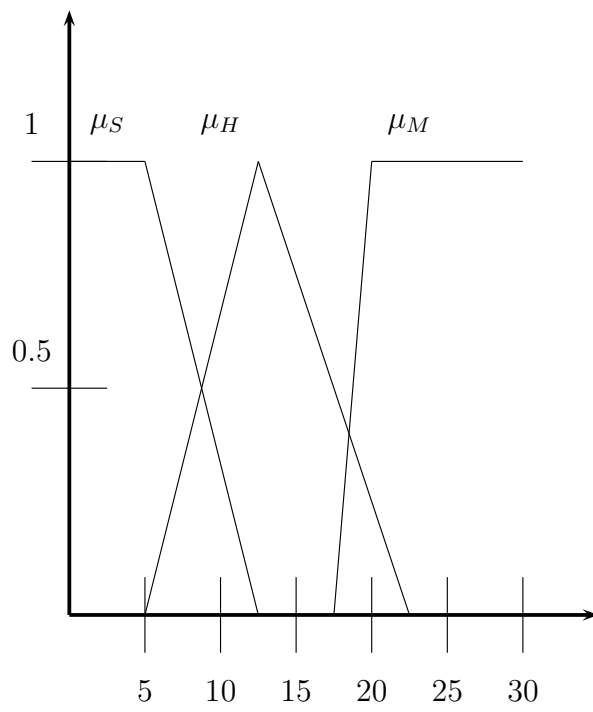
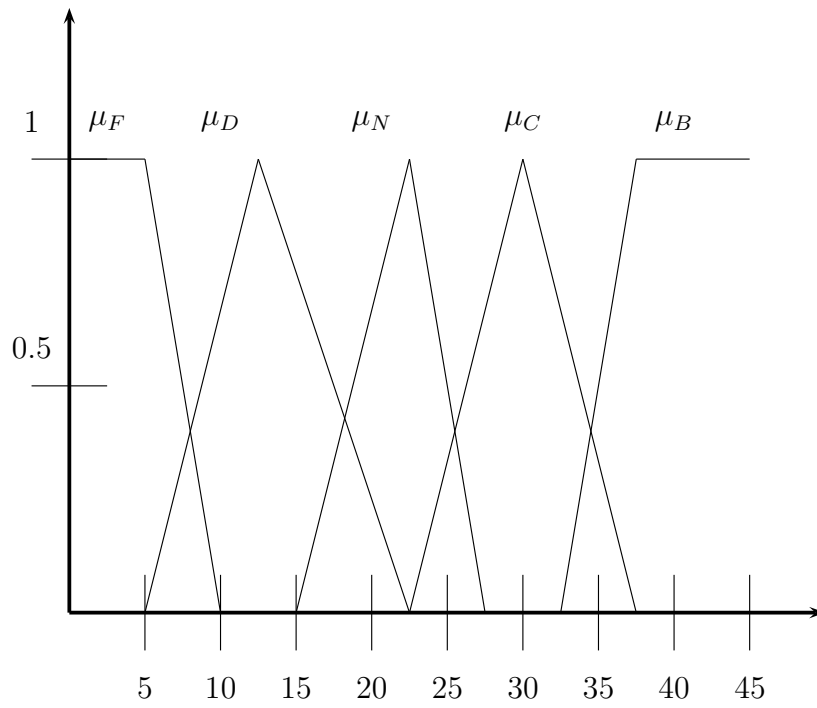
Les grandes étapes pour passer du monde réel au monde interne sont, l'étape de “fuzzification” (passage des entrées vulgaires aux entrées floues) et l'étape de “défuzzification” (passage des sorties floues aux sorties vulgaires).

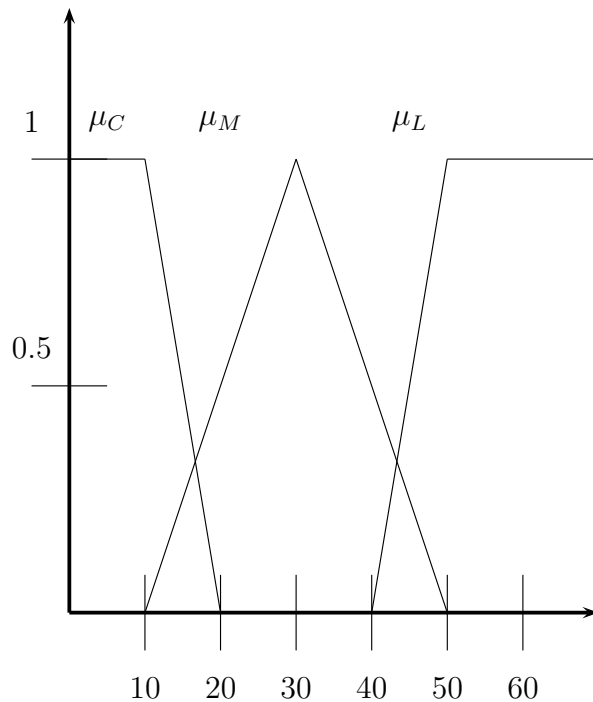
3 Exemple : système d'arrosage

On se propose de construire un contrôleur d'arrosage flou.

Les variables de vérité de ce système, en ce qui concerne les entrées sont la température et l'humidité alors que la sortie peut être réduite à la durée d'arrosage.

Les fonctions d'appartenances sont données par les figures ci-dessous.





Les règles d'inférence sont les suivantes :

- R1 Si la température est brulante ET le sol sec ALORS la durée d'arrosage est longue
- R2 Si la température est chaude ET le sol humide ALORS la durée d'arrosage est moyenne
- R3 Si la température est chaude ET le sol sec ALORS la durée d'arrosage est longue
- R4 Si la température est brulante ET le sol humide ALORS la durée d'arrosage est moyenne
- R5 Si la température est froide ET le sol humide ALORS la durée d'arrosage est courte
- R6 Si la température est froide ET le sol mouillé ALORS la durée d'arrosage est courte
- R7 Si la température est douce ET le sol sec ALORS la durée d'arrosage est longue
- R8 Si la température est normale ET le sol humide ALORS la durée d'arrosage est moyenne
- R9 Si la température est normale ET le sol sec ALORS la durée d'arrosage est longue
- R10 Si la température est douce ET le sol mouillé ALORS la durée d'arrosage est courte

Fuzzification

La température relevée est de 35° et le degré d'hygrometrie est de 10%.

On en déduit les valeurs de fonction d'appartenance :

Pour la température :

- $\mu_F(35) = 0, \mu_D(35) = 0, \mu_N(35) = 0$
- $\mu_C(35) = 0.2$

- $\mu_B(35) = 0.45$

Pour l'hygrométrie :

- $\mu_S(10) = 0.35$
- $\mu_H(10) = 0.61$
- $\mu_M(0) = 0$

Les seules règles activées sont celles dont les fonction d'appartenance de chacun des prémisses est non-nul.

On calcule alors la valeur min des prémisses déclencheurs :

$$R1 \min(\text{Brulante,Sec})=\min(0.45,0.37)=0.35 \rightarrow \text{Longue}$$

$$R2 \min(\text{Chaude,Humide})=\min(0.2,0.61)=0.2 \rightarrow \text{Moyenne}$$

$$R3 \min(\text{Chaude,Sec})=\min(0.2,0.35)=0.2 \rightarrow \text{Longue}$$

$$R4 \min(\text{Brulante,Humide})=\min(0.45,0.61)=0.45 \rightarrow \text{Moyenne}$$

La portée de la conclusion est définie comme la portée maximum du système de déduction :

$$\max(0.37,0.2)=0.37 \rightarrow \text{Longue}$$

$$\max(0.2,0.45)=0.45 \rightarrow \text{Moyenne}$$

En ce qui concerne la défuzzification (et donc l'élaboration de la commande réelle), plusieurs méthodes existent.

1. Soit on prend la valeur max
2. Soit on effectue une moyenne entre les valeur moyennes de chaque classe, pondérée par la valeur des fonctions d'appartenance.
3. Soit on calcule le centre de gravité de chaque classe, puis de prendre la valeur moyenne pondérée de ces centres de gravités.

Chapitre 6

Neurone biologique et formalisme

1 Le neurone biologique

A la suite des observations de l'anatomiste espagnole Ramòn y Cajal, dès la fin du 19^{ème} siècle, on a pu déterminer que le cerveau était composé de cellules distinctes appelées *neurones* formant un ensemble dense d'environ 10 à 100 milliards d'unités interconnectées[Rosenzweig and Leiman, 1992]. La principale caractéristique de ces neurones est qu'ils permettent de véhiculer et de traiter des informations en faisant circuler des messages électriques dans le réseau ténu et massivement parallèle formé par leur *axone*¹. L'extrémité des axones se divise en une multitude de ramifications. A l'inverse, les arborescences qui amènent l'information vers le corps cellulaire sont appelés *dendrites*. Les informations sont transmises d'un neurone à l'autre, de manière unidirectionnelle, par l'intermédiaire de points de jonction appelés *synapses*. En moyenne le nombre de connexions entre neurones est de 10^4 . Le schéma d'un neurone réel est proposé figure 1.

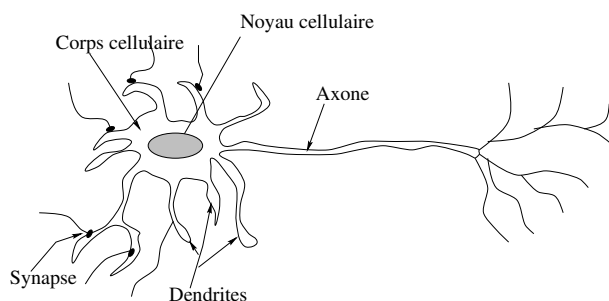


Figure 1. Représentation schématique d'un neurone biologique.

L'activité d'un neurone se mesure en fonction de la fréquence² du train de *poten-*

¹L'axone est le prolongement terminal du corps du neurone. Ils peuvent atteindre jusqu'à un mètre de long.

²Cette fréquence est inférieure à 300 Hz ce qui est bien sur très lent comparé aux capacités des circuits électroniques modernes.

tiels d'actions (ou *spikes*) propagés sur l'axone. Cet *influx nerveux*, lorsqu'il parvient à une synapse, permet de libérer des *neuromédiateurs* qui excitent (neuromédiateurs excitateurs) ou inhibent (neuromédiateurs inhibiteurs) le neurone suivant et peuvent ainsi générer ou interdire la propagation d'un nouvel influx nerveux. Une caractéristique fondamentale de la synapse est qu'elle est capable de *s'adapter* et ainsi faciliter ou non le passage des influx nerveux. Cette plasticité est à l'origine des mécanismes d'apprentissage.

L'influx nerveux se caractérise par une impulsion électrique (Potentiel d'Action — PA) se déplaçant le long de l'axone depuis le corps cellulaire jusqu'aux synapses. Le déplacement du PA est généré par une diminution locale de la différence de potentiel entre l'intérieur et l'extérieur de la membrane de la cellule qui provoque l'ouverture des canaux sodiques de la membrane laissant entrer les ions sodium (Na^+). Le potentiel ayant toujours tendance à revenir à une valeur d'équilibre, la cellule libère des ions potassium (K^+). C'est ainsi que de proche en proche se propage l'influx électrique jusqu'aux synapses.

Une synapse est constituée d'un bouton présynaptique situé en prolongement de l'axone du neurone afférent et d'une partie réceptrice sur le corps du neurone efférent, les deux étant séparés par un espace extra-cellulaire très faible appelé *fente synaptique*. Le bouton pré-synaptique contient des *vésicules synaptiques* remplies de transmetteurs chimiques. Lorsque le PA parvient à la synapse, il fait rentrer des ions calcium (Ca^{2+}) dans le corps cellulaire du neurone. Cela provoque l'alignement d'une partie des vésicules synaptiques le long de la membrane du bouton synaptique et donc la libération des neurotransmetteurs³ dans la fente synaptique. Les molécules transmettrices libérées arrivent sur le site récepteur du neurone efférent et rentrent en interaction avec les protéines réceptrices situées sur ces sites. La réaction résultante peut être dépolarisante (synapse excitatrice) ou hyperpolarisante (synapse inhibitrice) selon les affinités entre le neurotransmetteur et la protéine réceptrice. Les activités polarisantes ou dépolarisantes sont ensuite sommées par le neurone ce qui déclenche ou non un nouvel influx nerveux dans le neurone efférent, selon qu'un seuil de dépolarisation est franchi ou non.

1.1 Modèle du neurone formel

Dans la plupart des modèles formels, on représente l'activité du neurone par une grandeur analogique qui s'identifie à la fréquence d'émission des potentiels d'action sur l'axone. On ne tient donc pas compte de l'aspect séquentiel de la propagation de l'information dans les neurones biologiques. Dans la majorité des cas, ce modèle est suffisant.

Considérons le modèle de neurone formel présenté figure 2.

Soit I_i l'entrée d'un neurone (activité du neurone précédent).

Soit W_{ij} le poids synaptique associé à la synapse liant le neurone i au neurone j .

Soit O_j la sortie du neurone j .

³Il existe de nombreux neurotransmetteurs mais nous ne les détaillerons pas ici.

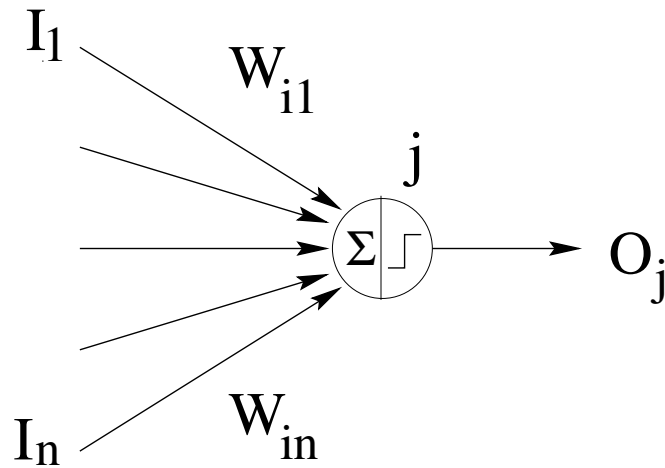
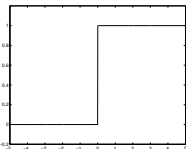
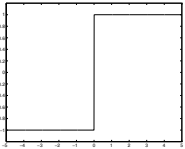
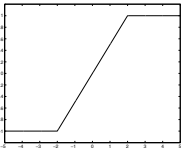
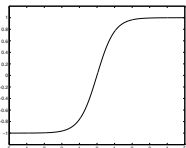


Figure 2. Schéma d'un neurone formel

On définit aussi,

- Le potentiel : $Pot_j = \sum W_{ij} \cdot I_i - \theta$
- La fonction d'activation (ou fonction de sortie) : $O_j(t) = f(Pot_j)$

Les entrées et sorties sont soit binaires soit réelles. La fonction de sortie est une fonction non linéaire plus ou moins proche de la sigmoïde. Les principales fonctions de sortie sont :

- la fonction de Heaviside : $H(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$ 
- la fonction signe : $Sign(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ -1 & \text{si } x < 0 \end{cases}$ 
- les fonctions linéaires à seuils : $f(x) = \begin{cases} 1 & \text{si } x > a \\ \frac{1}{a}x & \text{si } x \in [-a, a] \\ -1 & \text{si } x < -a \end{cases}$ 
- les fonctions sigmoïdes : $f(x) = \tanh(kx)$ ou $\frac{1}{1+e^{-kx}}$ 

- les fonctions à base radiale (Radial Basis Functions - RBF) :

$$f(x) = \exp\left(-\left(\frac{x}{\sigma}\right)^2\right), f(x) = \frac{1}{1 + \frac{\|x-t\|^2}{\sigma_2}}, \dots$$

La modélisation des phénomènes de plasticité synaptique ont, par ailleurs, permis de développer des modèles de neurone formel expliquant certains des mécanismes d'apprentissage. L'un des premiers essais de modélisation de l'apprentissage associatif, a été réalisé par Hebb [Hebb, 1949] dès 1949. La règle qu'il a énoncé s'exprime ainsi :

“Quand une cellule A excite par son axone une cellule B et que, de manière répétée et persistante, elle participe à la genèse d'une impulsion dans B, un processus de croissance ou un changement métabolique a lieu dans l'une ou dans les deux cellules, de telle sorte que l'efficacité de A à déclencher une impulsion dans B est, parmi les autres cellules qui ont cet effet, accrue.”

En utilisant les notations propres au neurone formel, on peut traduire cet énoncé sous la forme suivante :

$$W_{ij}(t + 1) = W_{ij}(t) + \varepsilon \cdot I_i \cdot O_j \tag{1}$$

Cette règle est à la base de nombreux mécanismes d'apprentissage, et en particulier à l'origine des modèles que nous présenterons dans les paragraphes suivants.

Chapitre 7

Réseaux de neurones supervisés

1 Mémoire associative linéaire

Le réseau de neurones comporte p entrées ($x_i \in [1, p]$) entièrement connectées à n sorties ($y_k \in [1, n]$). Pour simplifier la notation, on utilise une représentation matricielle.

La mémoire associative fonctionne en 2 phases :

- 1 une phase d'apprentissage, pendant laquelle les poids ont des valeurs modifiables
- 2 une phase de rappel, pour laquelle les connexions sont figées

1.1 Apprentissage

La phase d'apprentissage consiste à faire apprendre une association entre un vecteur \mathbf{x}^r et un vecteur \mathbf{y}^r . A chaque présentation d'une association, on propose de modifier les poids suivant la règle :

$$\Delta \mathbf{W} = \mathbf{y}^r \cdot (\mathbf{x}^r)^T$$

Après une présentation de tous les exemples, la matrice de poids vaut :

$$\mathbf{W} = \sum_{r=1}^N \mathbf{y}^r \cdot (\mathbf{x}^r)^T$$

La matrice est égale à la somme des matrices de corrélations des vecteurs. Par ailleurs, il faut noter que les associations sont apprises de manière diffuse dans la matrice (on dit aussi délocalisée ou distribuée).

1.2 Rappel

La matrice des poids est maintenant fixée, on stimule le réseau avec un vecteur \mathbf{x} (clef) et on observe la sortie $\mathbf{y} = \mathbf{W} \cdot \mathbf{x}$. En remplaçant \mathbf{W} , on a :

$$\mathbf{y} = \mathbf{W} \cdot \mathbf{x} = \sum_{r=1}^N \mathbf{y}^r [(\mathbf{x}^r)^T \cdot \mathbf{x}]$$

Soit le produit scalaire entre la clef \mathbf{x} et les prototypes \mathbf{x}^r .

Si la clef est égale à un vecteur \mathbf{x}^j , présenté pendant l'apprentissage, et si tous les vecteurs \mathbf{x}^r sont orthogonaux, un seul produit scalaire est non-nul, celui qui correspond à $r = j$. On a alors :

$$\mathbf{y} = Norme^2(\mathbf{x}^j) \cdot \mathbf{y}^j = k \cdot \mathbf{y}^j$$

Que se passe-t-il en présence de bruit, par exemple si $\mathbf{x} = \mathbf{x}^j + \mathbf{b}$?

$$\mathbf{y} = k \cdot \mathbf{y}^j + \sum_{r \neq j} \mathbf{y}^r [(\mathbf{x}^r)^T \cdot \mathbf{b}]$$

Le premier terme correspond au rappel, le second à une erreur qui dépend :

- de l'énergie du bruit
- de la corrélation bruit/prototype
- du nombre N de prototypes à mémoriser

2 Réseaux de Hopfield

Le modèle de Hopfield est issu de la physique statistique (études des verres de spins).

Dans le formalisme de la physique, on considère un ensemble de "sites" possédant un spin (+1 ou -1). Les interactions T_{ij} entre 2 sites sont supposées symétriques ($T_{ij} = T_{ji}$) sans autoentretient ($T_{ii} = 0$). On étudie la relaxation du système.

L'élaboration d'un modèle neuronal inspiré de ce formalisme date de 1982. On identifie aisément les sites avec des neurones binaires, les interconnxions avec les poids synaptiques et la relaxation avec la convergence du réseau.

L'intérêt du formalisme de Hopfield est d'associer au réseau une grandeur homologue à une énergie.

2.1 Dynamique du réseau

On définit un potentiel local u_i pour chaque site i :

$$u_i = \sum_{k=1}^N T_{ik} v_k, k \neq i$$

avec $v_i = \text{Signe}(u_i)$

L'évolution dynamique du réseau est la suivante :

1. On tire au hasard un site i
 - Si $u_i > 0$ alors $v_i \leftarrow +1$
 - Si $u_i < 0$ alors $v_i \leftarrow -1$
2. on recommence en 1

Petit à petit, l'état du réseau se stabilise à des valeurs qui dépendent des états initiaux et de la matrice des poids.

Considérons en effet l'énergie du réseau. Soit I l'état du réseau constitué par l'ensemble des états élémentaires de chaque site : $I = (v_1, v_2, \dots, v_N)$. On peut associer à I une énergie totale $E(I)$ qui est égale à la demie somme des produits entre les potentiels locaux u_i et les états des sites v_i :

$$E(I) = -\frac{1}{2} u^t v = -\frac{1}{2} \sum_i u_i v_i = -\frac{1}{2} \sum_i \sum_k T_{ik} v_k v_i$$

Cette forme $E(I) = -\frac{1}{2} v^t T v$ est une énergie car T est symétrique. Comme v , T et u sont bornés, l'énergie E est bornée et évolue de manière à être minimale.

2.2 Evolution de l'énergie

Au temps t :

$$E(t) = -\frac{1}{2} v^t(t) T v(t)$$

Au temps $t + 1$:

$$E(t + 1) = -\frac{1}{2} v^t(t + 1) T v(t + 1) = -\frac{1}{2} [v^t(t) + \Delta v^t(t)] T [v(t) + \Delta v(t)]$$

D'où :

$$\Delta E = -\frac{1}{2} [v^t(t) T \Delta v + \Delta v^t T v(t) + \Delta v^t T \Delta v]$$

Comme à chaque pas, uniquement un site change d'état $\Delta v = (0, 0, \dots, \Delta v_i, \dots, 0)^t$:

$$v^t(t)T\Delta v = \Delta v_i \sum_j T_{ij}v_j$$

$$\Delta v^t T v(t) = \Delta v_i \sum_j T_{ij}v_j$$

$$\Delta v^t(t)T\Delta v = \Delta v_i T_{ii} \Delta v_i = 0, \text{ car } T_{ii} = 0$$

Donc $\Delta E = -\Delta v_i \sum_j T_{ij}v_j = -\Delta v_i u_i$

On peut étudier tous les cas possibles en fonction des valeurs de $u_i(t)$ et $v_i(t)$:

$v_i(t)$	-1	+1
$u_i(t) > 0$	$\Delta v_i = +2$	$\Delta v_i = 0$
$u_i(t) < 0$	$\Delta v_i = 0$	$\Delta v_i = -2$

Dans tous les cas, le produit est donc soit positif soit nul, donc, l'énergie décroît.

2.3 Application au cas des MA

On veut mémoriser des états I^k associés à des états de sites v_i^k . Construisons une fonction d'énergie associée :

$$F(I) = -\frac{1}{2} \langle v^k, v \rangle^2 = -\frac{1}{2} [\sum_k (\sum_i v_i^k v_i)^2]$$

Si les états I^k sont orthogonaux, on a $\sum_i v_i^k v_i^l = N \delta_{kl}$, où N est le nombre de sites et δ_{kl} le symbole de Kronecker.

$$F(I^l) = E_{\min} = \frac{-N^2}{2}$$

Comme la dynamique du réseau de Hopfield converge toujours vers un minimum d'énergie, nous avons $E = F(I)$:

$$E = -\frac{1}{2} [\sum_i v_i (\sum_j T_{ij}v_j)] = F(I) = -\frac{1}{2} [\sum_k (v_i^k v_i)^2]$$

En développant :

$$E = -\frac{1}{2} \sum_k [\sum_i \sum_j v_i^k v_i v_j^k v_j] = -\frac{1}{2} \sum_i v_i \sum_j (\sum_k v_i^k v_j^k) v_j$$

En identifiant, on trouve donc :

$$T_{ij} = \sum_k v_i^k v_j^k$$

Ou encore :

$$T = \sum_k v^k (v^k)^t$$

Ce qui correspond à la matrice de connexion d'une mémoire associative utilisant l'apprentissage de Hebb.

Le rappel d'un mémoire associative correspond donc à la minimisation d'une énergie.

3 Limitations des MA

En pratique, il est difficile de satisfaire la contrainte d'orthogonalité. Cependant, pour des tailles de réseaux importants (par exemple des images), la condition d'orthogonalité est presque vérifiée (on peut aussi pré-traiter les données de manière à les orthogonaliser). Ces mémoires peuvent alors être utilisées en mémoire *auto-associatives*. Dans ce cadre, les vecteurs d'entrée et de sortie sont égaux ($\mathbf{y} = \mathbf{x}$). La mémoire est alors utilisée pour retrouver un vecteur à partir d'un morceau de celui-ci (cf. exemple images).

On voit que les limitations de ce genre de réseaux de neurones sont importantes. C'est à mettre en rapport avec sa faible capacité de stockage (pour une dimension \mathbf{N} , le nombre de connexions est $\mathbf{N} \cdot (\mathbf{N} - 1)$, soit une capacité de $\frac{1}{\mathbf{N}-1}$).

Un autre point important est que, toute distorsion par translation, homothétie ou rotation empêche la conservation de l'orthogonalité des vecteurs.

Ceci montre la limitation des modèles linéaires à une couche.

4 Perceptron

Le Perceptron, mis au point par Rosenblatt dans les années 50, est l'un des plus simples classifieurs neuronaux. Il comporte une seule couche de neurone qui reçoivent comme vecteur d'entrée \mathbf{X} . Leur activité est donnée par :

$$\begin{aligned} Act_i &= \sum_i w_i x_i - \theta \\ y_i &= \text{signe}(Act_i) \end{aligned}$$

La règle d'apprentissage du Perceptron est la suivante :

$$w_i(t + 1) = w_i(t) + \varepsilon(y_i^d(t) - y_i(t))x_i(t)$$

Si les entrées sont linéairement séparables, l'algorithme converge vers un hyperplan séparant les entrées [Rosenblatt, 1957, Rosenblatt, 1958, Rosenblatt, 1962].

5 Adaline

Le réseau Adaline a été développé par Widrow [Widrow and Hoff, 1960]. Il est constitué d'un unique neurone effectuant la combinaison linéaire de ses entrées. Il s'agit en fait d'un Perceptron sans saturation des sorties.

La règle d'apprentissage de ce réseau consiste à minimiser l'erreur quadratique en sortie du réseau de neurone. La règle d'apprentissage est identique à celle du Perceptron, à la différence près que ce sont les entrées non-saturées qui sont prises en compte.

Par ailleurs, la règle d'apprentissage est celle du LMS que nous avons décrite précédemment.

6 Limitations

Les réseaux Adaline et Perceptron ne peuvent partitionner l'espace d'entrée que de manière linéaire. On peut ainsi écrire un réseau une couche qui fabrique un **OR**, un **AND** ou un **NOT**, mais il est impossible de fabriquer la fonction **XOR** à moins d'augmenter le nombre de couches.

La limitation est que pour résoudre un problème donné, on ne sait pas *a priori* le nombre de couches nécessaires.

7 Rétropropagation du gradient

Nous avons que les modèles Adaline et le Perceptron avaient de bonnes performances s'il s'agissait de traiter des problèmes linéairement séparables. L'idée de la rétropropagation consiste à utiliser une fonction de transfert non-linéaire mais dérivable, de manière à poursuivre la minimisation de l'erreur de couche en couche.

Le calcul se déroule en 2 temps :

- on effectue la mise à jour du réseau en avant
- on rétropropage l'erreur à partir de la sortie jusqu'à l'entrée

Soit y_i^k la sortie d'un neurone i de la couche k .

$$y_i^k = \sigma \left[\sum_{j=1}^p w_{ij} y_j^{k-1} - \theta \right]$$

On veut minimiser l'erreur quadratique sur la couche de sortie, par rapport à une sortie désirée :

$$E = \frac{1}{2} \sum_{i=1}^n (y_i - y_{d_i})^2$$

7.1 Une couche

Pour un réseau à une couche, la différentiation de l'erreur donne :

$$\frac{\partial E}{\partial w_{ik}} = \sum_{j=1}^n (y_j - y_{d_j}) \frac{\partial (y_j - y_{d_j})}{\partial w_{ik}}$$

La seule valeur non-nulle est la composante correspondant à $j = i$, car seule la sortie y_i est fonction du poids w_{ik} . De plus, la sortie désirée ne dépend pas du poids. On a alors :

$$\frac{\partial E}{\partial w_{ik}} = (y_i - y_{d_i}) \frac{\partial y_i}{\partial w_{ik}} = (y_i - y_{d_i}) x_k \sigma'(p_i - \theta)$$

En posant $\delta_i = (y_i - y_{d_i}) \sigma'(p_i - \theta)$ on a alors le gradient de l'erreur quadratique :

$$\frac{\partial E}{\partial w_{ik}} = \delta_i \cdot x_k$$

Pour minimiser l'erreur quadratique, il faut donc utiliser la règle d'apprentissage :

$$w_i(t+1) = w_i(t) + \varepsilon (y_i^d(t) - y_i(t)) x_i(t)$$

qui correspond bien à la règle d'adaptation de l'Adaline.

7.2 2 couches

De la même manière, on cherche à minimiser l'erreur quadratique. Calculons le gradient de l'erreur par rapport au poids w_{kh}^2 connectant le neurone h de la couche 1 au neurone k de la couche 2 :

$$\frac{\partial E}{\partial w_{kh}^2} = \sum_{i \in \text{couche 3}} \frac{\partial E}{\partial p_i^3} \frac{\partial p_i^3}{\partial w_{kh}^2}$$

Calculons d'abord :

$$\frac{\partial E}{\partial p_i^3} = \frac{\partial E}{\partial y_i^3} \frac{\partial y_i^3}{\partial p_i^3}$$

de : $y_i^3 = \sigma \left[\sum_{j \in \text{couche 2}} w_{ij}^3 \cdot y_j^2 \right]$ et comme $E = \frac{1}{2} \sum_{i=1}^n (y_i^3 - y_{d_i})^2$

on tire :

$$\frac{\partial E}{\partial p_i^3} = (y_i^3 - y_{d_i}) \sigma' [p_i^3]$$

Par ailleurs :

$$\frac{\partial p_i^3}{\partial w_{kh}^2} = \frac{\partial}{\partial w_{kh}^2} \left[\sum_{j \in \text{couche 2}} w_{ij}^3 \cdot y_j^2 \right]$$

Comme y_k^2 est le seul terme qui dépend de w_{kh} on a :

$$\frac{\partial p_i^3}{\partial w_{kh}^2} = w_{ik}^3 \frac{\partial}{\partial w_{kh}^2} [y_k^2]$$

en remplaçant y_k^2 par sa valeur, on a :

$$\frac{\partial p_i^3}{\partial w_{kh}^2} = w_{ik}^3 \frac{\partial}{\partial w_{kh}^2} \left[\sigma \left(\sum_{m \in \text{couche 1}} w_{km}^2 \cdot y_m^1 \right) \right]$$

Un seul terme est non nul ($m = k$) :

$$\frac{\partial p_i^3}{\partial w_{kh}^2} = w_{ik}^3 \cdot \sigma'(p_k^2) \cdot y_k^1$$

Revenons à l'équation initiale :

$$\frac{\partial E}{\partial w_{kh}^2} = \sum_{i \in \text{couche 3}} (y_i^3 - y_{d_i}) \cdot \sigma'(p_i^3) \cdot w_{ik}^3 \cdot \sigma'(p_k^2) \cdot y_k^1$$

En posant $\delta_i^3 = (y_i^3 - y_{d_i}) \cdot \sigma'(p_i^3)$, on a :

$$\frac{\partial E}{\partial w_{kh}^2} = \left[\sum_{i \in \text{couche 3}} \delta_i^3 \cdot w_{ik}^3 \right] \cdot \sigma'(p_k^2) \cdot y_k^1$$

Cette expression est formellement identique au gradient de E par rapport à un poids de la dernière couche si on pose :

$$\delta_k^2 = \left[\sum_{i \in \text{couche } 3} \delta_i^3 \cdot w_{ik}^3 \right] \cdot \sigma'(p_k^2)$$

Par conséquent, l'erreur affectée à un neurone k de la couche 2 est égale à la somme des erreurs de la couche suivante pondérée par les poids reliant le neurone k à ces neurones de la couche 3.

On propose finalement la règle de modification suivante :

$$\Delta w_{kh}^j = -\alpha \cdot \delta_k^j \cdot y_h^{j-1}$$

Avec :

$$\delta_k^j = \begin{cases} y_k^j - y_{d_k} & \text{pour la dernière couche} \\ \left[\sum_{i \in \text{couche } j+1} \delta_i^{j+1} \cdot w_{ik}^{j+1} \right] \cdot \sigma'(p_k^j) & \text{pour les autres couches} \end{cases}$$

7.2.1 Remarques sur l'utilisation de la rétropropagation

Lissage

L'algorithme de rétropropagation brut correspond à un algorithme de type gradient stochastique fondé sur la minimisation d'une erreur instantanée. En pratique, on ajoute un terme de filtrage sur les incréments d'adaptation qui correspond à une pondération des erreurs quadratiques :

$$E(n) = \frac{1}{2} \sum_{i=1}^N \gamma^{n-i} \|y - y_d\|^2$$

Avec $\gamma < 1$.

On a alors la règle d'apprentissage :

$$\Delta w_{jk}(n) = -a \delta_j(n) x_k(n) + \gamma \Delta w_{jk}(n)$$

Ce qui correspond à la pondération exponentielle des erreurs. En pratique γ est inférieur et voisin de 1. On peut choisir un oubli variable faible au départ et tendant vers 1.

Valeurs initiales

Les valeurs initiales des poids doivent être différentes de 0 sinon le réseau ne peut pas apprendre ! En pratique, on choisit une valeur aléatoire comprise dans la dynamique des poids.

Test d'arrêt

Le plus simple est de fixer un nombre d'itérations.

En pratique on possède une base d'apprentissage et une base de test. On alterne apprentissage sur la BA et mesure de performances sur la BT jusqu'à atteindre des résultats convenables.

La rétropropagation comme approximateur universel

Des résultats théoriques ont montré que les RN à 3 couches (2 couches cachées) peuvent approximer des fonctions continues de R^p (ou des compacts) dans $[0,1]$. Cependant, on ne sait pas comment construire le réseau ni combien de neurones sont nécessaires dans la couche cachée. Celui-ci peut être infini !

Chapitre 8

Réseaux de neurones non-supervisés

1 Winner Take All (WTA)

Le WTA simule les mécanismes de compétition existant entre neurones ou populations de neurones. Le modèle courant utilise des groupes de neurones formels dont l'apprentissage est fixé par la règle de Hebb. L'ajout de liaisons inhibitrices latérales permet de simuler le processus de compétition. Après convergence, seul le neurone ayant la plus grande activité reste actif et inhibe tous les autres [Rumelhart and Zipser, 1985, Lippman, 1987]. Ce type de WTA est utilisé comme moyen de catégoriser les vecteurs présentés en entrée du réseau. En effet, si on présente ceux-ci un nombre suffisant de fois, chaque neurone du WTA acquiert une sensibilité différente (il faut cependant que les vecteurs soient orthogonaux entre eux sinon le même neurone peut gagner pour toute une famille de formes) [Grossberg, 1976, Grossberg, 1988, Kohonen, 1984].

Le modèle "Instar" de Grossberg utilise un mécanisme de compétition sur deux couches [Grossberg, 1973]. Ce n'est pas un WTA pur, mais plutôt un mécanisme de rehaussement de contraste. Nous en présentons cependant les caractéristiques afin de mieux appréhender le fonctionnement des mécanismes de compétition.

Dans ce modèle, chacun des N neurones d'une première couche envoie des liaisons inhibitrices sur l'ensemble des N neurones de la couche suivante excepté le neurone correspondant à la position du neurone de la couche d'entrée qui reçoit un lien activateur (voir figure 1).

x_i est la valeur instantanée du potentiel d'un neurone i du groupe de sortie. I_k est l'activité du neurone k de la couche d'entrée.

La dynamique de chacun des neurones de la couche de sortie est modélisée par l'équation :

$$\frac{dx_i}{dt} = -Ax_i + (B - x_i) \cdot I_i - x_i \sum_{k \neq i} I_k$$

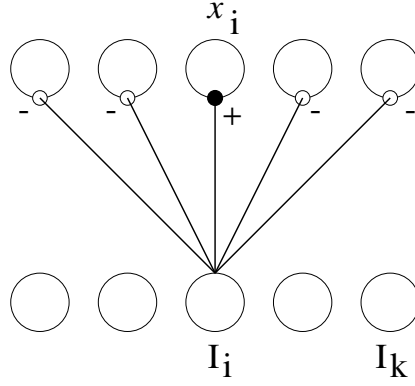


Figure 1. Compétition à deux couches : modèle INSTAR

On pose A et $B > 0$, $I = \sum_{i=1}^N I_i$ et $\theta_i = \frac{I_i}{I}$. Le vecteur $\theta = (\theta_1, \dots, \theta_N)^T$ est appelé *reflectance pattern*.

Le terme $-Ax_i$ est un terme de décroissance passif du potentiel du neurone. $(B - x_i) \cdot I_i$ représente l'influence du lien activateur connectant le neurone de la couche d'entrée et celui de la couche de sortie. Enfin, le terme $-x_i \sum_{k \neq i} I_k$ représente l'influence des inhibitions provenant des autres neurones. Il est à noter que le potentiel x_i est borné entre 0 et B .

Après un régime transitoire, on obtient la valeur à l'équilibre lorsque $\frac{dx_i}{dt} = 0$. La valeur moyenne X_i du potentiel des neurones de la couche de sortie est alors :

$$X_i = \frac{I_i \cdot B}{A + I} = \theta_i \frac{B \cdot I}{A + I}$$

On déduit de cette équation que l'activité globale de sortie est normalisée puisque $\sum X_i = \sum \left(\frac{I_i \cdot B}{A + I} \right) = \frac{B \cdot I}{A + I} < B$.

Un mécanisme de compétition à une couche a aussi été proposé par Grossberg (voir figure 2). L'équation différentielle régissant le comportement du réseau s'écrit alors :

$$\frac{dx_i}{dt} = -Ax_i + (B - x_i) \cdot [f(x_i) + I_i] - x_i \sum_{k \neq i} [f(x_k) + I_k]$$

Avec $f(\cdot)$ la fonction de sortie du neurone.

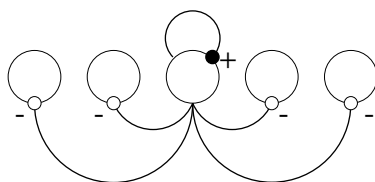


Figure 2. *Compétition sur une couche*

2 Carte de Kohonen

Il a été observé que, dans de nombreuses zones du cortex, des colonnes voisines ont tendance à réagir à des entrées similaires. Dans les aires visuelles, par exemple, deux colonnes proches sont en correspondance avec deux cellules proches de la rétine [Hubel and Wiesel, 1977]. Des observations identiques on pu être faites dans le bulbe olfactif, ou l'appareil auditif [Knudsen and Konishi, 1979] (voir plus généralement [Woolsey, 1981a, Woolsey, 1981b, Woolsey, 1981c]). Ces observations ont mené Kohonen [Kohonen, 1989] à proposer un modèle de *carte topologique auto-adaptative* qui permet de coder des motifs présentés en entrée tout en conservant la topologie de l'espace d'entrée.

Dans la plupart des applications, les neurones d'une carte de kohonen sont disposés sur une grille 2D (cf. figure 3-b). Le poids associé aux liaisons latérales entre neurones est fonction de la distance entre le neurone source et le neurone cible. La valeur du poids est donnée par une fonction "chapeau mexicain" (Difference Of Gaussians - DOG - voir figure 3-b)). Les connexions provenant des entrées, quant à elles, arrivent perpendiculairement au plan formé par les liaisons latérales.

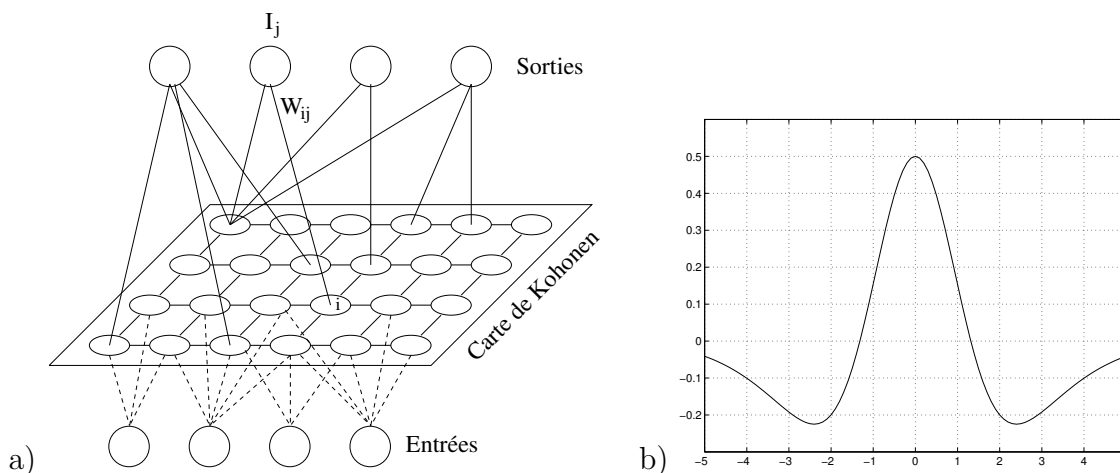


Figure 3. a) *Schématisation d'une carte de Kohonen.* b) *Représentation d'une différence de 2 gaussiennes.*

On distingue deux types de fonctionnement. Dans un premier temps, l'ensemble des formes devant être apprises sont présentées au réseau et les vecteurs de poids sont mis à jour de manière à approximer les vecteurs d'entrée. Les paramètres de la carte sont

adaptés au fur et à mesure pour qu'elle se stabilise de plus en plus. La deuxième étape est la phase d'utilisation proprement dite. Dans ce cas, on présente un motif particulier et c'est le neurone dont le vecteur de poids minimise la distance avec le vecteur d'entrée qui réagit.

Le mécanisme d'apprentissage est le suivant. La carte comporte N neurones organisés sur une grille rectangulaire. Chaque neurone est relié à M entrées. Le vecteur des poids est donné par $\mathbf{W}_i = [W_{i1}, \dots, W_{iM}]^T$. Le vecteur d'entrée est donné par $\mathbf{I} = [I_1, \dots, I_M]$.

- Les poids des liaisons latérales sont initialisés aléatoirement.
- On calcule la distance entre le vecteur présenté et le vecteur de poids de chaque neurone :

$$d_i = \sum_{j=1}^M (I_j - W_{ij})^2$$

- On choisit le neurone dont la distance avec le vecteur d'entrée est la plus petite.
- Les poids de ce neurone sont mis à jour selon l'équation :

$$W_{kj}(t+1) = W_{kj}(t) + \eta(t) \cdot (I_j(t) - W_{kj}(t))$$

- Le coefficient d'apprentissage $\eta(t)$ est choisi dans $[0, 1]$. Il doit décroître peu à peu.
- Le voisinage des neurones est mis à jour de manière à se rétrécir au cours du temps.
- On recommence sur l'ensemble des motifs jusqu'à la stabilisation complète des poids.

Après un long temps de convergence, le réseau évolue de manière à représenter au mieux la topologie de l'espace de départ. Il faut en fait noter que la notion de conservation de la topologie est en fait abusive puisqu'en général la taille du vecteur d'entrée est bien supérieure à la dimension de la carte (souvent 2) et il est donc impossible de conserver parfaitement la topologie. En fait le terme de quantification vectorielle de l'espace d'entrée doit plutôt être utilisé.

L'implémentation n'est pas biologiquement plausible : le temps de convergence est trop long pour expliquer notre capacité à apprendre une forme en un seul essai. Le point le plus important est que pour permettre la réduction du voisinage, la valeur des liens latéraux doit changer. Ainsi un lien activateur peut devenir inhibiteur ! Un autre point est que la qualité de la convergence du réseau dépend grandement des paramètres spécifiant la vitesse d'apprentissage et la taille du voisinage. Ces paramètres sont fixés de manière subjective par le programmeur. Dans [Kohonen, 1993], Kohonen souligne que la modification de ces paramètres pourrait provenir des transformations ontogénétiques du cerveau.

Il faut aussi noter que les cartes de Kohonen ne peuvent pas être utilisées "en ligne" puisque pour assurer la convergence : il faut pouvoir présenter plusieurs centaines de fois

l'ensemble des motifs. Or, dans le contexte d'un système autonome, on ne peut connaître à l'avance l'ensemble des formes qui seront rencontrées. Par ailleurs, pour que la carte apprenne correctement, il est nécessaire de présenter les formes de façon aléatoire (ce qui est difficile à imaginer dans un contexte réel). On ne pourra donc pas utiliser les cartes de Kohonen pour catégoriser des formes, mais elles peuvent servir de justification à nos pré-traitements au niveau des images (apprentissage de la forme des filtres de bas niveau, détecteurs de contours...).

3 Adaptive Resonance Theory (ART)

Le principe général du modèle ART conçu par Carpenter et Grossberg est de créer un système d'apprentissage autonome capable de concilier adaptation et stabilité (compromis à effectuer entre plasticité et rigidité synaptique). Le problème d'un système trop plastique est qu'il peut apprendre des informations même si elles ne sont pas pertinentes ou oublier des informations apprises. A l'inverse, un système trop rigide empêche tout apprentissage. La résolution de ce dilemme entre plasticité et stabilité est donc centrale à toute recherche visant à comprendre ou créer un système capable d'apprendre de manière autonome. Une description complète des équations régissant le modèle ART peut être trouvée dans [Grossberg and Mingolla, 1985, Carpenter and Grossberg, 1987, Grossberg, 1988, Grossberg and Somers, 1991]. Pour notre part, nous nous sommes surtout intéressés au facteur de vigilance introduit par Grossberg pour gérer de manière autonome le passage d'un mode "rigide" à un mode "plastique".

La modèle proposé par Grossberg s'appuie sur une analyse descendante du problème de reconnaissance, du niveau le plus abstrait (classification) vers la couche d'entrée des vecteurs binaires à reconnaître (voir figure 4). Le premier niveau **F1** est constitué d'une mémoire à court terme (Short Term Memory - STM) qui permet de normaliser l'activité des formes d'entrée par rapport à l'activité totale des entrées. Le deuxième niveau, **F2** est un WTA qui effectue la catégorisation des formes. Les niveaux **F1** et **F2** sont reliés l'un à l'autre dans les deux sens et c'est la modification des poids associés à ces liens qui permet l'apprentissage à long terme.

La reconnaissance et l'apprentissage d'une forme reposent sur un fonctionnement inspiré de la résonance dans les systèmes physiques. Si le système a déjà appris une catégorie, F1 et F2 entrent en résonance rapidement et il y a renforcement de la forme apprise grâce à un système d'apprentissage associatif des liens entre **F1** et **F2**.

Pour ce qui est de la décision d'apprendre une nouvelle forme, un paramètre appelé *vigilance* est introduit. Il permet de décider si oui ou non, la forme présentée est suffisamment nouvelle, vis à vis d'un critère de similarité, pour être apprise. Le calcul de similarité est effectué en comparant l'erreur de reconstruction de la forme mémorisée avec le motif actuel. Ce paramètre est extrêmement intéressant puisqu'il permet de régler le niveau de généralisation ou de discrimination (voir figure 5).

Le modèle ART offre une solution originale à l'apprentissage autonome et non-supervisé

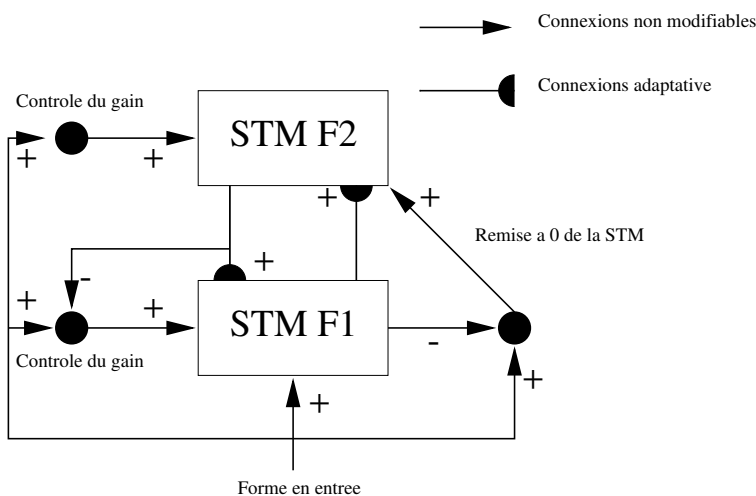


Figure 4. Schéma complet du modèle ART faisant apparaître les deux niveaux d'analyse, les liens et les contrôles de gain.

de catégories. Ses qualités sont principalement dues à l'introduction d'un paramètre de vigilance qui permet de définir le "degré" d'apprentissage et de gérer correctement le dilemme plasticité-stabilité. Cependant, la vigilance étant un paramètre essentiel au bon fonctionnement de l'apprentissage, il est nécessaire que ce paramètre soit correctement maîtrisé.

Un développement de ART, appelé ART2 a été effectué afin de permettre de catégoriser des formes analogiques plutôt que des motifs binaires [Carpenter et al., 1991a]. Le ARTMAP ou fuzzy-ARTMAP, quant à lui, est l'association de deux modules ART ou fuzzy-ART reliés entre eux par l'intermédiaire d'un *map field*. Ces cartes permettent de prédire les associations pouvant exister entre les catégories fabriquées par les deux modules ART [Carpenter et al., 1991b, Carpenter et al., 1992]. Dans ce cas, on peut apprendre la topologie existant entre les formes présentées. Cependant, cette topologie ne peut être apprise qu'a posteriori (une fois que les catégories ont été créés dans les deux modules ART). Les informations de topologie ne peuvent donc pas être utilisées comme moyen de généralisation a priori.

La seule lacune du modèle ART est son incapacité à conserver la topologie, contrairement au réseau de Kohonen. L'utilisation d'un WTA au niveau de F2 ne permet pas d'utiliser des informations de "ressemblance" entre les formes apprises.

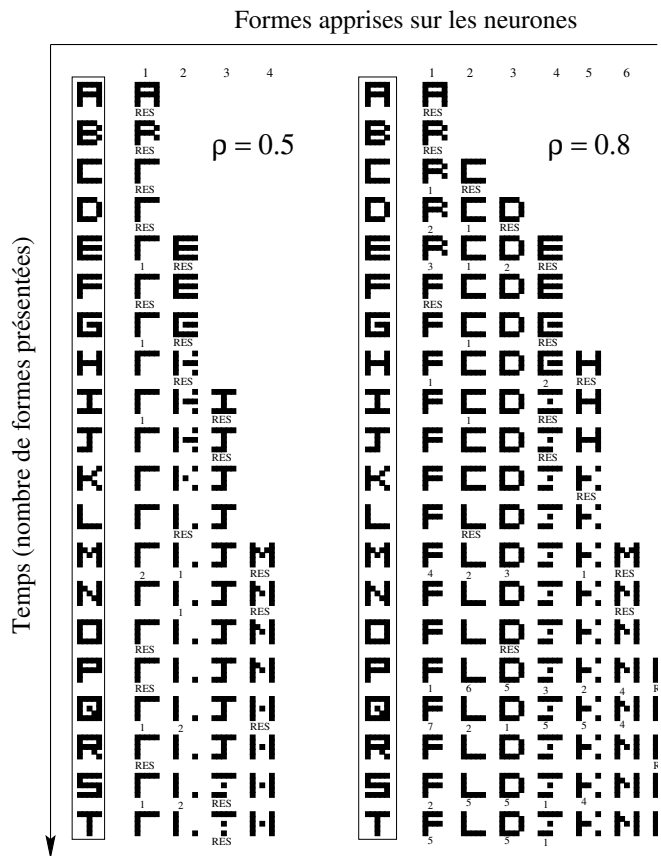


Figure 5. Exemple de formation de catégories pour deux valeurs du paramètre de vigilance.

Chapitre 9

Renforcement

Les techniques de renforcement constituent toutes les méthodes qui permettent à un système d'apprendre à résoudre une tâche en indiquant simplement le degré de réussite à un instant donné par le biais d'une récompense ou d'une punition (*apprentissage par essai et erreur*).

1 Retour sur la règle de Hebb et le LMS

La règle de Hebb, et mieux, celle du LMS, permettent de modéliser le conditionnement pavlovien.

2 Conditionnement instrumental

La règle de Sutton et Barto permet de modéliser ce type d'apprentissage.

Le principal modèle du conditionnement opérant, est celui proposé par Sutton et Barto [Barto et al., 1981]. La fonction d'activation des neurones utilisés dans leur modèle est la suivante :

$$O_j = H\left(\sum_i W_{ij} \cdot I_i + \text{bruit}\right) \quad (1)$$

Avec $H(x)$ la fonction de Heaviside définie par :

$$H(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{sinon} \end{cases}$$

Le bruit ajouté en sortie des neurones est introduit pour permettre au réseau de choisir une réaction alors qu'aucune association sensori-motrice n'existe au préalable.

Le principe de la règle d'apprentissage est de ne renforcer le poids synaptique entre un stimulus et un neurone donné que si l'on a détecté que le déclenchement de sa réponse axonale entraînait une récompense. On peut exprimer cette idée sous la forme de l'équation suivante :

$$W_{ij}[t + 1] = W_{ij}[t] + \epsilon \cdot [R[t] - R[t - 1]] \cdot [O_j[t - 1] - O_j[t - 2]] \cdot I_i[t - 1] \quad (2)$$

Où $R[t]$ est la récompense donnée à l'instant t .

On peut aussi écrire cette équation sous une forme différentielle :

$$\frac{\partial W_{ij}(t + 1)}{\partial t} = \epsilon \cdot \frac{\partial R(t)}{\partial t} \cdot \frac{\partial O_j(t - 1)}{\partial t} \cdot I_i(t - 1) \quad (3)$$

Cette équation modélise correctement les mécanismes du conditionnement instrumental quand il n'y a pas de retard entre l'apparition du stimulus et la récompense. On considère que le **SC** est présent pendant toute la durée de l'apprentissage. Si la sortie du neurone est activée à l'instant t et qu'une récompense est accordée à l'instant $t + 1$, il y a renforcement du poids. Si au contraire, une punition est donnée, le poids est diminué. Le tableau 1 récapitule l'ensemble des cas envisageables et l'apprentissage réalisé.

Variation du renforcement	Variation de la sortie	Entrée	dW
↗	↗	Active	↗
↘	↗	Active	↘
0	↘ ou ↗	Active	0
↘ ou ↗	0	Active	0
↘ ou ↗	↘ ou ↗	Inactive	0

Tableau 1. Récapitulation du fonctionnement du modèle de Sutton et Barto. dW : variation du poids. Entrée : Active \equiv activité \neq 0; Inactive \equiv activité = 0

Cependant, il faut remarquer que ce mécanisme n'est possible que s'il y a concomitance de la réponse et du signal de renforcement. Ce modèle ne rend en effet pas compte des cas où l'on aurait un temps d'attente entre l'apparition du stimulus et le signal de renforcement.

2.1 Formalisme des méthodes de renforcement

Considérons un système inter-agissant avec son milieu. On représente l'ensemble des situations que l'agent peut rencontrer par un ensemble d'états accessibles S , liés entre eux par des relations de proximité. Dans chaque état, le système est susceptible de

réaliser un certain nombre d'actions appartenant à l'ensemble A . Après la réalisation d'une certaine action depuis l'état $s \in S$, le système se retrouve dans l'état $s' \in S$ (voir figure 1). Le problème est de savoir quel action effectuer dans tel ou tel état. Il s'agit en fait de mettre en correspondance un état donné et une action à effectuer de manière à résoudre au mieux une tâche donnée.

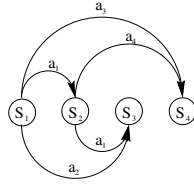


Figure 1. Représentations de différents états et des transitions possibles entre eux-ci.

Le signal de renforcement r est calculé lors du passage d'un état à un autre en fonction des contraintes de viabilité du robot. Si par exemple, après qu'un mouvement a été effectué, le robot se cogne dans un mur, il faudra lui donner un signal de renforcement négatif. Si, par contre, le robot effectue la tâche attendue, il faudra le récompenser. Dans les autres cas le signal peut avoir une valeur nulle. Dans la plupart des cas, les signaux de renforcement sont pris tels que $r \in [-1, 1]$.

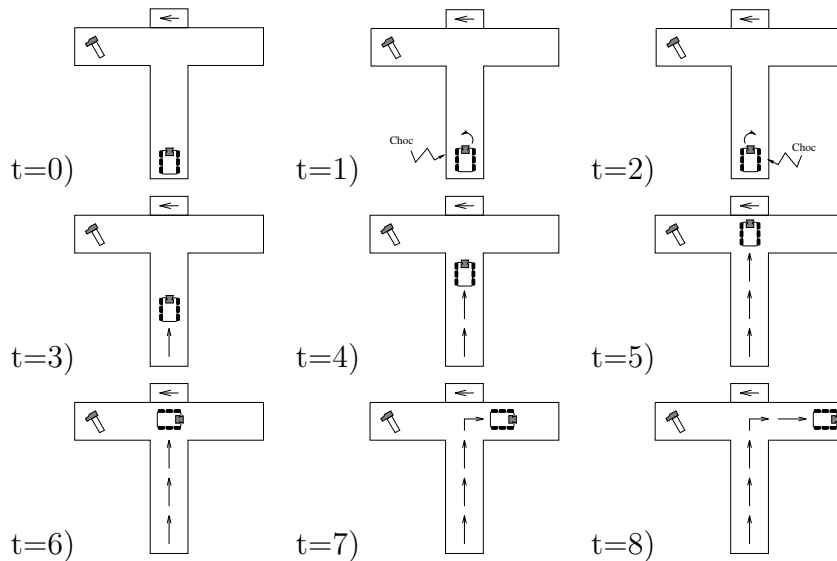


Figure 2. Exemple de problème de labyrinthe. Le but du robot est d'aller chercher le marteau situé dans la branche gauche du labyrinthe.

Prenons l'exemple du labyrinthe de la figure 2. On peut considérer que l'ensemble des états accessibles par l'agent est l'ensemble des cases du labyrinthe. Les actions peuvent être les mouvements haut, bas, gauche et droite. Pour le renforcement, on peut choisir $r = 1$ lorsque le robot atteint le but (le robot a atteint le marteau), $r = -1$ lorsque le robot se cogne dans un mur (au temps $t=1$ ou $t=2$ par exemple) et $r = 0$ autrement

(*goal-reward representation*); ou bien on peut être plus sévère et punir le robot pour toute autre action que celle menant au but (*action-penalty representation*). Le choix du type de renforcement n'est pas aussi anodin qu'il peut le paraître. En effet, il a été montré que la solution action-penalty offrait une représentation plus dense et facilitait donc la convergence des algorithmes de renforcement [Barto et al., 1991].

On appelle *politique* $\pi(s)$ du système, l'application qui, à tout état $s \in S$, fait correspondre une action a . Le but des algorithmes de renforcement est de trouver la politique optimale, c'est à dire, l'ensemble des associations état-action devant être effectuées pour maximiser un critère d'optimalité $V^*(s)$ représentant la valeur maximale du cumul des renforcements pouvant être obtenues à partir de cet état. Il existe plusieurs critères d'optimalité basés sur la maximisation d'une fonctionnelle des récompenses obtenues au cours du parcours.

Le critère d'optimalité à *horizon fini*, ne permet d'assurer l'optimalité que sur la fenêtre temporelle explorée. Si N est la largeur de cette fenêtre, le critère s'écrit :

$$C_{HF}(N) = E \left(\sum_{k=0}^N r_k \right)$$

où r_k est la récompense à l'instant k , et $E(\cdot)$ est l'espérance mathématique. Le défaut de ce critère est qu'on n'assure l'optimalité que si le temps pour parcourir le chemin le plus court est inférieur à N . Ce paramètre agit donc comme une connaissance a priori sur le temps pour parcourir le chemin optimal. Dans un contexte entièrement autonome, ce critère peut ne pas sembler adéquat.

Le critère de *récompense moyenne* s'applique à un horizon infini. Il s'écrit :

$$C_{RM} = \lim_{N \rightarrow +\infty} E \left(\frac{1}{N} \sum_{k=0}^N r_k \right)$$

Le problème principal de ce critère est qu'il ne fait pas de différence entre deux politiques ayant même moyenne mais dont l'une fait croître le gain brutalement et l'autre qui garde un gain quasi-constant.

Le dernier modèle est le critère *infinite-horizon discounted*. Il essaie de tenir en compte le problème du modèle précédent en pondérant de manière géométrique les récompenses obtenues au cours du temps selon un facteur $\gamma \in [0, 1]$. Analytiquement, on écrit :

$$C_{IHD}(\gamma) = E \left(\sum_{k=0}^{+\infty} \gamma^k \cdot r_k \right)$$

Les techniques de renforcement nécessitent deux comportements qui peuvent sembler contradictoires. Tout d'abord, un mécanisme doit pouvoir permettre d'explorer l'ensemble des états afin d'assurer que la solution est réellement optimale (exploration). Cependant, la politique doit aussi converger vers une situation optimale et stable (exploitation). Plusieurs méthodes permettent cette ambivalence de comportement.

La stratégie de la gourmandise consiste à prendre toujours l'action qui apporte la meilleure récompense. Le problème est qu'une action sous-optimale peut, dans un premier temps, apporter une meilleure récompense que l'action optimale. Et l'algorithme peut ainsi se bloquer sur cette solution sous-optimale.

Cette stratégie peut être rapprochée de l'algorithme A^* employé en IA classique pour l'exploration de graphes. Cependant, dans ce cas, cette exploration est guidée par l'utilisation d'une fonction heuristique qui constitue une représentation des connaissances a priori sur les propriétés du graphe et le but à atteindre.

Les stratégies de *recherche aléatoire* prennent différentes formes : une solution consiste à effectuer l'action apportant la meilleure récompense avec une probabilité p ou une action prise au hasard avec une probabilité $1 - p$. Pour faire converger l'algorithme vers une situation stable, on fait décroître p peu à peu.

Une alternative consiste à décider l'action effectuée de manière aléatoire selon une distribution de Boltzmann dépendant de la valeur de Récompense Attendue pour chaque action a ($RA(a)$).

$$Prob(a) = \frac{e^{\frac{RA(a)}{T}}}{\sum_{a' \in A} e^{\frac{RA(a')}{T}}}$$

2.2 Les techniques de renforcement

Dans ce paragraphe, nous passons en revue l'ensemble des méthodes courantes utilisées pour le renforcement. Chacune de ces méthodes tente d'approximer la valeur optimale $V^*(s)$ d'un état par sa valeur estimée $V(s)$. Les algorithmes de renforcement doivent permettre de faire converger $V(s)$ vers $V^*(s)$.

2.2.1 Schéma de fonctionnement général

Les algorithmes de renforcement fonctionnent sur le même schéma de base. Ils comportent trois étapes principales :

1. A partir de l'état s , le choix d'un mouvement a est fait en fonction de la stratégie utilisée (gourmandise, recherche aléatoire)
2. Le système se retrouve dans l'état s' qui se trouve être un état récompensé ($r = 1$), puni ($r = -1$) ou neutre ($r = 0$).
3. En fonction du résultat obtenu on met à jour la valeur estimée $V(s)$ de l'état et on retourne en 1.

Les différences entre les algorithmes de renforcement résident dans la vitesse de convergence et la complexité de l'algorithme.

Le schéma 3 montre un exemple de propagation de la récompense due à une méthode de renforcement pour trois états, les uns à la suite des autres, dans une séquence d'actions.

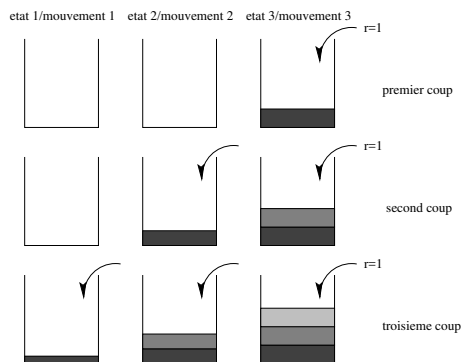


Figure 3. Exemple de propagation du renforcement de proche en proche.

2.2.2 L'algorithme TD(λ)

A la suite de ses travaux de modélisation du conditionnement instrumental, Sutton a proposé une technique de renforcement appelée $TD(\lambda)$ ¹ [Sutton, 1988]. Le principe est d'utiliser les prédictions successives effectuées au cours du temps, et non l'erreur entre la prédiction actuelle et la récompense réellement reçue, pour régler le problème du *credit assignment*. Si s est un état quelconque, s' un état déjà atteint à partir de cet état, l'algorithme de mise à jour est :

$$V(s) \leftarrow V(s) + \alpha \cdot (r + \gamma \cdot V(s') - V(s)) \cdot e(s) \quad (4)$$

où $e(s)$ est l'éligibilité de l'état s , calculée selon la formule :

$$e(s) = \sum_{k=1}^T (\lambda\gamma)^{T-k} \cdot \delta_{s_k}(s)$$

avec $\delta_{s_k}(s)$ la fonction indicatrice de dirac discrète telle que

$$\delta_{s_k}(s) = \begin{cases} 1 & \text{si } s = s_k \\ \text{sinon} & \end{cases} \quad (5)$$

$(\alpha, \lambda, \gamma) \in [0, 1] \times [0, 1] \times [0, 1]$. Il faut bien remarquer que ce n'est pas seulement l'état courant qui est mis à jour, mais aussi les états possédant une certaine rémanence.

Il est intéressant de regarder le comportement de cet algorithme pour les valeurs extrêmes de λ .

¹TD pour Temporal Difference

Dans le cas du $TD(0)$, seul l'état précédent est utilisé et il n'y a pas de rémanence. L'équation devient :

$$V(s) \leftarrow V(s) + \alpha \cdot (r + \gamma \cdot V(s') - V(s)) = (1 - \alpha) \cdot V(s) + \alpha \cdot (r + \gamma \cdot V(s'))$$

Cela revient à mettre à jour la fonction d'estimation de la valeur de récompense uniquement en fonction de la fonction valeur de l'état qui est actuellement l'état suivant et de la récompense r réellement perçue. Dans le cas du $TD(1)$, cela revient à mettre à jour les états en fonction du nombre de fois qu'ils ont été visités.

Des résultats forts de convergence ont été montrés pour cet algorithme. Dès la publication de son algorithme, Sutton a montré la convergence en moyenne mais uniquement pour $TD(0)$ [Sutton, 1988]. Dayan a par la suite élargi ce résultat à tous λ [Dayan, 1992]. Il a aussi démontré la convergence avec un critère plus dur en démontrant que l'algorithme $TD(\lambda)$ convergeait avec une probabilité 1 [Dayan and Sejnowski, 1994].

Cet algorithme est donc intéressant du point de vue théorique. Cependant, il nécessite un temps de calcul important, surtout pour un λ quelconque et bien que la convergence soit plus rapide pour des valeurs de λ grandes.

2.2.3 Le Q-learning

Pour expliquer la technique du Q-learning de Watkins [Watkins, 1989], il faut introduire un nouveau paramètre $Q(s, a)$ qui, pour chaque état s donne la valeur estimée de la récompense totale pouvant être obtenue en effectuant a . L'algorithme consiste à actualiser les valeurs de $Q(s, a)$ chaque fois que l'action a a été choisie, alors que le système se trouvait dans l'état s . L'équation de mise à jour est :

$$Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma \cdot Q_{Max}(s'))$$

Avec $Q_{Max}(s') = \text{Max}_{a' \in A} Q(s', a')$

s' n'est autre que l'état atteint depuis s après avoir effectué l'action a .

Watkins a montré que les valeurs de $Q(s, a)$ tendaient vers les valeurs optimales $Q^*(s, a)$ avec une probabilité de 1. Ce résultat fort ainsi que sa facilité d'implémentation ont fait que cet algorithme est devenu l'un des plus utilisés pour le renforcement bien que sa vitesse de convergence soit modérée.

2.2.4 L'algorithme Dyna-Q

Jusqu'à présent, on a vu que les mises à jour se faisaient en fonction de l'expérience effectuée, à savoir, des récompenses obtenues par expérience pour avoir utilisé telle action dans tel état. Ces expériences sont difficiles à acquérir, et c'est parce qu'il faut pouvoir en effectuer beaucoup pour assurer la convergence que les méthodes de renforcement du type $TD(\lambda)$ ou Q-learning requièrent un temps d'exploitation parfois prohibitif. L'idée

de Sutton pour l'algorithme Dyna [?, Sutton, 1991] est de mieux utiliser cette information difficile à acquérir. Pour cela le système peut évoluer entre deux comportements; l'acquisition de nouvelles expériences (identique au Q-learning); l'utilisation des données déjà existantes pour essayer de propager l'information disponible dans certains états à d'autres états. Il est, de plus, nécessaire de mémoriser la récompense estimée $R(s, a)$ pour effectuer l'action a dans l'état s .

- Passage de l'état s à l'état s' après avoir effectué l'action a et reçu le renforcement r
- Mise à jour des valeurs $Q(s, a)$ selon la technique de Watkins
- Choix de k valeurs de $Q(s, a)$ au hasard dans l'ensemble des paires état-action et mise à jour comme pour Watkins en employant $R(s, a)$ au lieu de r .
- Itération du processus jusqu'à convergence

Le Dyna-Q fait croître drastiquement la vitesse de convergence comptée en nombre d'essais. Par contre, elle demande évidemment k fois plus de temps de calcul pour chaque expérience.

Chapitre 10

Multi-agents

1 Introduction

L'approche multi-agent provient, en partie, d'une critique de l'intelligence artificielle classique qui considère qu'un problème doit être traité par un superviseur qui décompose la tâche à réaliser en tâches plus simples, puis en itérant le procédé jusqu'à atteindre un niveau de description atomique, soluble par le superviseur. A l'inverse, dans les systèmes multi-agents, la résolution de problèmes est le fruit d'interaction entre entités relativement autonomes appelées *agents*. Ces interactions sont plus ou moins complexes (coopération, conflits, compétitions...) et peuvent faire *émerger* des comportements qui n'étaient pas spécifiquement comprise dans la description exhaustive des agents.

1.1 Pourquoi distribuer l'intelligence ?

- Les problèmes sont *physiquement* distribués : les problèmes complexes le sont souvent (réseaux, chaînes de montage, gestion de trafic...)
- Les problèmes sont *fonctionnellement* distribués
- Les réseaux imposent une vision distribuée
- La complexité impose une vision locale
- Les systèmes doivent pouvoir s'adapter à des modifications de structures ou d'environnement
- Le génie logiciel va dans le sens d'une conception en termes d'unités autonomes en interactions : langages objets

1.2 Définition d'un agent (physique ou virtuel)

C'est une entité "computationnelle" évoluant de manière autonome dans un environnement donné. Par autonomie, on entend capable d'agir sans intervention externe en fonction de sa perception et de sa mémoire.

Ex:

- robot autonome
- agent personnel (Wizard MS...)

On ne peut réellement parler de SMA que si les agents sont lancés en parallèle dans un même "environnement".

Dans ce cas, agissant sur des ressources communes, ils interagissent forcément à des degrés divers.

La dynamique résultante de leur interaction peut évoluer dans le temps en fonction de plusieurs critères :

- leur capacité à apprendre
- leur aptitude à communiquer (via un langage ou autre...)

Lorsque l'on a une tâche donnée à résoudre, avoir recourt à un SMA c'est réfléchir à la division des tâches entre les agents (IA distribuée) et à leur capacité d'interaction.

Ex:

- Bras manipulateur
- Problème des tours de Hanoi

Pour résumer, un agent est un système :

- a qui est capable d'agir dans un environnement
- b qui peut communiquer directement avec d'autres agents
- c qui est mue par un ensemble de tendances (sous forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser)
- d qui possède des ressources propres
- e qui est capable de percevoir (mais de manière limitée) son environnement
- f qui ne dispose que d'une représentation partielle de cet environnement (éventuellement aucune)

g qui possède des compétences et offre des services

h qui peut éventuellement se reproduire

i dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.

1.3 Définition d'un système multi-agents (SMA)

C'est un système composé de :

1. Un environnement E , disposant généralement d'une métrique
2. Un ensemble d'objets O . C'est objets sont situés, c.a.d. que pour chaque objet il est possible d'associer une position dans E . Ils sont passifs, c.a.d. qu'ils peuvent être perçus, créés, détruits et modifiés par les agents
3. Un ensemble A d'agents, qui sont des objets particuliers, lesquels représentent les entités actives du système
4. Un ensemble de relations R qui unissent des objets entre eux
5. Un ensemble d'opérations Op permettant aux agent de percevoir, produire, consommer, transformer et manipuler des objets
6. Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification (lois de l'univers)

1.4 Agent logiciel/agent physique

Agent logiciel :

- a se trouve dans un système informatique ouvert (ensemble d'applications, de réseaux et de systèmes hétérogènes)
- b peut communiquer avec d'autres agents
- c est mue par un ensemble d'objectifs propres
- d possède des ressources propres
- f ne dispose que d'une représentation partielle des autres agents
- g possède des compétences (services) qu'il peut offrir aux autres agents

i a un comportement tendant à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de ses représentations et des communications qu'elle reçoit.

Agent physique : Agent logiciel :

a se trouve situé dans un environnement

c est mue par une fonction de survie

d possède des ressources propres, sous la forme d'énergie et d'outils

e est capable de percevoir (mais de manière limitée) son environnement

f ne possède pratiquement aucune représentation de son environnement

g possède des compétences

h peut éventuellement se reproduire

i a un comportement tendant à satisfaire sa fonction de survie, en tenant compte des ressources, des perceptions et des compétences dont elle dispose

1.5 Niveaux d'organisation

On distingue 3 niveaux d'organisation entre agents :

- Le niveau micro-social : on s'intéresse aux interactions entre agents
- Le niveau des groupes : on s'intéresse aux structures intermédiaires (différenciation des rôles et activités, émergence de structures organisatrices...)
- Le niveau société globale : on s'intéresse à la dynamique d'un grand nombre d'agent, à sa structure générale et à son évolution.

2 Conception d'un SMA

2.1 Pourquoi utiliser des SMA ?

De plus en plus de systèmes sont des systèmes répartis :

- Bases de données (ressources)
- Internet (CPU)
- IHM multi-modales (information)

Il y a une augmentation sensible des ressources informatiques :

- explosion du WWW
- augmentation des puissance CPU
- développement du haut-débit

2.2 Quelle est la nécessité d'utiliser des SMA ?

- Pour intégrer la nouveauté, rendre le système adaptatif, modulable
- Pour faire coopérer des machines distantes
- Pour permettre l'inter-opérabilité entre systèmes hétérogènes

2.3 Inspiration

La POO : un objet encapsule les données et les méthodes pour pouvoir agir dessus.
MAIS, un agent peut en plus contrôler son comportement.

2.4 Quand utiliser un SMA ?

- Quand le problème est trop complexe mais peut être décomposé
- Quand il n'y a pas de solution générale ou lorsque celle-ci est trop coûteuse en CPU
- A des fins de modélisation (populations, structures moléculaires, réseaux de spins, tas de sables...)
- Quand on peut paralléliser le problème (gain de temps)
- Quand on veut une certaine robustesse (redondance)
- Quand l'expertise provient de différentes sources
- Quand les données, contrôles, ressources sont distribués
- Quand le système doit être adaptatif

2.5 Conception

Il faut se poser les questions suivantes :

- Que perçoit l'agent ?
- Quels sont ces moyens d'action ?
- Quelles sont ses capacités de décision ?

Par ailleurs, pour la tâche à résoudre on peut se poser les questions suivantes ? :

- Quelles sont les capacités d'interactions avec les autres agents ?
- Quelle est leur nature ? (coopération, compétition, neutre...)
- Est-il nécessaire de leur donner des capacités de communication ?
- De quel type ? (imitation, signaux, langage...)

Il faut repenser la programmation.

Ex: Tâche émergentes : pas besoin de coder la tâche. C'est la dynamique d'interaction qui fait émerger le comportement global. Par ailleurs, cette interaction existe mais n'est pas "codée". Exemple des fourmis de Deneubourg ou des robots ramasseurs.

Ex: Tâche de déminage : la tâche consiste à déminer un terrain.

Différentes approches :

- Supervision : il existe un superviseur qui planifie le déminage. On optimise alors la recherche mais tout se fait séquentiellement.
- Tâche émergente : on utilise une stratégie de type exploration aléatoire par un ensemble d'agents qui s'évitent. La recherche est parallélisée mais il n'y a pas de coordination et l'on peut repasser au même endroit.
- Stratégie mixte : on effectue un compromis entre supervision et parallélisme en élisant un "gouvernement" (3 agents : premier ministre, ministre de la défense, ministre de la communication) dont les tâches sont spécifiques (resp. récupérer les informations provenant des agents et fabriquer une représentation globale, donner les ordres stratégiques, gérer les communications entre agents). Une stratégie intéressante consiste à élire le gouvernement parmi les citoyens en fonction de leur mérite (changement de rôle).

2.6 Applications

Les télécommunications, le commerce électronique, la robotique, l'optimisation de réseaux (transport, routage...), gestion.

Par ailleurs, des applications spécifiques visent la simulation de sociétés (écologie, économie).

3 Agents mobiles

3.1 Intérêt

Pour définir l'intérêt d'un SMA dans lequel les agents sont mobiles, il faut étudier l'approche client/serveur actuelle : dans ce type d'architecture, il y a beaucoup d'utilisation de la bande passante pour faire communiquer le client et le serveur et pour le transfert de données.

Dans une approche "agents mobiles", il n'y a pas de communication entre le client et le serveur mais, une fois que l'agent s'est déplacé sur le "serveur", les communication se font en local.

3.2 Avantages

- En terme de performances : il y a moins d'utilisation de la bande passante (les communications sont beaucoup plus rapides puisqu'elles sont locales), ou du moins de manière plus ponctuelle. Il faut cependant faire un compromis entre BP ou QoS (Qualité de service), et puissance CPU nécessaire au niveau du "serveur" (utilisation du RTC).
- En terme de sécurité : les données ne circulent pas sur le web.
- Le système est plus robuste car réparti.
- Les charges sont réparties, parallélisées.
- Le "client" peut faire autre chose pendant ce temps (voire même être éteint) : application en sans-fil ou utilisation nomade.
- Tout ceci est avant tout intéressant si les interaction entre agents sont nombreuses. Or ce devrait être le cas si le serveur est conçu comme un SMA puisque l'agent mobile doit alors interroger les différents agents "serveurs".
- A partir des fonctions de base fournies par les "serveur", le client peut composer un agent complexe qui utilise un ensemble de ces capacités (ex: ftp récursif).

- Les agents étant dynamiques, ils peuvent évoluer au cours du temps : ils peuvent apprendre des choses et les ramener au “client”, leur stratégie tant du côté client que serveur peut évoluer, ils n’agissent pas de manière automatique mais en fonction de leur passé...

3.3 Inconvénients

- Interopérabilité et portabilité : les agents peuvent fonctionner sur différentes plateformes.

4 Nouvelle architecture ISO

Au niveau application, il faut prévoir de nouvelles couches “réseau” pour l’interaction entre agents.

Base : langage de programmation commun multi-plateforme (ex: Java) ou architecture de communication au niveau réseau. (AGLETS/IBM, Grasshopper, Voyager, Concor- dia...), (AgentTCL...) Compréhension entre les agents : langage commun (ex: ACL)

⇒ Utilisation de CORBA (IIOP)

Compréhension au niveau sémantique : ontologie commune (ex: FIPA-OS)

5 Application : Projet Netmarché

5.1 Présentation

Le projet Netmarché a pour but d’aider un utilisateur à acheter des produits sur internet grâce à un agent mobile l’aidant à effectuer les négociations.

La première étape consiste à trouver les endroits potentiellement intéressants en effectuant une recherche (cf. trsp.).

5.2 Implantation : FIPA-OS

FIPA : Foundation for Intelligent Physical Agents

ACL : Agent Communication Language

Un agent FIPA est une entité possédant son propre état, comportement, thread et contrôle. Par ailleurs, il a la possibilité d’interagir et de communiquer avec les autres agents.

Architecture :

- AMS: Agent Management System : il se charge de la supervision des opérations liées au cycle de vie des agents (création, suppression, modification...). Page blanche (agents en cours localement).
- DF: Directory Facilitator : Pages jaunes. Un agent s'enregistre pour mettre ses services à la disposition des autres.
- ACC: Agent Communication Channel : Gestion des communication entre les agents.
- IPMT: Internal Platform Message Transport. Service de communication entre agent sur 1 plate-forme.

5.3 Autre stratégie possible

Apprentissage d'une carte cognitive.

Problèmes : faut-il communiquer explicitement ? Peut-on imiter ? Comment ?

5.4 Développement

Interopérabilité globale : agents comprenant plusieurs ontologies.

6 Applications

6.1 Tours de Hanoï

Deux types d'agents : les disques et les tours.

Les disques sont satisfaits s'ils reposent sur un disque de taille immédiatement supérieure ou sur une tour libre.

Les comportements des disques sont les suivants :

- 1 Comportements de satisfaction : aller sur un disque de taille immédiatement supérieure
- 2 Comportement de fuite : aller sur une autre tour. Ne pas aller sur la tour où veut aller l'agresseur !

Avec trois tours, la solution est optimale bien que cela ne fut pas le but a priori ! (Vient du fait que le problème des tours de Hanoï est très contraint).

Si on augmente le nombre de tour, le comportement devient alors sous-optimal.

Annexes A

Filtrage de Wiener

Signaux

- (u_n) processus aléatoire observé
- $\mathbf{U}(n) = [u_n, u_{n-1}, \dots, u_{n-M+1}]$ vecteur d'observation
- (x_n) processus "désiré", à estimer par filtrage linéaire de (u_n)
- (y_n) estimation de x_n
- (e_n) erreur d'estimation $(x_n - y_n)$

Filtre

- Filtre $H(z) = w_0 + w_1 z^{-1} + \dots + w_{M-1} z^{M-1}$
- $\mathbf{W}^T = [w_0, w_1, \dots, w_{M-1}]$ vecteur d'observation

$$y_n = \mathbf{W}^T \cdot \mathbf{U}(n) = \sum_0^{M-1} w_k u_{n-k}$$

Statistiques

- $\gamma_u(p) = E(u_n u_{n-p})$ fonction d'auto-corrélation de u
- $\mathbf{R} = E(\mathbf{U} \cdot \mathbf{U}^T)$ matrice de corrélation de l'observation
- $\gamma_{xu}(p) = E(x_n u_{n-p})$ fonction d'intercorrélation de x et de u
- $\mathbf{p} = E(x_n \cdot \mathbf{U})$ vecteur de corrélation observation/signal désiré

Le filtre optimal de Wiener \mathbf{W}^* s'obtient en minimisant l'erreur quadratique moyenne d'estimation :

$$J(\mathbf{w}) = E[(x_n - y_n)^2] = E(e_n^2) \quad \text{avec} \quad e_n^2 = (x_n - \mathbf{w}^T \mathbf{U}(n))(x_n - \mathbf{U}(n)^T \mathbf{w})$$

d'où
$$J(\mathbf{w}) = \sigma_x^2 - 2\mathbf{p}^T \mathbf{w} + \mathbf{w}^T \mathbf{R} \mathbf{w}$$

On cherche alors à annuler le gradient : $\nabla J(\mathbf{w}) = 2(\mathbf{R} \mathbf{w} - \mathbf{p}) = 0$

On obtient le filtre optimal, la solution unique des équations normales :

$$\boxed{\mathbf{R} \mathbf{w}^* = \mathbf{p}}$$

On a de plus :

$$\boxed{J_{min} = J(\mathbf{w}^*) = \sigma_x^2 - \sigma_y^2 = \sigma_x^2 - \mathbf{p}^T \mathbf{w}^*}$$

Annexes B

Algorithmes d'optimisation

0.2 Le Gradient Déterministe (GD)

Dans le gradient déterministe, on cherche à minimiser $\mathbf{J}(\mathbf{w}) = \mathbf{E}(\mathbf{e}_n)$. Pour cela, à l'instant n , on se déplace selon la plus grande pente dans la direction $-\nabla\mathbf{J}(\mathbf{w}_n)$ avec un pas μ_n . D'où la forme récursive de l'équation de minimisation :

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu_n(\mathbf{p} - \mathbf{R}\mathbf{w}_n)$$

μ_n peut être décroissant ou constant $\mu_n = \mu$ (avec $0 < \mu < \frac{2}{\lambda_{max}(R)}$).

0.3 Le Gradient Stochastique (GS) et le LMS (Least Mean Square)

Essayons maintenant d'appliquer le GD au filtre de Wiener. D'après les définitions de \mathbf{p} et \mathbf{R} , on a :

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu_n E(\mathbf{e}_n \mathbf{U}(n))$$

Le principe du Gradient Stochastique est de remplacer $E(\mathbf{e}_n \mathbf{U}(n))$ par $\mathbf{e}_n \mathbf{U}(n)$ uniquement. D'où l'algorithme GS (ou LMS).

GS ou LMS

$\mathbf{U}(n)$: vecteur d'observations

$\mathbf{e}_n = \mathbf{x}_n - \mathbf{w}_n^T \cdot \mathbf{U}_n$: erreur d'estimation

$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu_n E(\mathbf{e}_n \mathbf{U}(n))$

Bibliographie

- [Barto et al., 1991] Barto, A., Bradtke, S., and Singh, S. (1991). Real-time learning and control using asynchronous dynamic programming. Technical Report 91-57, Department of Computer Science, University of Massachusetts, Amherst.
- [Barto et al., 1981] Barto, A., Sutton, R., and Brouwer, D. (1981). Associative search network : A reinforcement learning associative memory. *Biological cybernetics*, 40:201–211.
- [Carpenter and Grossberg, 1987] Carpenter, G. and Grossberg, S. (1987). Invariant pattern recognition and recall by an attentive self-organizing art architecture in a nonstationary world. *Proceeding of Neural Network*, 2:737–745.
- [Carpenter et al., 1991a] Carpenter, G., Grossberg, S., and Rosen, D. (1991a). Art2-a : an adaptive resonance algorithm for rapid category leaning and recognition. *Neural Networks*, 4:493–504.
- [Carpenter et al., 1992] Carpenter, G. A., Grossberg, S., Markuzon, N., Reynolds, J. H., and Rosen, D. B. (1992). Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Trans. Neural Networks*, 3(5):698–712.
- [Carpenter et al., 1991b] Carpenter, G. A., Grossberg, S., and Reynolds, J. (1991b). ARTMAP: A self-organizing neural network architecture for fast supervised learning and pattern recognition. In *Proc. Int. Joint Conf. on Neural Networks, Vol. 1*, pages 863–868, Seattle.
- [Dayan, 1992] Dayan, P. (1992). The convergence of $td(\lambda)$ for general λ . *Machine Learning*, 8(3):341–362.
- [Dayan and Sejnowski, 1994] Dayan, P. and Sejnowski, T. (1994). $td(\lambda)$ converges with probability 1. *Machine Learning*, 14(3):295–301.
- [Grossberg, 1973] Grossberg, S. (1973). Contour enhancement, short term memory, and constancies in reverberating neural networks. *Studies in Applied Mathematics*, L11:213–257.

- [Grossberg, 1976] Grossberg, S. (1976). Adaptive pattern classification and universal recoding, ii : Feedback, expectation, olfaction, and illusions. *Biological Cybernetics*, 23:187–202.
- [Grossberg, 1988] Grossberg, S. (1988). Nonlinear neural networks: principles, mechanisms, and architectures. *Neural Networks*, 1:17–61.
- [Grossberg and Mingolla, 1985] Grossberg, S. and Mingolla, E. (1985). Neural dynamics of form perception: Boundary completion, illusory figures and neon color spreading. *Psych.Rev.*, 92(2):173–211.
- [Grossberg and Somers, 1991] Grossberg, S. and Somers, D. (1991). Synchronized oscillations during cooperative feature linking in a cortical model of visual perception. *Neural Networks*, 4:453–466.
- [Hebb, 1949] Hebb, D. (1949). *The Organization of Behavior*. Wiley, New York.
- [Hubel and Wiesel, 1977] Hubel, D. and Wiesel, T. (1977). Ferrier lecture: Functional architecture of macaque monkey visual cortex. *Proc. Roy. Soc. Lond. B. Biol. Sci.*, 198:1–59.
- [Knudsen and Konishi, 1979] Knudsen, E. and Konishi, M. (1979). Mechanisms of sound localization in the barn owl (*tyto alba*). *Journal of Comparative Physiology*, 133:13–21.
- [Kohonen, 1984] Kohonen, T. (1984). *Self-Organization and Associative Memory*. Springer-Verlag, New York.
- [Kohonen, 1989] Kohonen, T. (1989). *Self-Organization and associative memory*. Heidelberg: Springer-Verlag, Berlin, 3rd edition.
- [Kohonen, 1993] Kohonen, T. (1993). Physiological interpretation of the self-organizing map algorithm. *Neural Networks*, 6:895–905.
- [Lippman, 1987] Lippman, R. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, pages 4–22.
- [Rosenblatt, 1957] Rosenblatt, F. (1957). The perceptron: A perceiving and recognizing automaton (project PARA). Technical Report 85-460-1, Cornell Aeronautical Laboratory.
- [Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408.
- [Rosenblatt, 1962] Rosenblatt, F. (1962). *Principles of Neurodynamics*. Spartan Books, New York.

- [Rosenzweig and Leiman, 1992] Rosenzweig, M. and Leiman, A. (1992). *Psychopsysiologie*. Intereditions, 2ème edition.
- [Rumelhart and Zipser, 1985] Rumelhart, D. and Zipser, D. (1985). Feature discovery by competitive learning. *Cognitive Science*, 9:75–112.
- [Sutton, 1988] Sutton, R. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.
- [Sutton, 1991] Sutton, R. (1991). Planning by incremental dynamic programming. In *Eight International Workshop on Machine Learning*, pages 353–357. Morgan Kaufmann.
- [Watkins, 1989] Watkins, C. (1989). *Learning from delayed rewards*. PhD thesis, Psychology Department, Cambridge University, Cambridge, England.
- [Widrow and Hoff, 1960] Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. In *IRE WESCON*, pages 96–104, New York. Convention Record.
- [Woolsey, 1981a] Woolsey, C. (1981a). Cortical sensory organization: Multiple auditory areas. *Crescent Manor, N.J.: Humana*.
- [Woolsey, 1981b] Woolsey, C. (1981b). Cortical sensory organization: Multiple somatic areas. *Crescent Manor: N.J.: Humana*.
- [Woolsey, 1981c] Woolsey, C. (1981c). Cortical sensory organization: Multiple visual areas. *Crescent Manor, N.J.: Humana*.