Intelligence Artificielle
Introduction

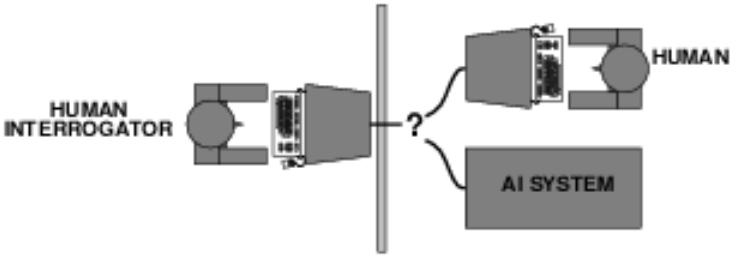
Fondations de l’IA

- Philosophie: comment l’intelligence fonctionne
  - Descartes: séparation monde réel/spirituel
- Mathématiques: la logique
- Économie: décision / optimisation (Nash)
- Psychologie : les comportements
- Neurophysiologie : le cerveau (W. McCulloch/W. Pits)
- Traitement de l’information : C. Shannon
- Informatique : J. von Neuman, A. Turing
- Cybernétique (N. Wiener)

Test de Turing

Turing (1950) "Computing machinery and intelligence":

- ♢ "Can machines think?" ⟶ "Can machines behave intelligently?"
- ♢ Operational test for intelligent behavior: the Imitation Game



- ♢ Predicted that by 2000, a machine might have a 30% chance of fooling a lay person for 5 minutes
- ♢ Anticipated all major arguments against AI in following 50 years
- ♢ Suggested major components of AI: knowledge, reasoning, language understanding, learning

Problem: Turing test is not reproducible, constructive, or amenable to mathematical analysis

Les approches IA

- IA s’est une *simulation*
  1. On peut simuler le mécanisme de l'intelligence, c'est-à-dire d'implanter dans les machines la même façon de résoudre des problèmes que les humains.
  2. *On peut aussi simuler le comportement extérieur, c'est-à-dire de faire en sorte que le résultat produit par une machine soit comparable à celui d'un humain.*

Que faut-il dans un système d’IA ?

- Des bases de connaissances
  - Implicitement: avec une représentation

- Des mécanismes d’exploitation de ces connaissances (raisonnement)

Les connaissances : différentes formes du savoir

- Objets du monde, propriétés de ces objets
- Classifications
- Règles heuristiques de savoir-faire
- Connaissances de bon sens (loi immuables)
- Méta-connaissances (connaissances sur les connaissances)

Pour exploiter des connaissances

- Il faut choisir un formalisme de représentation des connaissances

- Ce formalisme doit permettre de désigner une unité de connaissance

- Il doit permettre au raisonnement de produire de nouvelles connaissances

Modèle conceptuel: représentation simplificatrice

Quelques modes de représentation des connaissances en IA

- Représentations logiques : SI …ALORS
- Frames : assemblages d'objets définissant un contexte
- Les scripts : description d'une séquence d'événements
- Réseaux de concepts (ontologies) …

IA … quels problèmes ?

- Pas de méthode de résolution adaptée …
- Si:
  - Description formelle du problème
  - Procédure pour tester une solution proposée
  - On peut engendrer et énumérer les solutions potentielles

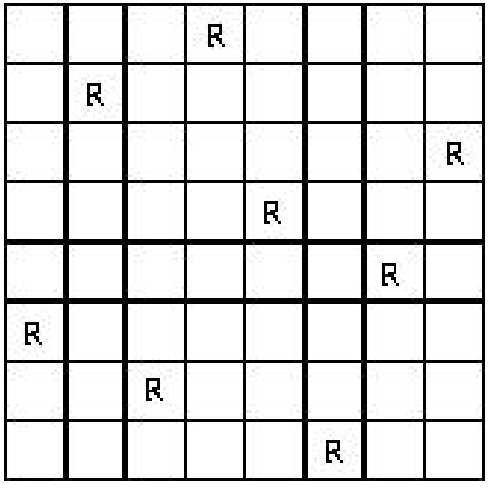
Énumération + test =
procédure constructive de résolution

Deux méthodes de résolution constructives

- Augmenter le nombre des solutions partielles
  - Exemple : problème des 8 reines
- Décomposer le problème en sous-problèmes
  - Exemple : les tours de Hanoi

Problème des huit reines

- Objectifs:
  - placer 8 reines sur un échiquier 8 x 8
  - sans prise possible



Une solution au problème des 8 reines

Exemple - problème des 8 reines

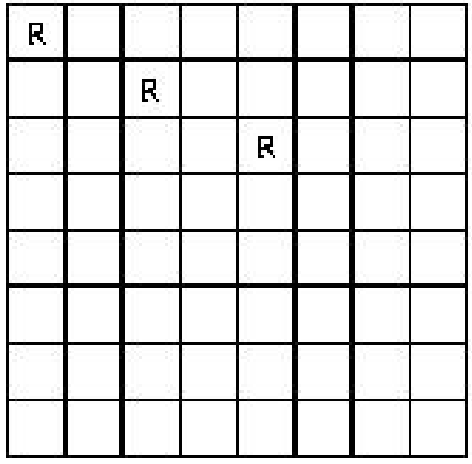
Approche constructive - augmenter les solutions partielles

- Systématiser la recherche en construisant les configurations de l'échiquier pas à pas:
  - commencer avec un échiquier vide
  - tenter de placer les 8 reines une par une
  - en respectant à chaque étape les contraintes du problème
    - placer une kème reine sur un échiquier où figurent déjà k-1 reines
    - tester les solutions partielles au fur et à mesure; elles peuvent être retenues comme des points de départs pour les étapes suivantes
  - jusqu'à l'obtention d'une solution

Exemple - problème des 8 reines

Approche constructive - augmenter les solutions partielles

- Respecter à chaque étape les contraintes du problème
  - placer la première reine sur la première ligne
  - placer la seconde reine sur la seconde ligne
  - etc.



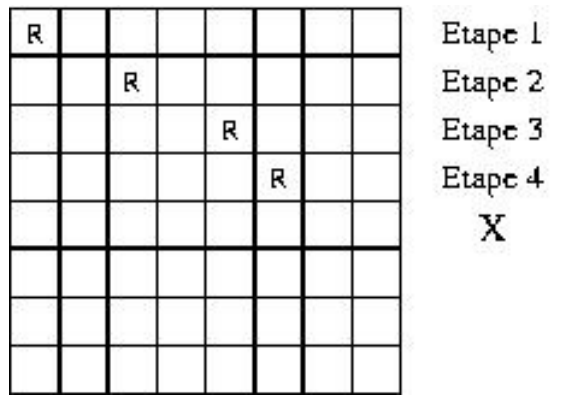
→Réduire à 8 le nombre de positions alternatives pour chaque reine

Exemple - problème des 8 reines

Approche constructive - augmenter les solutions partielles

Impossibilité de revenir sur le non-respect des contraintes avec des opérations ultérieures :

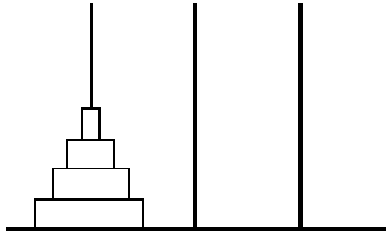
- si une configuration partielle ne respecte pas une des contraintes
- elle ne peut pas être rectifiée par l'ajout d'une reine



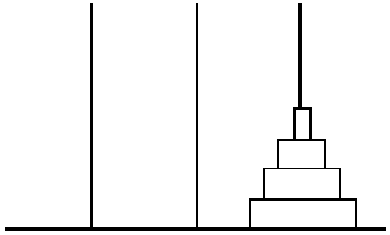
Élimination de nombreuses configurations inutiles.

Les tours de Hanoi

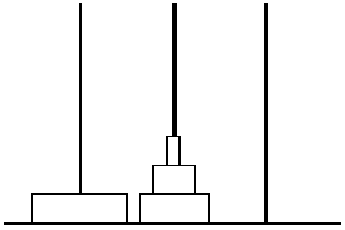
Approche constructive - Décomposer le problème en sous-problèmes



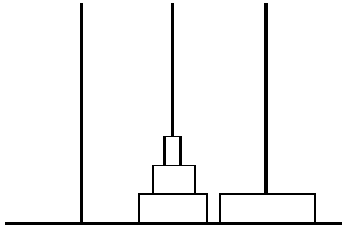
Situation initiale



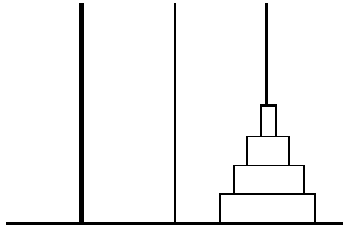
Objectif



Sous-problème 1



Sous-problème 2



Sous-problème 3

Procédure de résolution par transformation

- partir d'une configuration quelconque
- la tester et, si elle n'est pas solution la transformer localement
- recommencer à partir de la nouvelle configuration

Exemple - problème des 8 reines
Transformation de solution potentielle

Commencer avec une configuration initiale de 8 reines :

- tester la configuration
- si ce n'est pas une solution, la transformer localement (en déplaçant une reine en prise)
- etc.
- de façon récurrente jusqu'à l'obtention d'une solution

- chercher à se rapprocher d'une solution en suivant une série de décisions locales
- s'assurer que la séquence des transformations n'est pas aléatoire mais systématique
  - ne pas générer indéfiniment la même configuration (contrainte d'efficacité)
  - ne pas rater l'occasion de générer une configuration solution (contrainte d'utilité)

Résolution d’un problème

- Selon Newell et Simon (dans leur discours pour leur prix Turing, 1976), les activités intelligentes de la machine ou de l'homme se réalise par les trois moyens suivants:

1. La création d'une représentation symbolique du domaine d'application. Dans cette représentation, les aspects significatifs du problème est représentés par des patrons symboliques.
2. Les opérations sur les patrons symboliques. Ces opérations permettent de relier différents états du problème, ainsi de créer une structure parmi tous les états possibles du problème.
3. Une recherche dans la structure créée afin de trouver une solution.

Ils introduisent aussi la représentation graphique … d’un problème.

Caractérisation d’un problème

- États (initial, final)
- Actions possibles: {(état,action)}→ état
  - Espace d’états: 1+2
  - Chemin: séquence des états (+actions)
  - Coût (action,chemin)
- Test de fin
  - Solution (optimale)

Représentation par graphes d'états

- On représente l'ensemble des états du problème par un graphe orienté étiqueté
  - Les sommets sont les états du problème
  - Il existe un arc (u,v) si un opérateur transforme l'état u en état v, étiquette(u,v) = opérateur
- Recherche d'une solution = recherche d'un chemin entre le nœud initial et un nœud terminal
- Solution = séquence des opérateurs étiquetant les arcs de ce chemin

Choisir une bonne représentation des états
Problème des huit reines

- Description par un tableau:
  - État : échiquier(i,j) $\in$ {vide,reine}
- Un seul opérateur
  - opérateur : placer-reine(i,j) :

    si échiquier(i,j) = vide, alors échiquier(i,j) $\leftarrow$ reine
- 2 tests complètent la représentation:
  - échiquier-sans-prise
  - échiquier-complet

Les 8 reines

Construction du graphe d'état :

- du nœud initial n0 (échiquier vide) partent 64 arcs (placer une reine)
- de chacun de ses successeurs partent 63 arcs (placer deux reines)
- etc.

Le graphe d'état :

- ne comporte pas de circuit (une reine placée ne peut plus être enlevée)
- comporte des cycles (il est possible d'atteindre le même état de diverses façons)

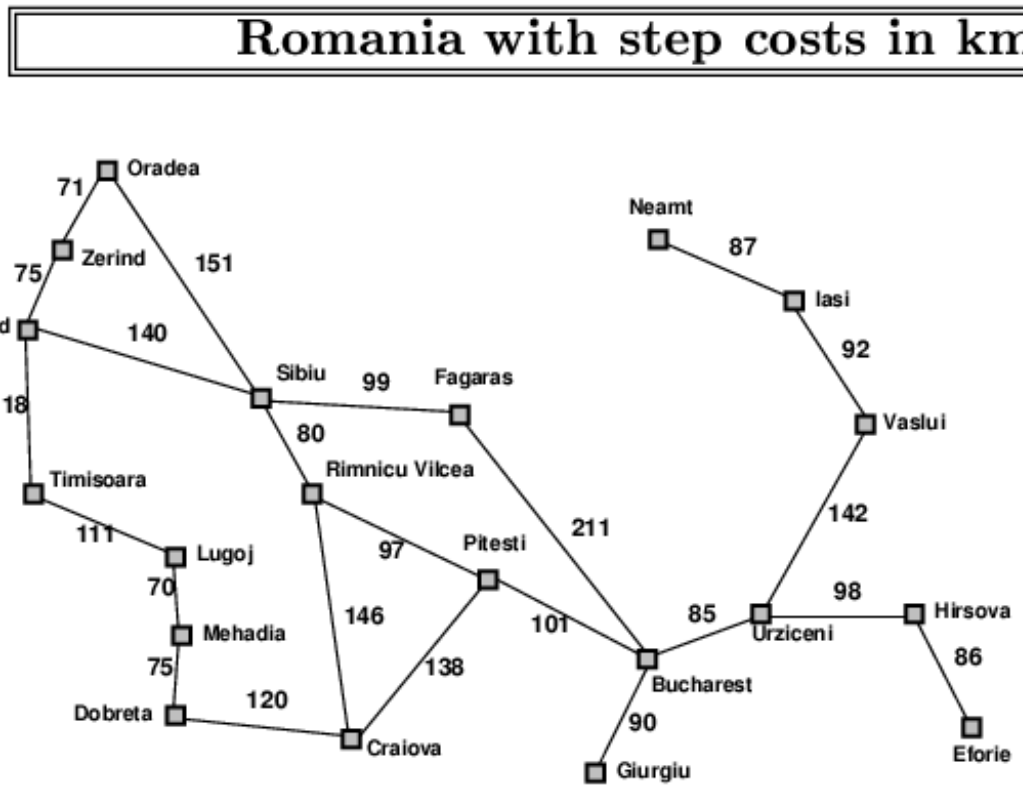
Les 8 reines

- Solution : chemin SC du nœud initial n0 à un nœud quelconque vérifiant les deux tests :
  - tous les nœuds de SC doivent vérifier la condition *sans prise*
  - une grande partie des 2^{64} états est donc inutile : tout état avec une reine en prise ainsi que tout successeur d'un tel état
- Amélioration :
  - inclure le test *sans prise* dans les conditions d'application de l'opérateur *placer-reine*
  - les états ne vérifiant pas ce test restent descriptibles mais ne sont plus atteignables à partir de n0 et du seul opérateur disponible
- Les deux représentations ne différent que de la position du test et peuvent être considérées comme équivalentes.

Stratégies d’organisation de la recherche

- Développement d’état : à chaque étape, l’état choisi sera
  - complètement développé
  - partiellement développé
- Organisation des alternatives : l’ensemble des états à développer est organisé dans
  - une pile $\rightarrow$ recherche en profondeur (LIFO)
  - une file $\rightarrow$ recherche en largeur (FIFO)
  - une liste selon un coût croissant

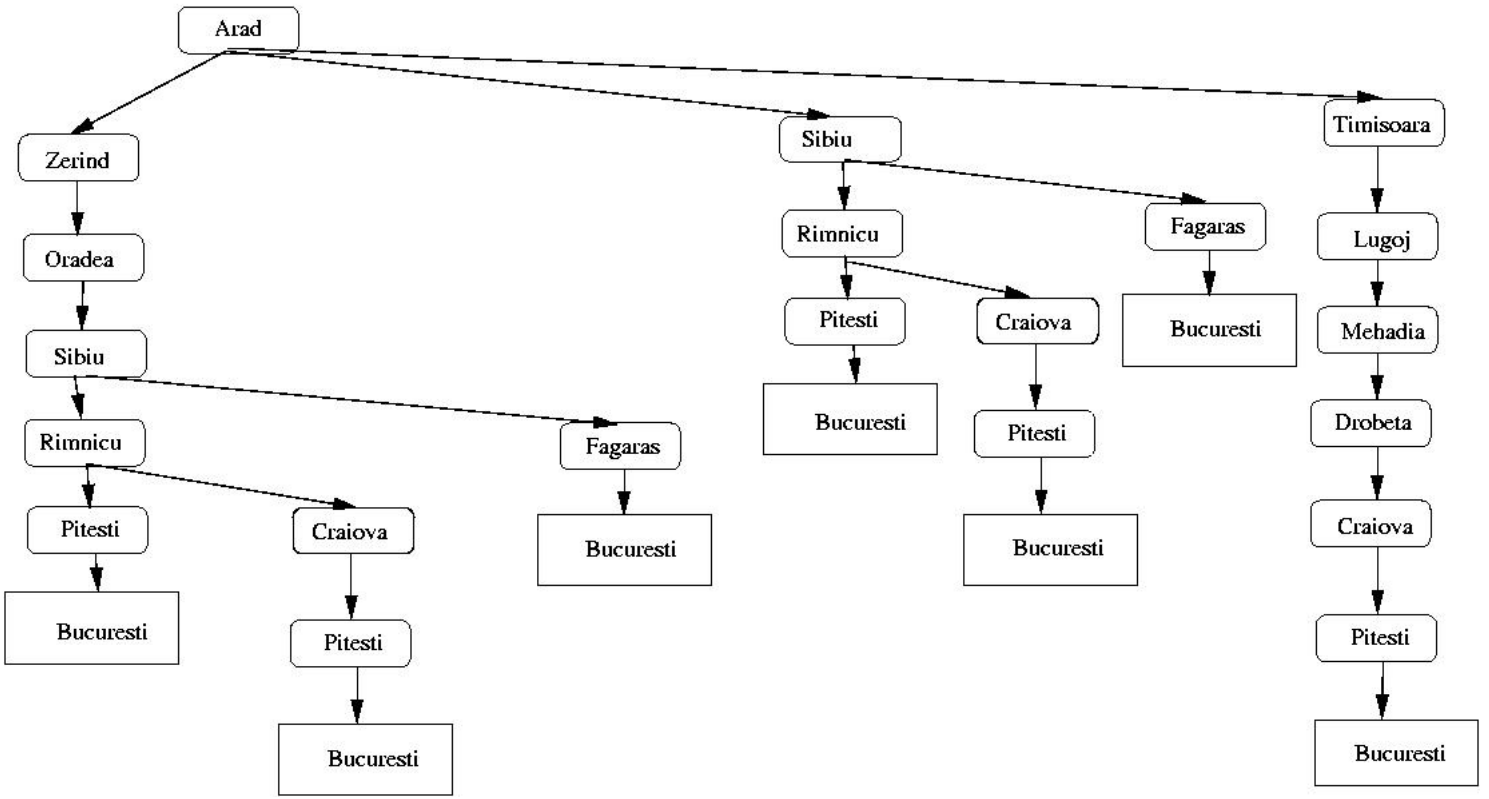
Recherche en largeur d'abord



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

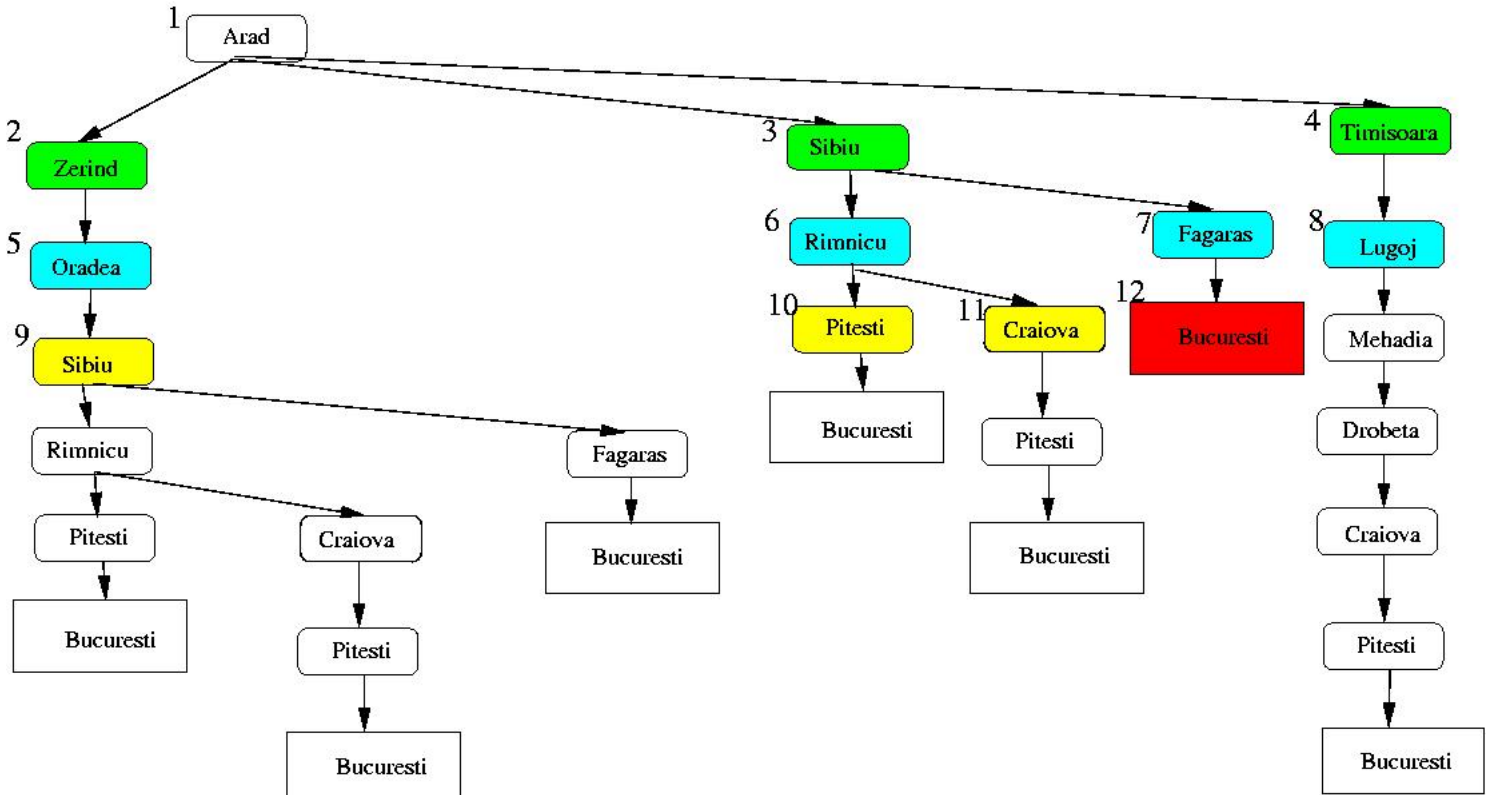
Arad →Bucuresti …

Construction du graphe d'états (partiel)



Recherche en largeur d'abord

Arad →Bucuresti …

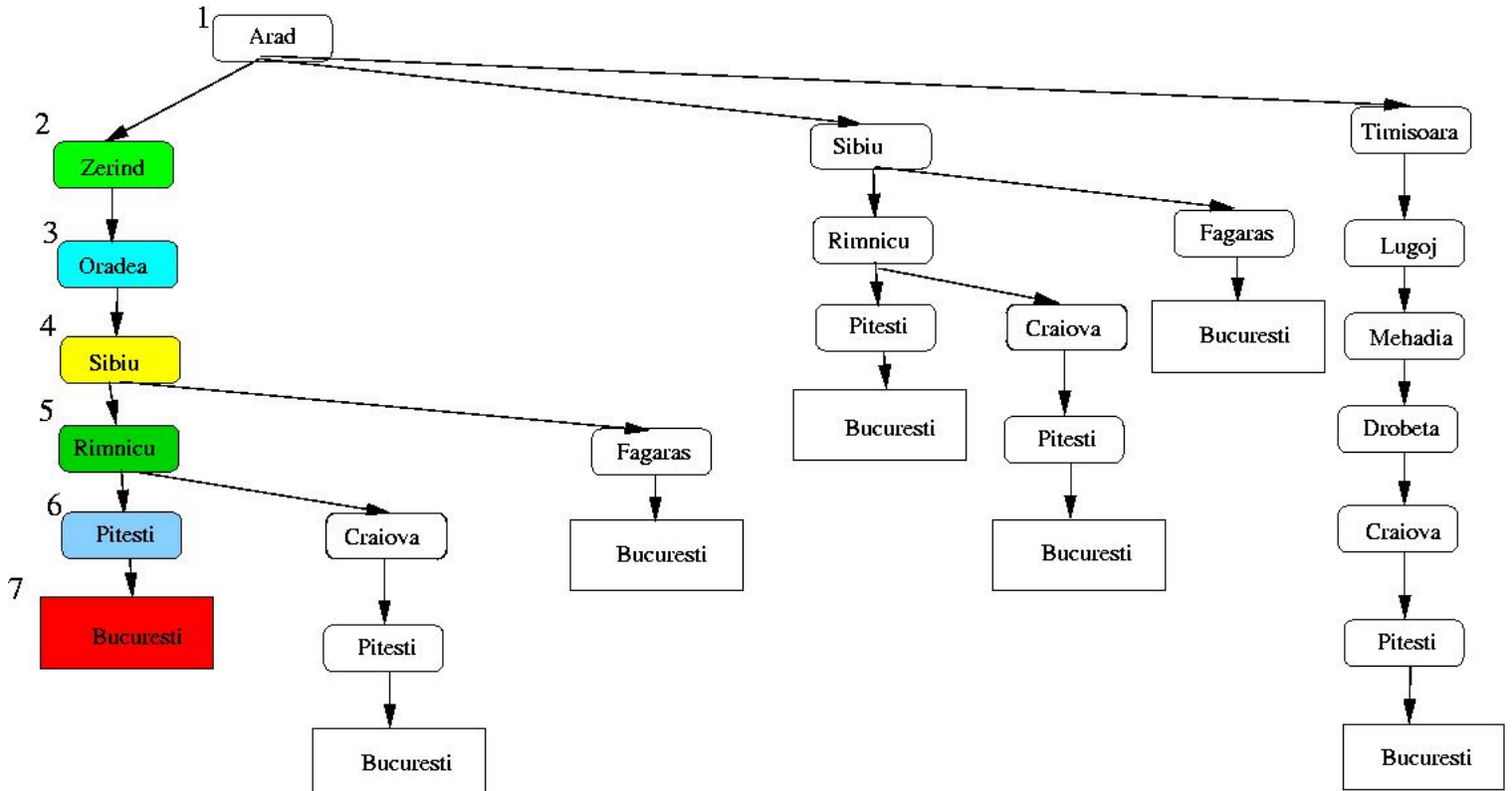


Recherche en largeur d'abord

- En moyenne b (facteur de branchement) successeurs pour chaque nœud (trouver un chemin solution de longueur d)
  - explorer 1+b+b^2 + b^3 + …+ b^d noeuds.
- Complétude : oui si b est fini
- complexité:
  - Temps : 1+b+b^2 + b^3 + …+ b^d = O(b^d)
  - Mémoire : O(b^d)
  - Optimalité : non

Recherche en profondeur d'abord

Arad →Bucuresti …



Recherche en profondeur d'abord

- Complétude : oui si espace de recherche fini
- Complexité :
  - Temps : O(b^m) (m profondeur maximum de l'espace de recherche)
  - mémoire~: O(bm) (linéaire)
- optimalité : non en général, oui si on a beaucoup de chance

- Variante : fixer une profondeur limitée!

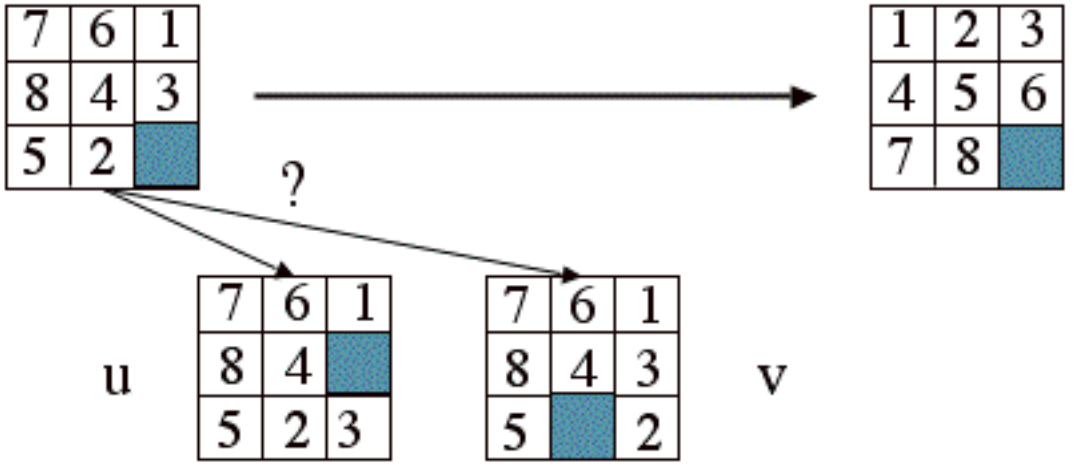
Notion d’heuristique

- Résolution de problèmes par développement et choix d’une alternative $\Rightarrow$ combinatoire
- Un algorithme de recherche doit guider la recherche d’une solution en faisant des choix et en gérant le retour sur ces choix tout en évitant l’explosion combinatoire
- Une *heuristique* est un moyen de guider les choix que doit faire l’algorithme pour réduire sa complexité, en ordonnant la liste des successeurs d’un état

Définir une heuristique

- Une heuristique est spécifique à un problème, elle ne peut pas être généralisée !
- Elle est souvent fondée sur une simplification du problème, grâce à une relaxation des contraintes difficiles
  - Simplification grossière $\rightarrow$ heuristique pauvre et peu efficace
  - Simplification élaborée $\rightarrow$ heuristique plus complexe à évaluer mais plus efficace

Exemple d'heuristique : le jeu du taquin



Choisir = apprécier l'écart à l'objectif

h1 : nombre de pièces mal placées

h2 : somme des déplacements minimaux pour amener une pièce à sa place

h2 est plus efficace que h1

Recherche heuristique dans les graphes d'états

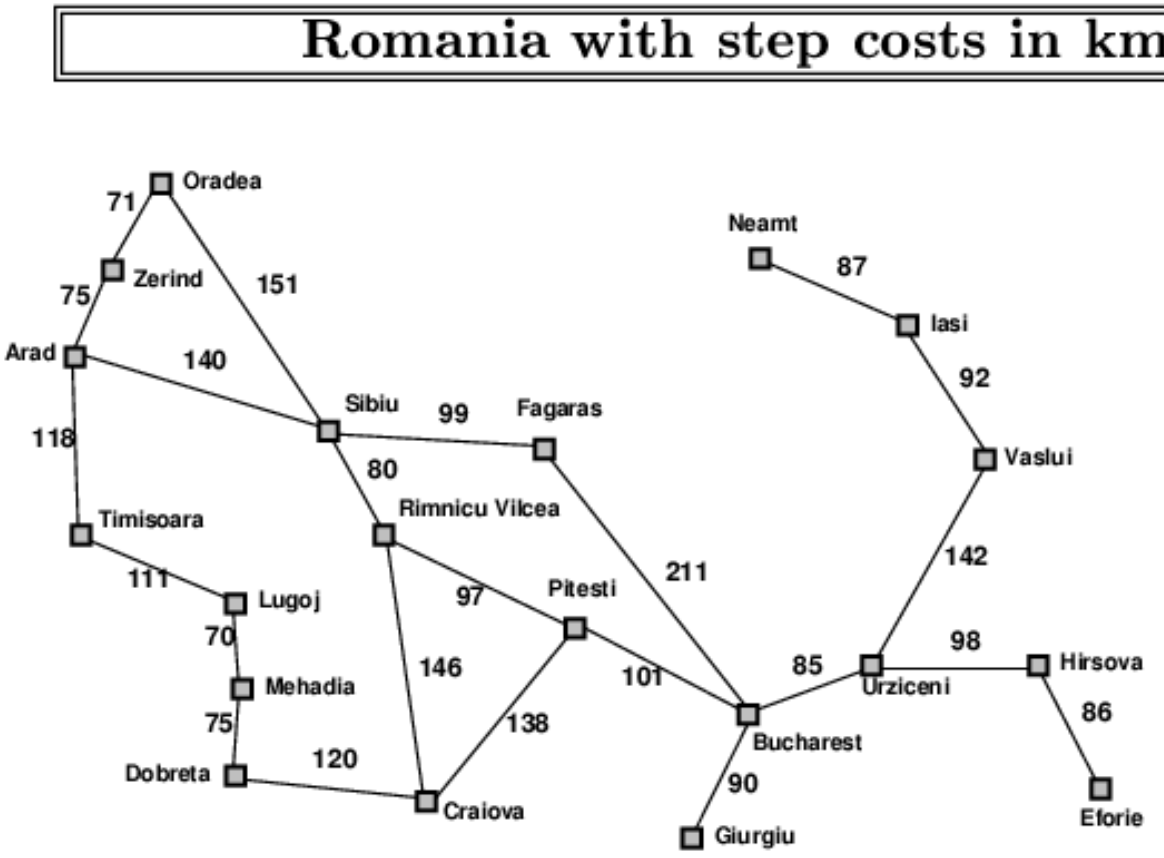
- Pour tout nœud u, on définit :
  - g*(u) le minimum du coût des chemins de u_0 à u
  - h*(u) le minimum du coût des chemins de u à un état terminal quelconque
  - f*(u) = g*(u) + h*(u) le coût du chemin solution optimal
- Pour ordonner la recherche, on utilise :
  - Une heuristique h(u) qui estime h*(u)
  - g(u) le coût du meilleur chemin connu pour aller jusqu'à u (estime g*(u))
  - f(u) = g(u) + h(u) la fonction d'évaluation

L’algorithme A*

- L’algorithme A* utilise $f=g+h$ pour ordonner les nœuds à développer

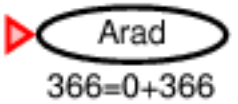
- Si le graphe est fini, A* s’arrête et fournit un chemin solution, s’il existe.
- Si $\forall u$, $h(u) = h^*(u)$, A* fournit la meilleure solution

Exemple: voyager en Roumanie



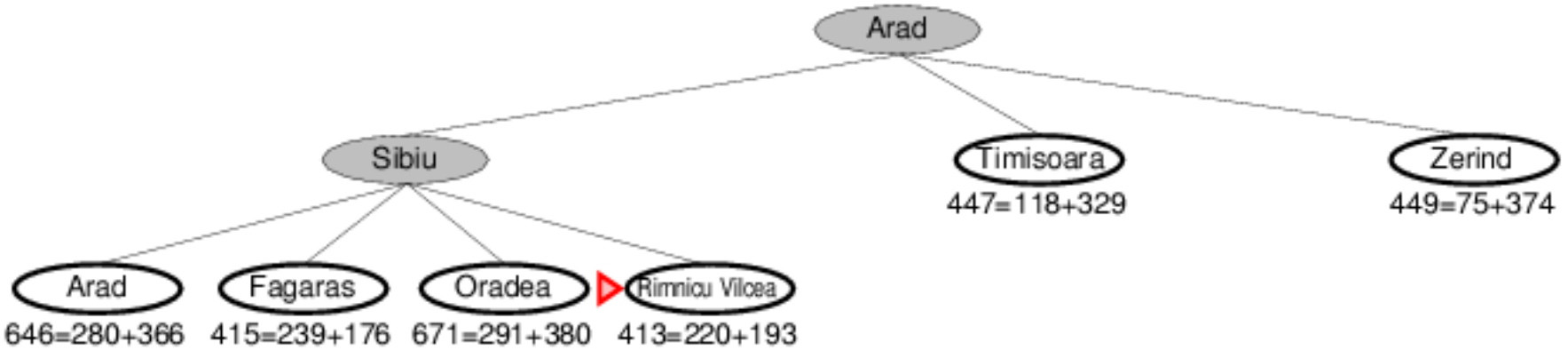
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

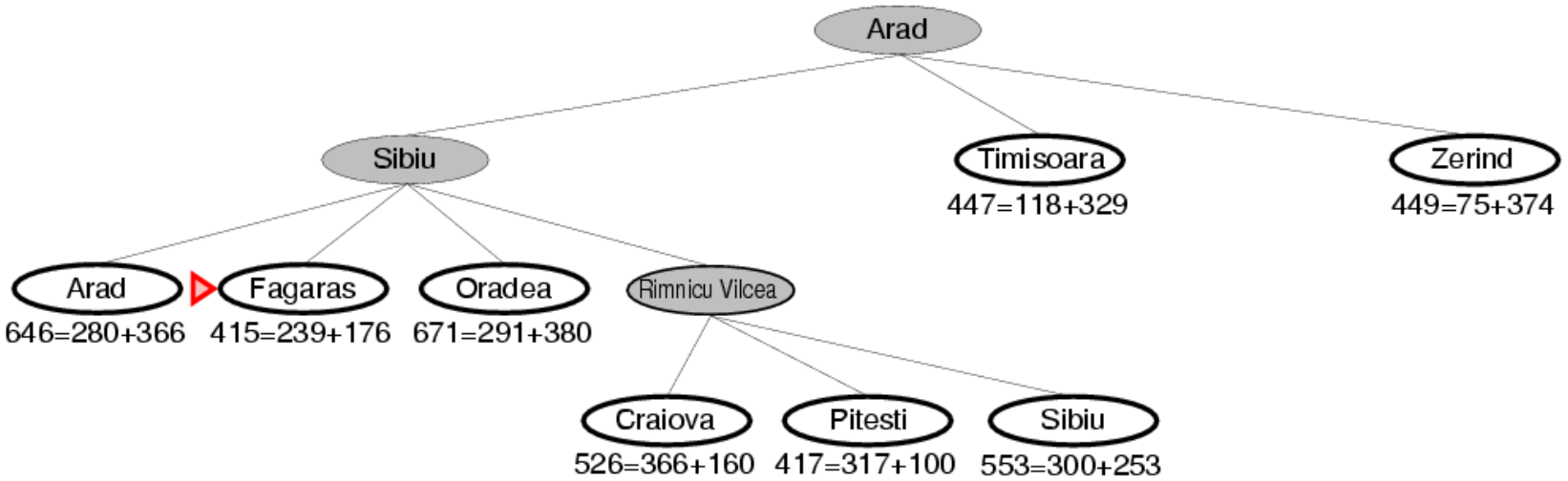


A* search example

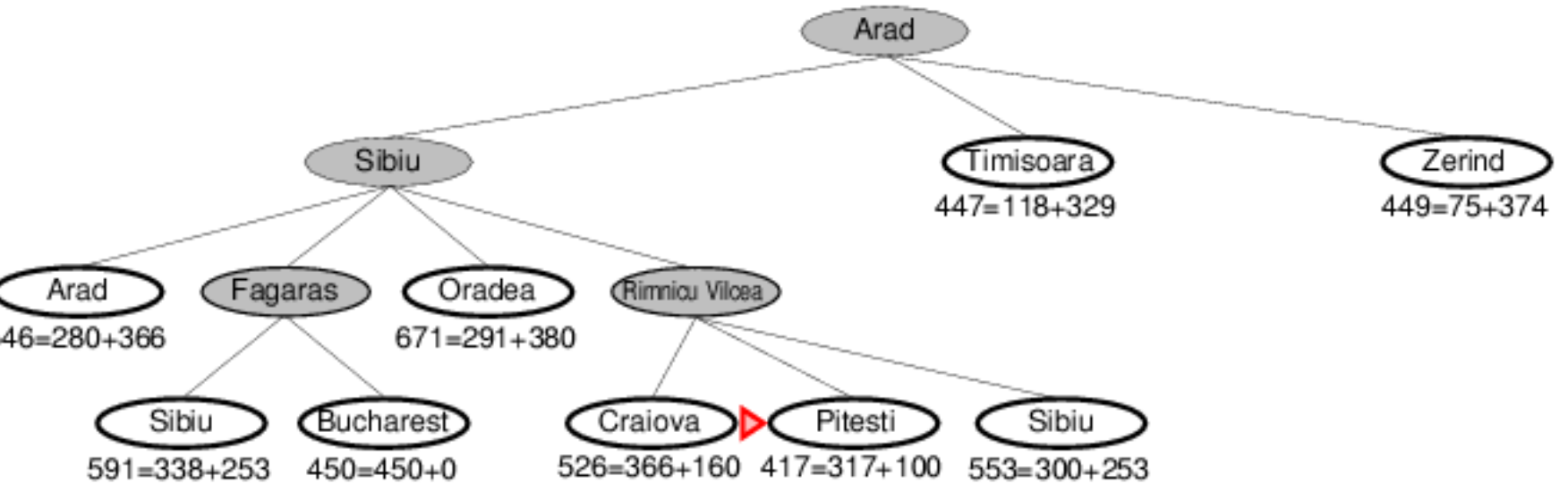
A* search example



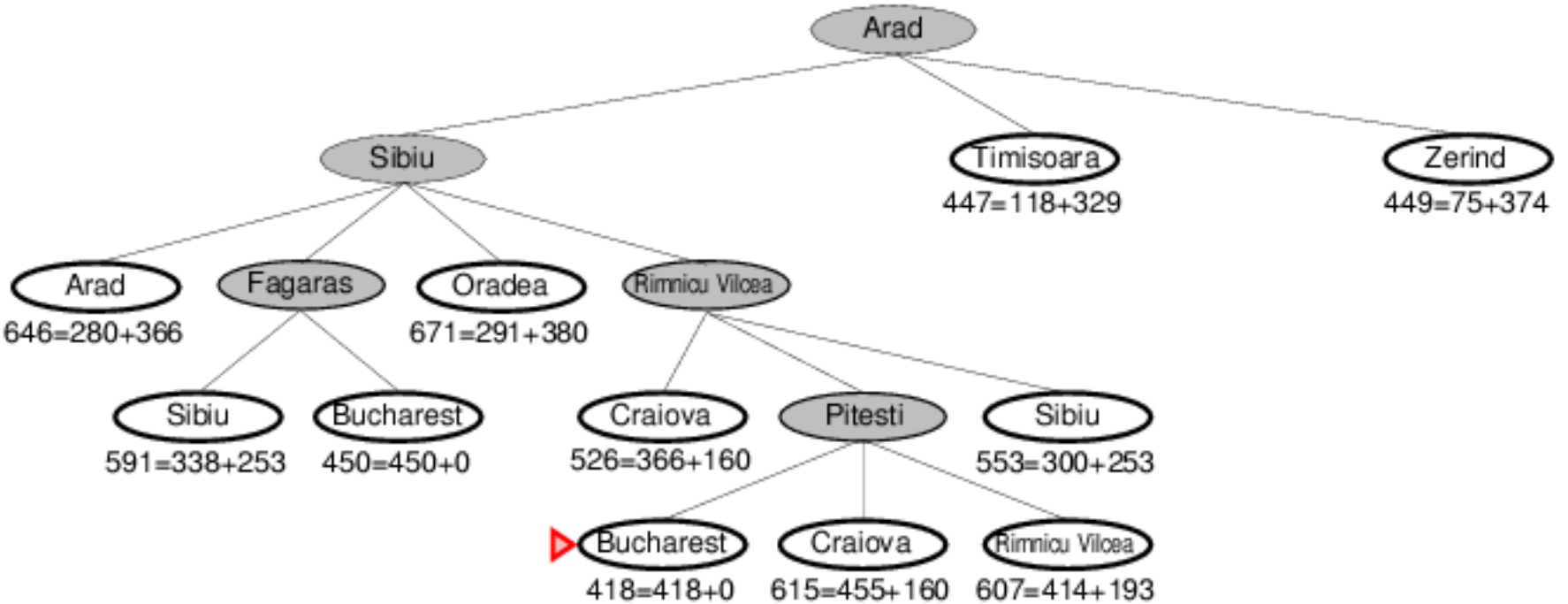
A* search example



A* search example



A* search example



Algorithmes de recherche

- Terminaison
  - L’algorithme trouve une solution
- Admissibilité
  - Un algorithme est admissible s'il trouve toujours la meilleure solution si elle existe.
- Complexité
  - Mémoire
  - Nb. d’opérations

Représentation par graphes de sous-problèmes

- État initial ou courant : problème à résoudre
- Opérateur : décomposition en plusieurs sous-problèmes
- États terminaux : problèmes triviaux

- Représentation par un graphe ET/OU sans circuit :
  - nœuds OU associés aux problèmes
  - nœuds ET de décomposition

Décomposition de problème

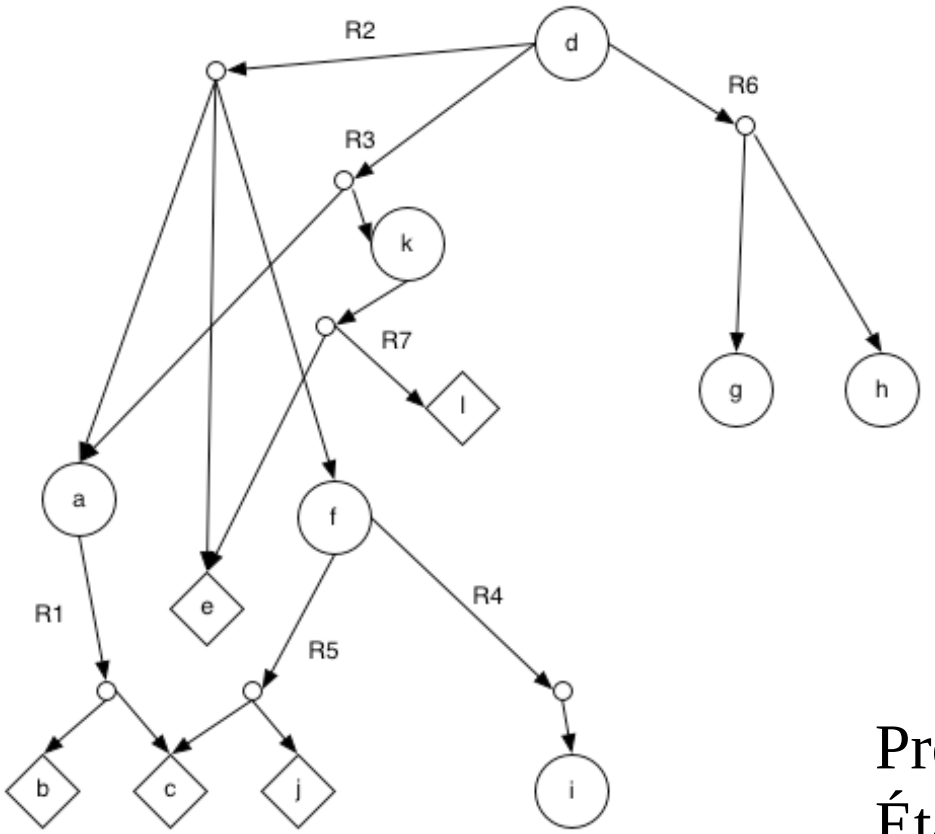
Ensemble de règles de décomposition de problème en sous-problèmes

a → b,c
d → a,e,f
d → a,k
f → i
f → c,j
d → g,h
k → e,l

Problème à résoudre :*d*
États finaux: *b,c,e,j,l*

Diff: les opérateurs sont des applications multivoques, la transformation d'un état u en plusieurs états ui
Les difficultés (choix des données, opérateurs, etc.) sont les mêmes

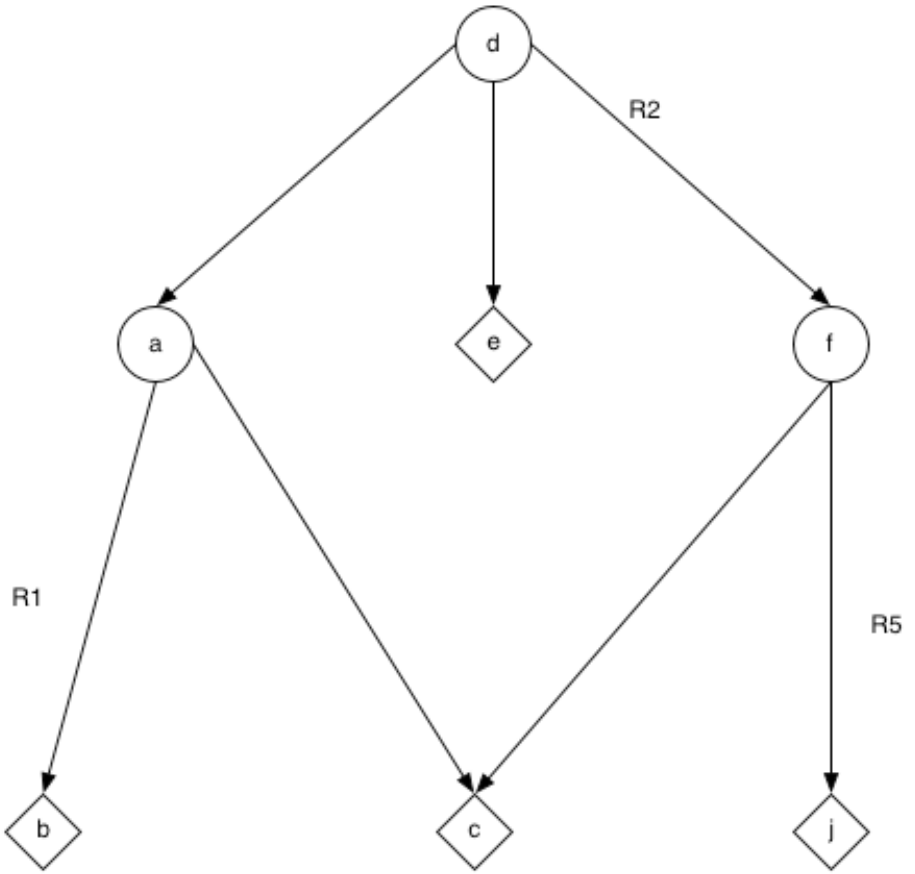
Graphe Et/Ou



R1 : a → b,c
R2 : d → a,e,f
R3 : d → a,k
R4 : f → i
R5 : f → c,j
R6 : d → g,h
R7 : k → e,l

Problème à résoudre :*d*
États finaux: *b*,*c*,*e*,*j*,*l*

Utilisation graphe Et/Ou (une solution)



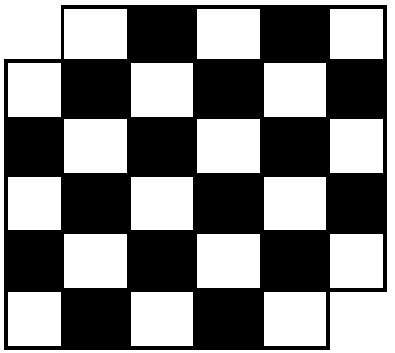
Principe de l'algorithme de recherche d'un sous-graphe ET/OU

- On définit
  - des états RÉSOLUS :
    - les états terminaux
    - des états dont un connecteur a tous ses fils résolus
  - des états INSOLUBLES :
    - les états non terminaux sans successeur
    - les états dont tous les connecteurs ont au moins un successeur insoluble
- On développe en profondeur un sous-graphe représentant une solution partielle avec retour arrière en cas d'échec

Améliorer l’efficacité de la recherche

- On peut introduire un seuil sur le rang des nœuds pour réduire l’espace exploré
- On peut exploiter une information heuristique pour
  - ordonner l’examen des connecteurs d’un nœud
  - interrompre la récursivité sur des états peu prometteurs en les rendant insolubles

Problème de l’échiquier écorné



À recouvrir de dominos 

Logique prédicative

- Aspects syntactique
  - Connecteurs logiques : $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\equiv$, $\forall$ et $\exists$
  - Un ensemble de variables : *x*, *y*, *z*, …
  - Symboles non logiques
    - Symboles fonctionnels : *f*, *g*, *h*, …
    - Symboles de relations (prédicats) : *P*, *Q*, *R*, …
  - Atomes : *P(x, f(y,z), z)*
  - Formules: $\forall x \; \exists y \;\; P(x, y) \Rightarrow Q(y, y)$
- Sémantique
  - Attribution des valeurs de vérité (vrais/faux) aux énoncés élémentaires →interprétation

Application de la logique dans l'IA

- pour représenter des faits, des connaissances
  - Faits: le block *a* est posé sur le block *b* : *sur(a,b)*
  - Connaissances : le fait que le block *a* est sur le block *b* implique que le block *b* n'est pas libre :

$$\forall X \forall Y \; sur(X,Y) \Rightarrow \neg \; libre(Y).$$

- utiliser la logique pour inférer des conclusions
  - Le bloque *b* n’est pas libre: $\neg$ libre(b)

Comment représenter des faits et des connaissances véhiculés en langue naturelle

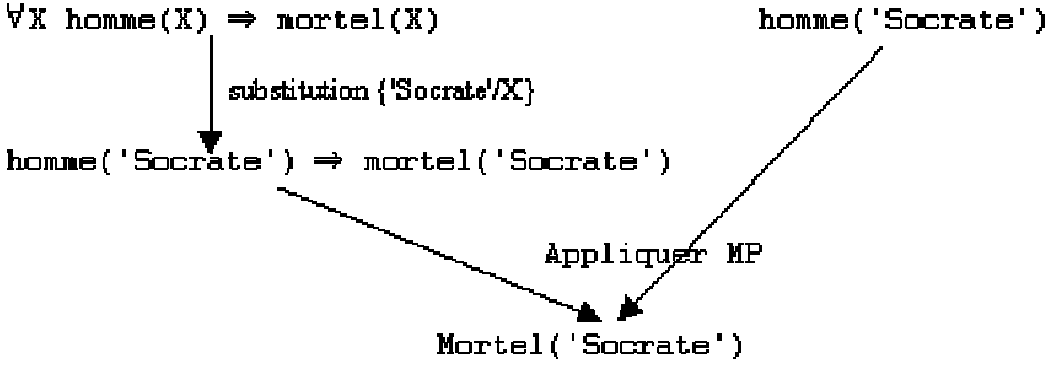
- Un objet est de certaine catégorie
  - Ex: Jean est un homme : *homme('Jean')*
- Un groupe nominal
  - un homme*: $ X homme(X)*
- Un fait
  - Un homme marche*: $ X homme(X) Ù marche(X).*
- Une connaissance (observation généralisée)
  - Tout homme marche *: " X homme(X) Þ marche(X).*

Règles de production

Si CONDITION Alors ACTION

CONDITION ou NON(ACTION)

Ex: « tout homme est mortel » et « Socrate est un homme »



Un exemple de base de règles : un système de conseil en voyages

R1 : SI distance<2km ALORS aller_à_pied
R2 : SI ¬distance<2km ET distance<300km ALORS
prendre_le_train
R3 : SI ¬distance<300km ALORS prendre_l'avion
R4 : SI acheter_un_billet_d'avion ET avoir_le_téléphone
ALORS téléphoner_à_l'agence
R5 : SI acheter_un_billet_d'avion ET ¬avoir_le_téléphone
ALORS aller_à_l'agence
R6 : SI prendre_l'avion ALORS
acheter_un_billet_d 'avion
R7 : SI durée>2jours ET être_fonctionnaire ALORS
¬prendre_l'avion

Architecture d’un système à base de règles

- La base de faits contient, pour une situation donnée, les faits avérés ou à établir
  - F1 : ¬distance<300km
  - F2 : avoir-le-téléphone
- Base de règles
- Le moteur d’inférence est un programme qui exploite la base de règles pour déduire de nouveaux faits. Il détient le contrôle du raisonnement.

Mon premier moteur d'inférence

- Boucle sur les règles : soit R une règle
  - ***Si*** les prémisses de R appartiennent à BF
  - ***Alors*** ajouter les conclusions de R à BF
  - **FinSi**
- FinBoucle
- R3 et F1 $\rightarrow$ F3 : prendre_l'avion
- R6 et F3 $\rightarrow$ F4 : acheter_un_billet_d'avion
- Problème : ne déclenche pas R4

Mon deuxième moteur d'inférence

- Changement ← vrai
- Tant que changement
  - Changement ← faux
  - Boucle sur les règles : soit R une règle
    - Si les prémisses de R appartiennent à BF
    - Alors ajouter les conclusions de R à BF
      changement ← vrai
    - FinSi
  - FinBoucle
- FinTQ

- R3 et F1 → F3, R6 et F3 → F4, R3 et F1 → F3
- R4 et F4 et F2 → F5 : téléphoner_à_l'agence, R6 et F3 → F4, ...
- Problème : les règles s'appliquent plusieurs fois
- Il faut donc « marquer » les règles déclenchées pour ne plus les considérer dans la boucle

Mon troisième moteur d'inférence

- Changement ← vrai
  Tant que changement
  Changement ← faux
  Boucle sur les règles : soit R une règle
  Si R n'est pas marquée et si les prémisses de R appartiennent à BF
  Alors ajouter les conclusions de R à BF
  changement ← vrai
  marquer R
  FinSi
  FinBoucle
  FinTQ

R3 et F1 → F3

R6 et F3 → F4

R4 et F4 et F2 → F5

Exemple

K, L, M → I
I, J, L → Q
C, D, E → B
A, B → Q
L, N, O, P → Q
C, H → R
R, J, M → S
F, H → B
G → F

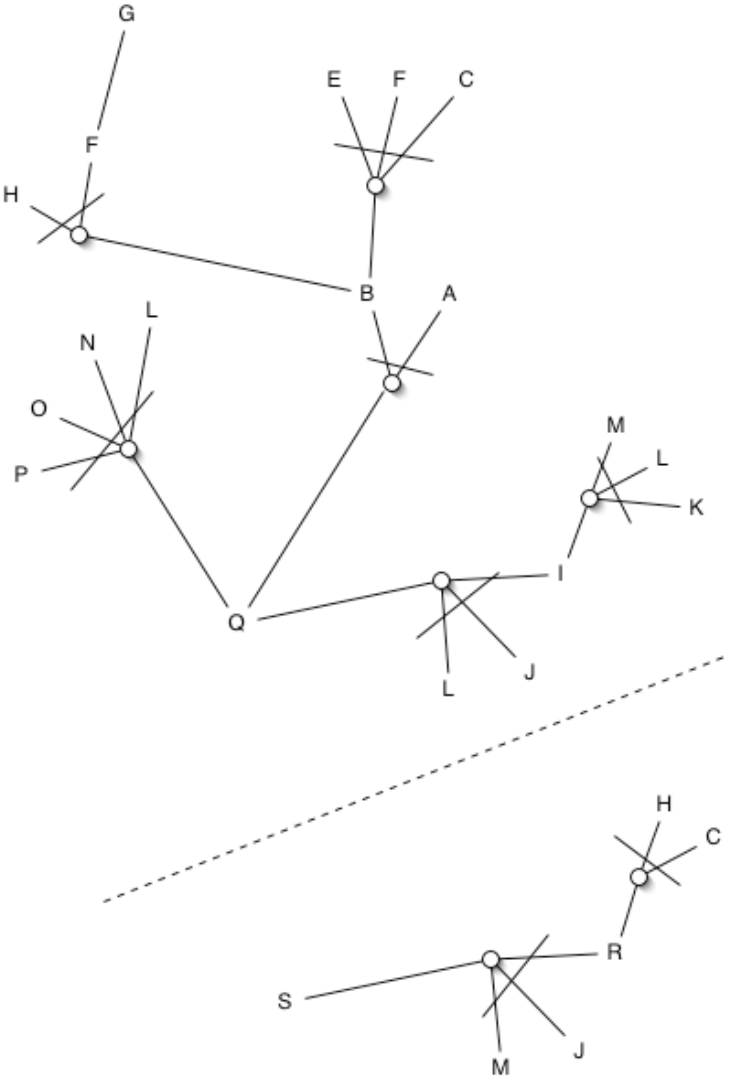
A, C, D, E, G, H, K

Base de règles

(prémisses →conclusion)

Base de faits

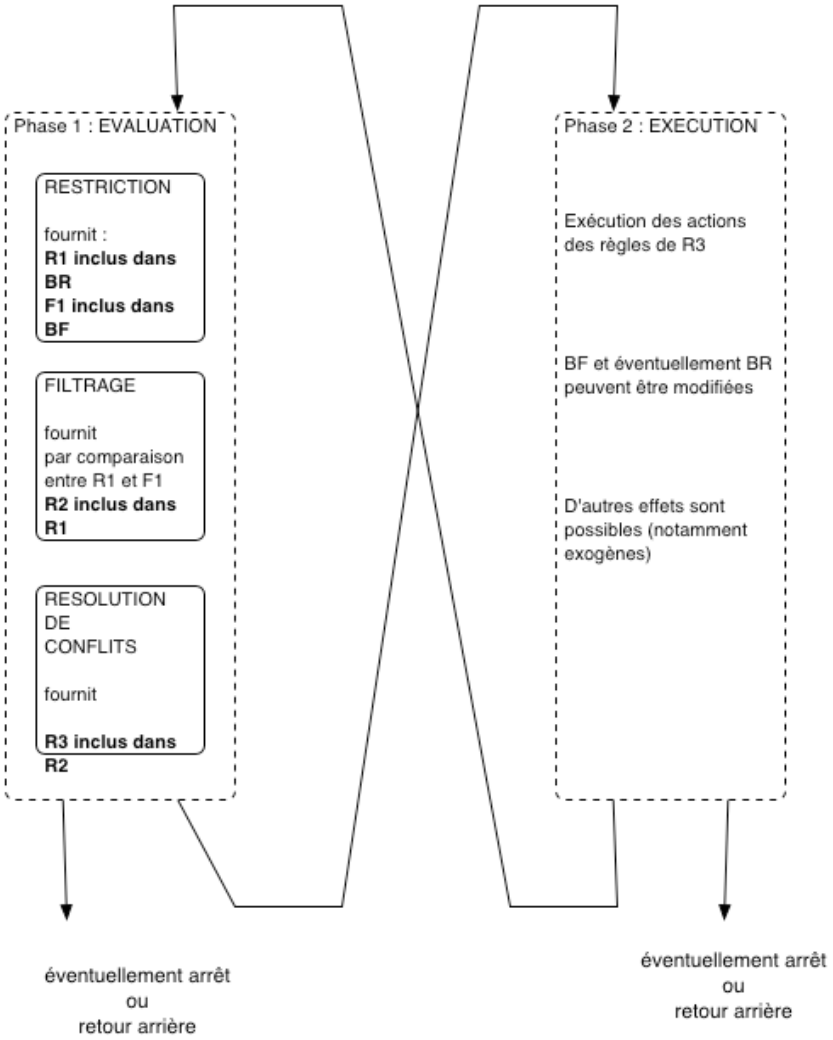
Règles de production et graphe



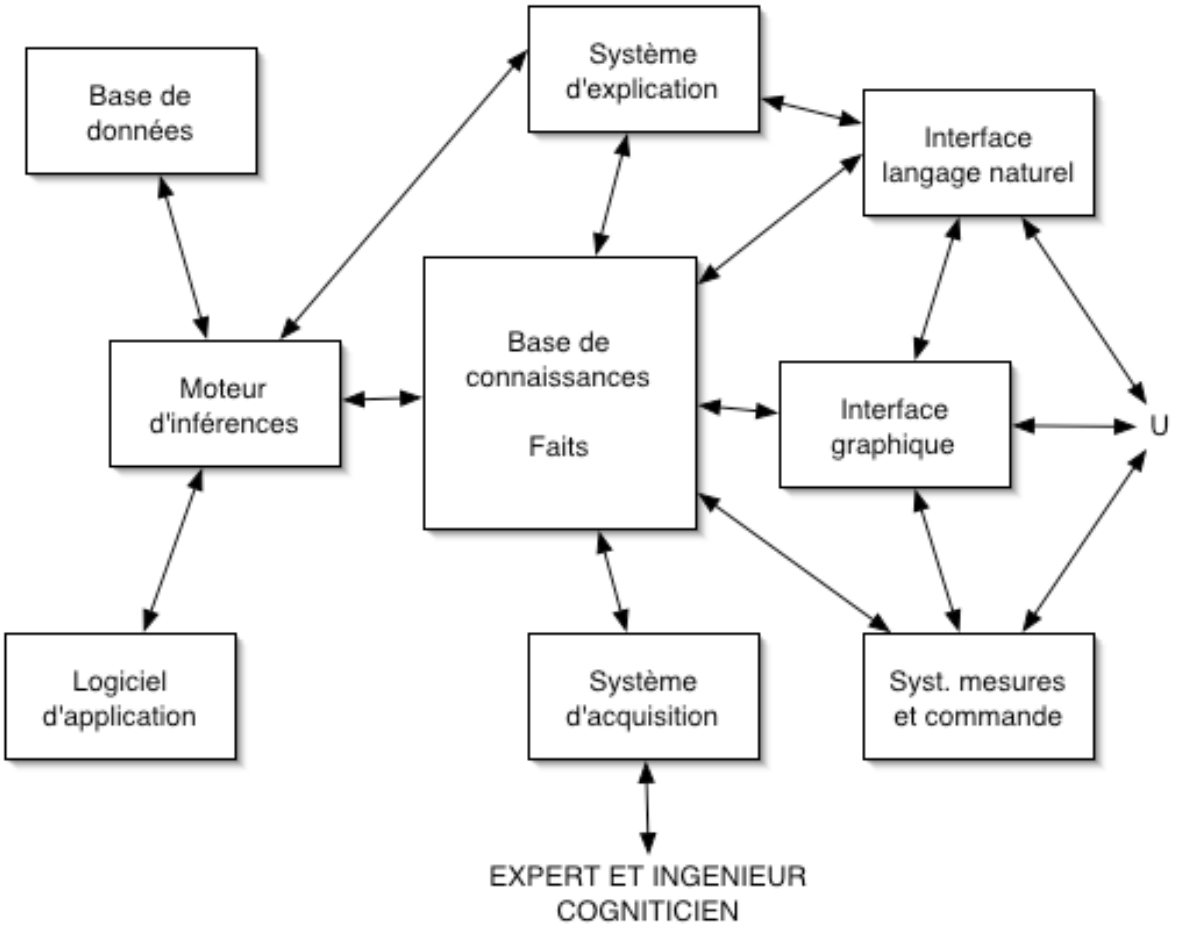
Parcours dans le graphe

- Chaînage avant
  - Le système déclenche des règles jusqu'à épuisement ou arrêt (le déclenchement d'une règle n'est pas remis en cause et les faits établis le demeurent par la suite)
- Chaînage arrière
  - Le système cherche à unifier la partie conclusion des règles avec ce but, et fixe comme nouveaux buts les prémisses de la règle sélectionnée
- Chaînage mixte
  - La partie déclencheur des règles contient à la fois un but et des prémisses

Moteur d’inférences



Architecture Système Expert



Applications S.E.

- Classification, diagnostic, interprétation de données
  - Médical (MYCIN (1975), PICON)
  - Chimie : DENDRAL
  - Géologie : PROSPECTOR
- Planification, ordonnancement, suivi de tâches
  - NASA (sonde spatiale)
- Allocation de ressources
  - Golf 1992
- Conception
  - OrCAD, ….
- Jeux
  - Deep Blue: 30 RS/6000 + 480 VLSI + 30 milliards positions/mvt.
  - Jeux de cartes, GO, …

Réseaux de neurones et apprentissage

Sorin Moga
Sorin.Moga@enst-bretagne.fr

LUSSI / ENST Bretagne

Introduction

Quand le seul outil dont tu disposes est un marteau, chaque problème que tu rencontreras aura tendance à ressembler à un clou.

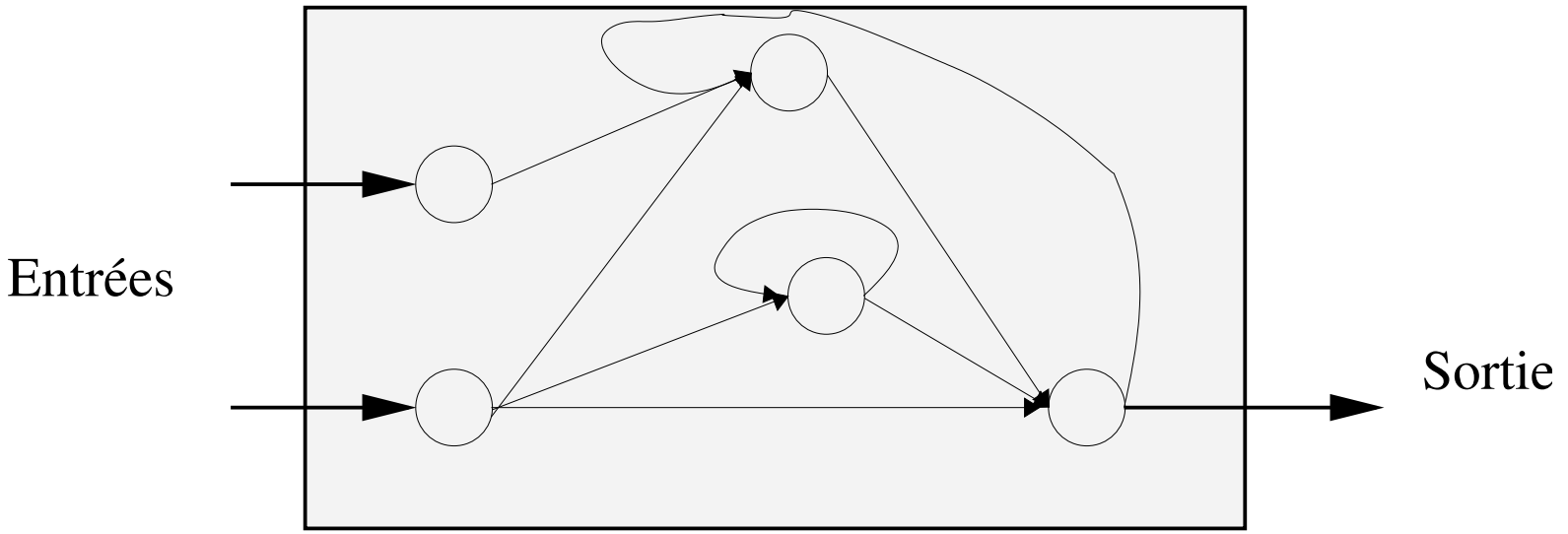
"Biologie de la conscience" – G. M. Edelman

Les ordinateurs sont adaptés a exécuter des tâches ... séquentielles.

Réseaux de neurones

- Ensemble de méthodes d'analyse et de traitement des données
  Sont une mèthode parmi d'autres !!!!
- Deux caractéristiques majeures :
  - combinaison (réseau) d'éléments simples (neurones)
  - non linéaire (par "opposition" aux méthodes classiques de l'analyse des données)
  - Vaguement issus de considérations biologiques
  - Essentiellement numériques, par opposition à l'IA symbolique (base de règles, raisonnement par cas, etc.)

Représentation graphique



Boite (très) noire ...

Différents modèles

Les réseaux de neurones diffèrent selon :

- le type des neurones utilisés
- la structure du réseau
- le mode d'apprentissage

Plan du cours

- Rappels
- Les types de neurones (biologique, formel)
- Les types d'architectures
- Les types d'apprentissages
- Conclusions

Applications : analyse des données

Organisation et exploration des données :

- discrimination : affectation de nouvelles données à des groupes connus (exemple : diagnostic médical)
- "régression" : modélisation de relations fonctionnelles (exemple : niveau d'ozone demain en fonction du niveau d'aujourd'hui et de mesures météo : la vitesse du vent, ...)
- classification : découverte de groupes dans des données (exemple : regroupement de profils de consommateurs)

Nature des données

Dans la pratique, les données posent des problèmes :

- données fausses (erreur de mesure, etc...)
- données incomplètes ou manquantes (absence de réponse, grosse erreur de mesure qui conduit à rejeter le résultat, etc.)
- données non exhaustives : presque toujours le cas

Les résultats sont donc "imprécis" :

- algorithmes exacts sur données réelles : le résultat est juste par rapport aux données, mais pas nécessairement par rapport à la réalité
- algorithmes heuristiques : on ne sait pas si l'algorithme donne le "bon" résultat, mais simplement qu'il est satisfaisant...

Modélisation statistique

Deux gros problèmes pratiques (fortement couplés) :

- mesure des performances :
  - bons résultats sur les données de départ, mais mauvais sur de nouvelles données
  - marges d'erreur, niveau de confiance
- choix du modèle :
  - dilemme biais/variance
  - modèle puissant : faible biais, mais très sensible aux données
  - modèle faible : grand biais, mais moins sensible aux données

Cadre mathématique général

Les données à étudier sont décrites de la façon suivante :

- on dispose de N individus ou exemples
- chaque individu est décrit par n variables réelles, i.e. chaque exemple est un vecteur $x \in \Re^n$
- dans le cas de la discrimination, chaque exemple est associé à une classe (un groupe)
- dans le cas de la régression, chaque exemple est associé à une variable cible, notée y (élément de $\Re^p$).

On peut reformuler les trois problèmes de l'AD :

- discrimination : trouver un lien entre x et sa classe
- régression : trouver un lien entre x et y
- classification : construire des classes en associant des étiquettes aux individus

Discrimination et statistique

En discrimination :

- pour chaque individu x_k, on connaît la classe C_j telle que $x_k \in C_j$
- on cherche à classer de nouveaux individus

On doit estimer :

$$P\left(C_j|x\right)$$

la probabilité que l'individu observé soit issu de la classe Cj sachant qu'il est décrit par le vecteur x.

Intérêts :

- permet d'affecter un individu à une classe (la plus probable, en général)
- permet de mesurer l'erreur liée à cette affectation

Régression et statistique

En régression :

- chaque individu x_k est associé à une grandeur $y_k \in \Re^p$
- on cherche à exprimer y_k comme une fonction de x_k

On doit estimer :

$$E\left(y|x\right)$$

l'espérance conditionnelle de y sachant x : c'est la meilleure approximation de y par une fonction de x au sens des moindres carrés.

Classification et statistique

En classification :

- les individus ne sont associés à aucune donnée explicative
- on chercher à trouver des groupes (des classes)

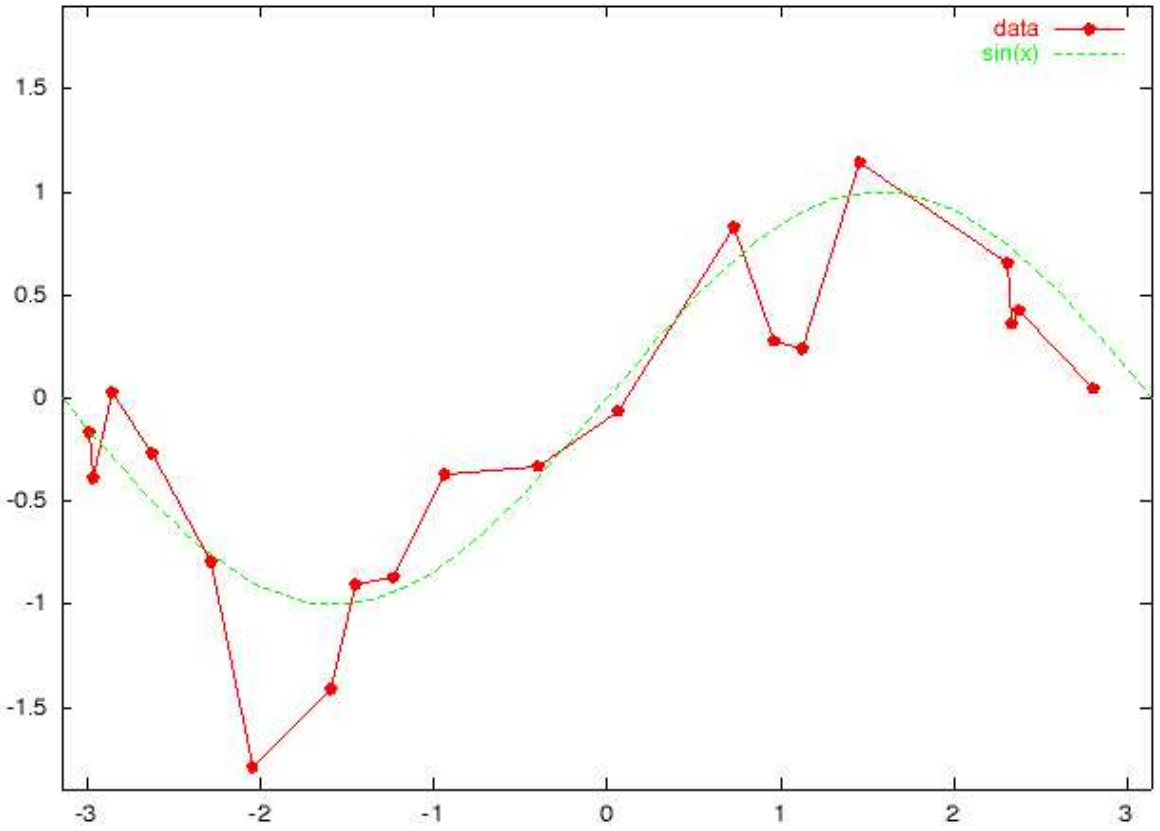
On peut chercher à exprimer $p\left(x\right)$ comme un mélange :

$$p\left(x\right) = \sum_{j=1}^{N} p\left(x|C_j\right) \cdot P\left(C_j\right)$$

On peut ensuite affecter les individus aux classes, grâce à la règle de Bayes :

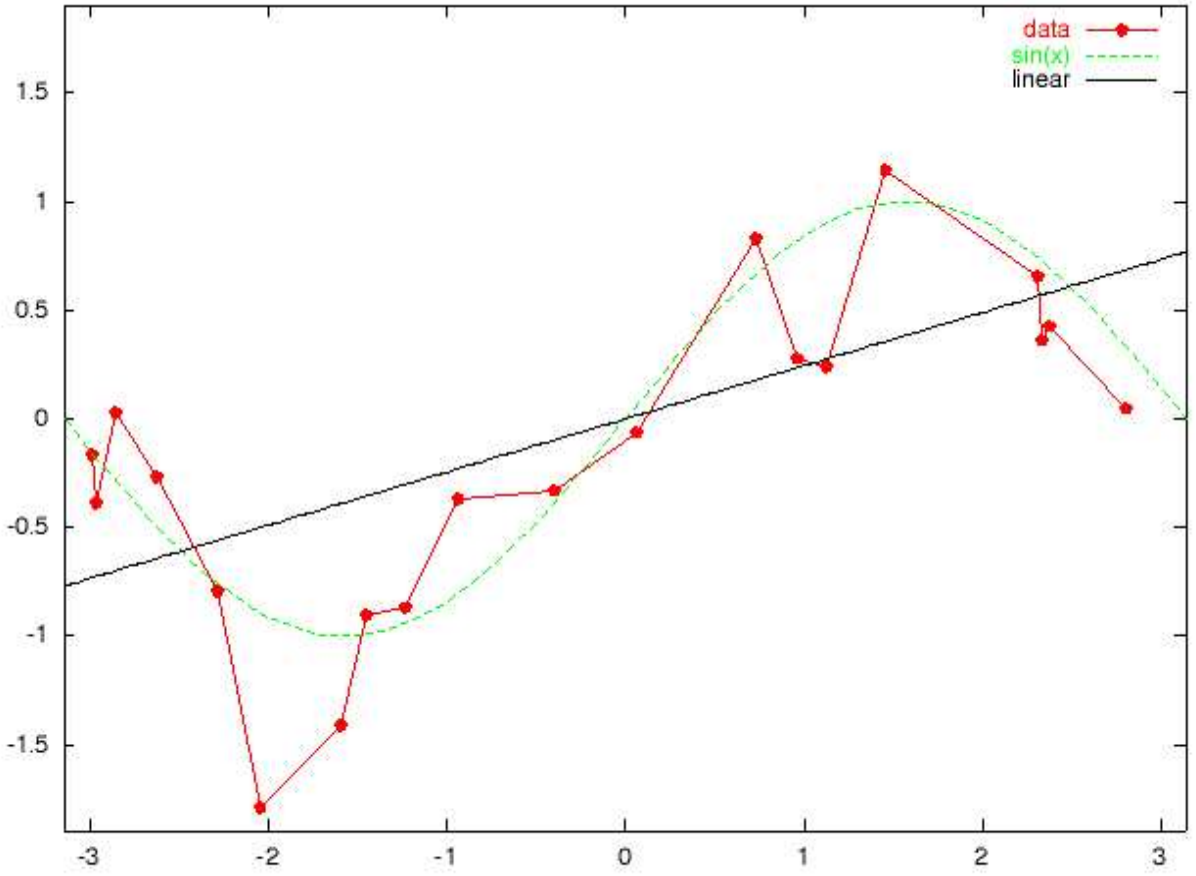
$$p\left(x|C_j\right) = \frac{P\left(C_j\right) P\left(C_j|x\right)}{p(x)}$$

Exemple: régression simple



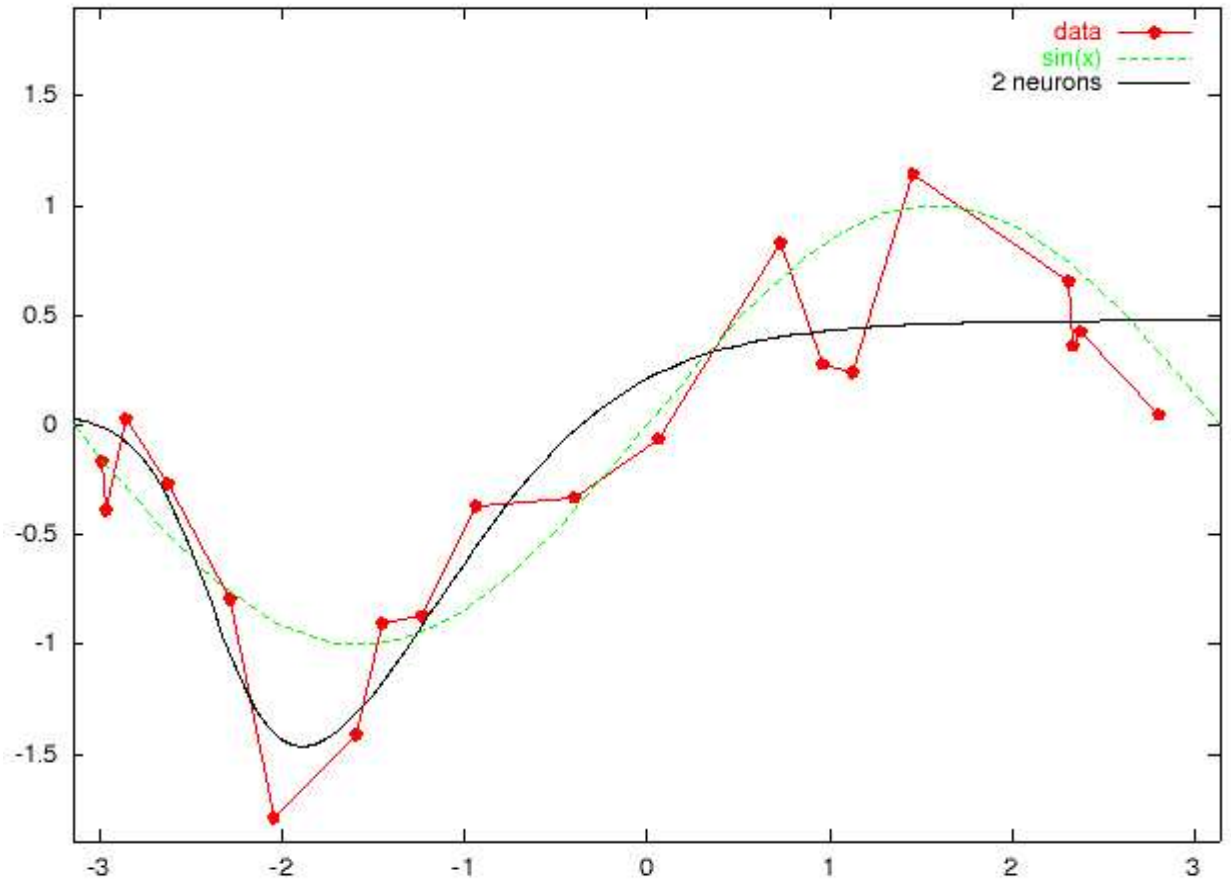
$$y = f(x) = \sin x + \epsilon(x)$$

Exemple: régression simple



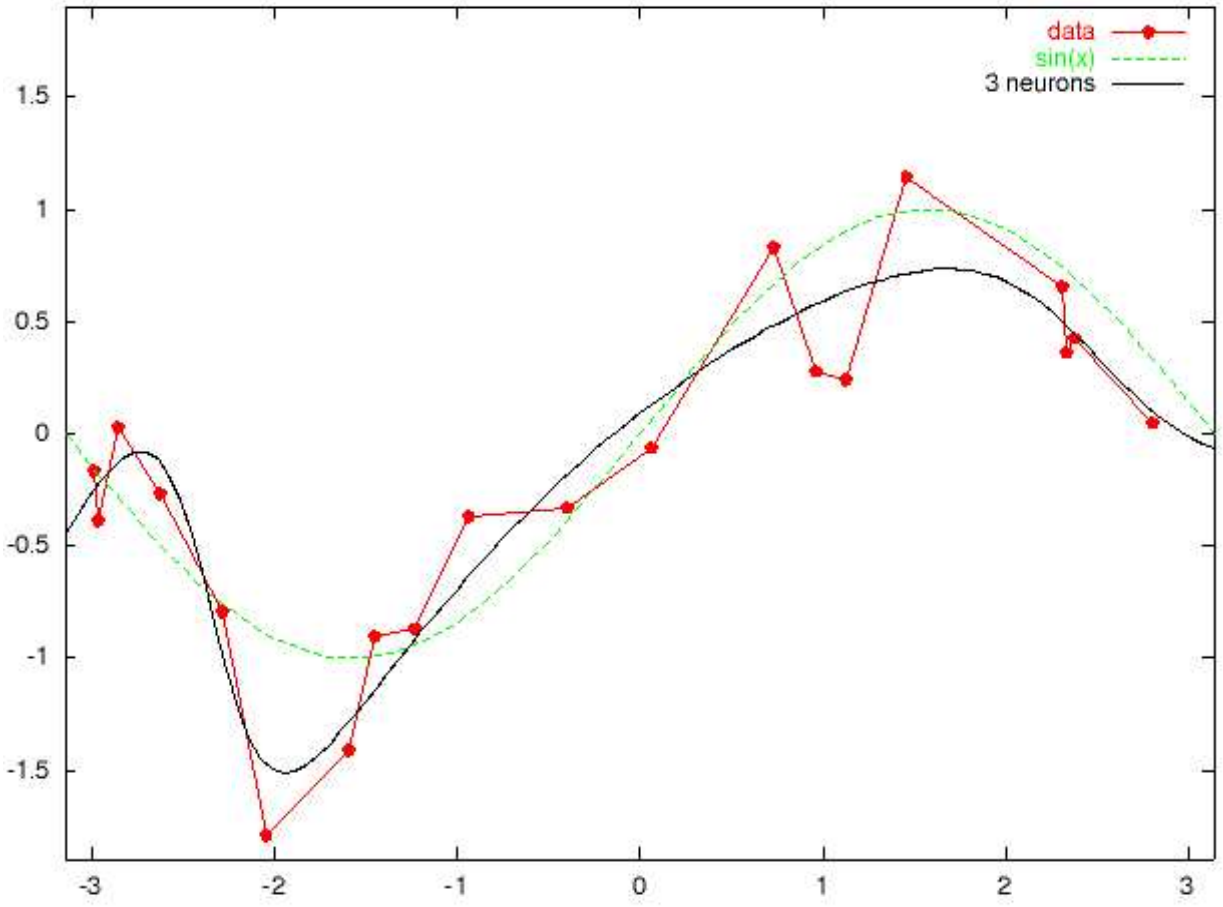
Modèle linéaire

Exemple: régression simple



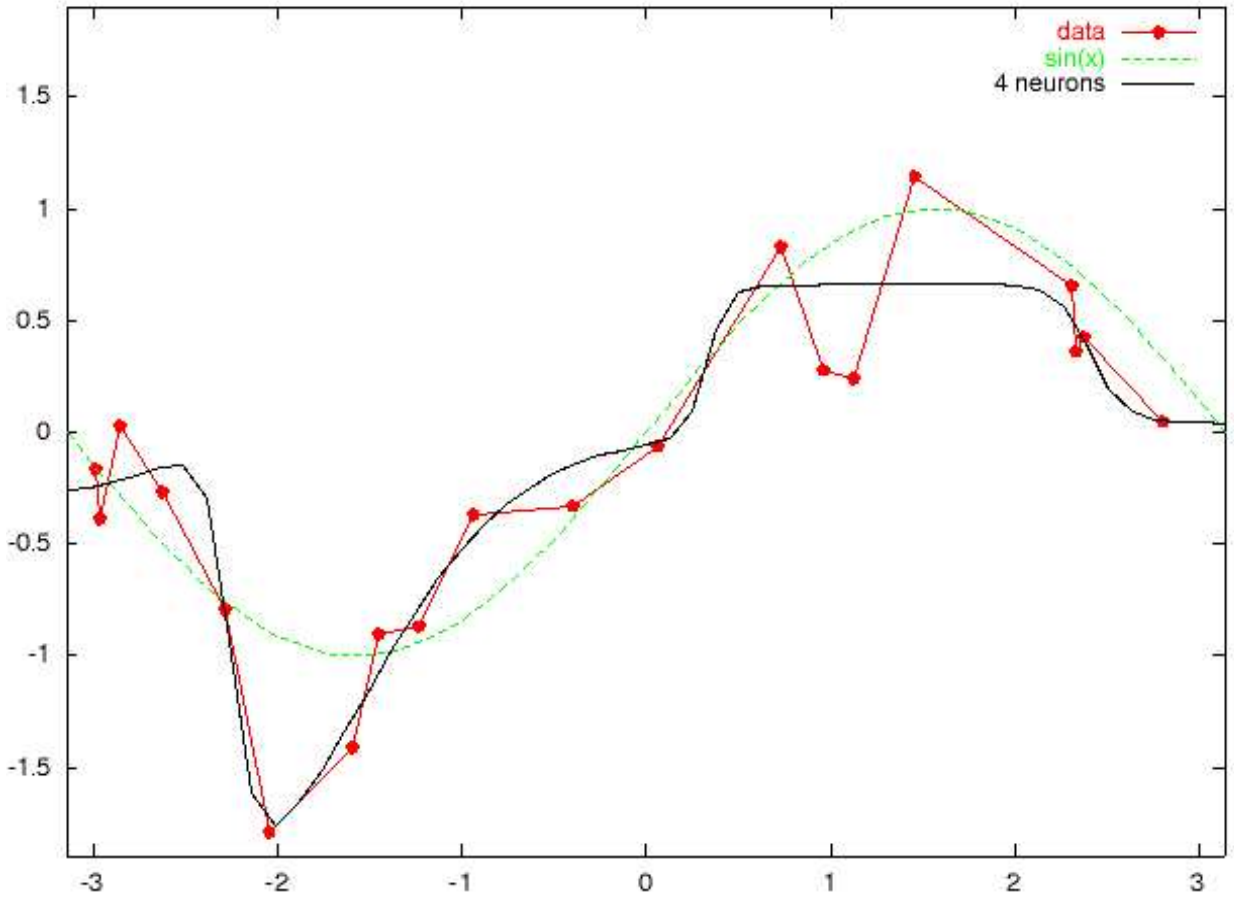
Réseau à 2 neurones cachés

Exemple: régression simple



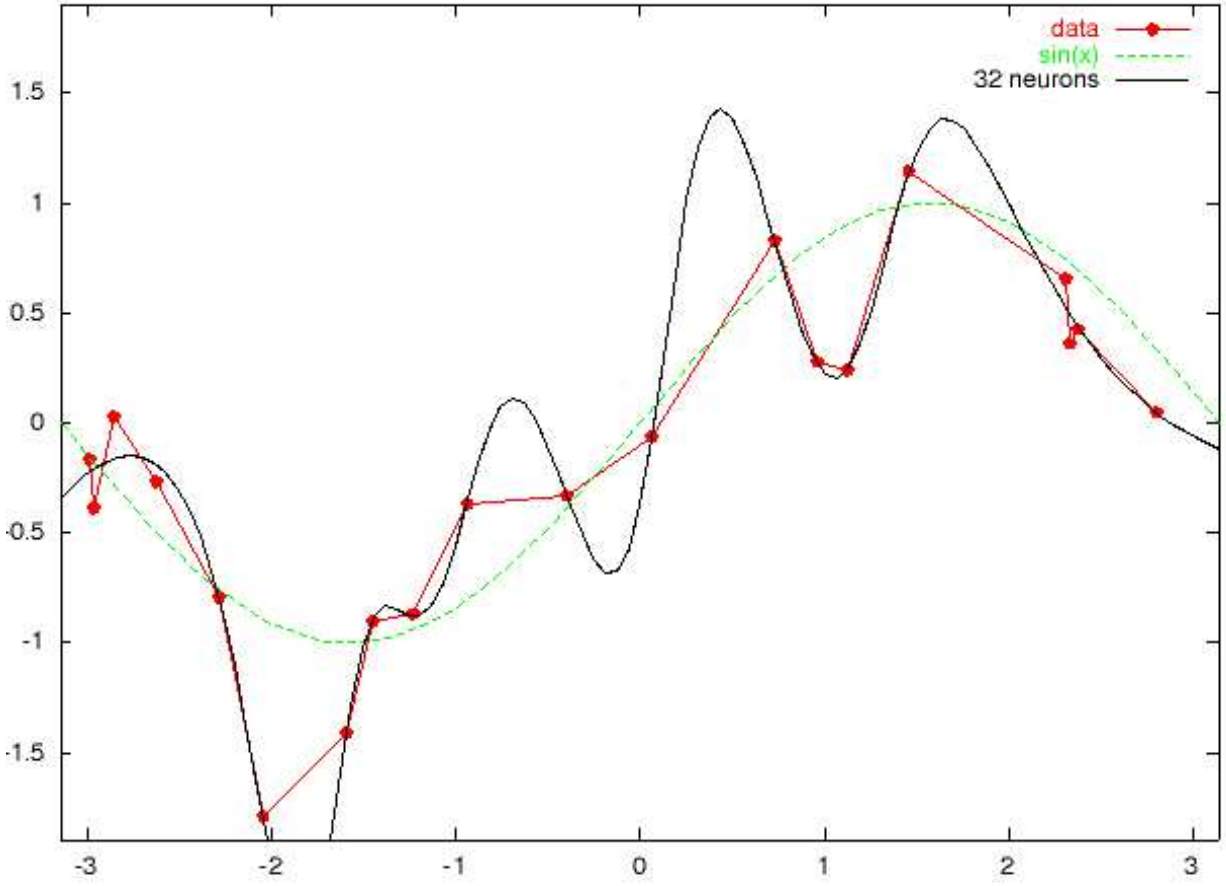
Réseau à 3 neurones cachés

Exemple: régression simple



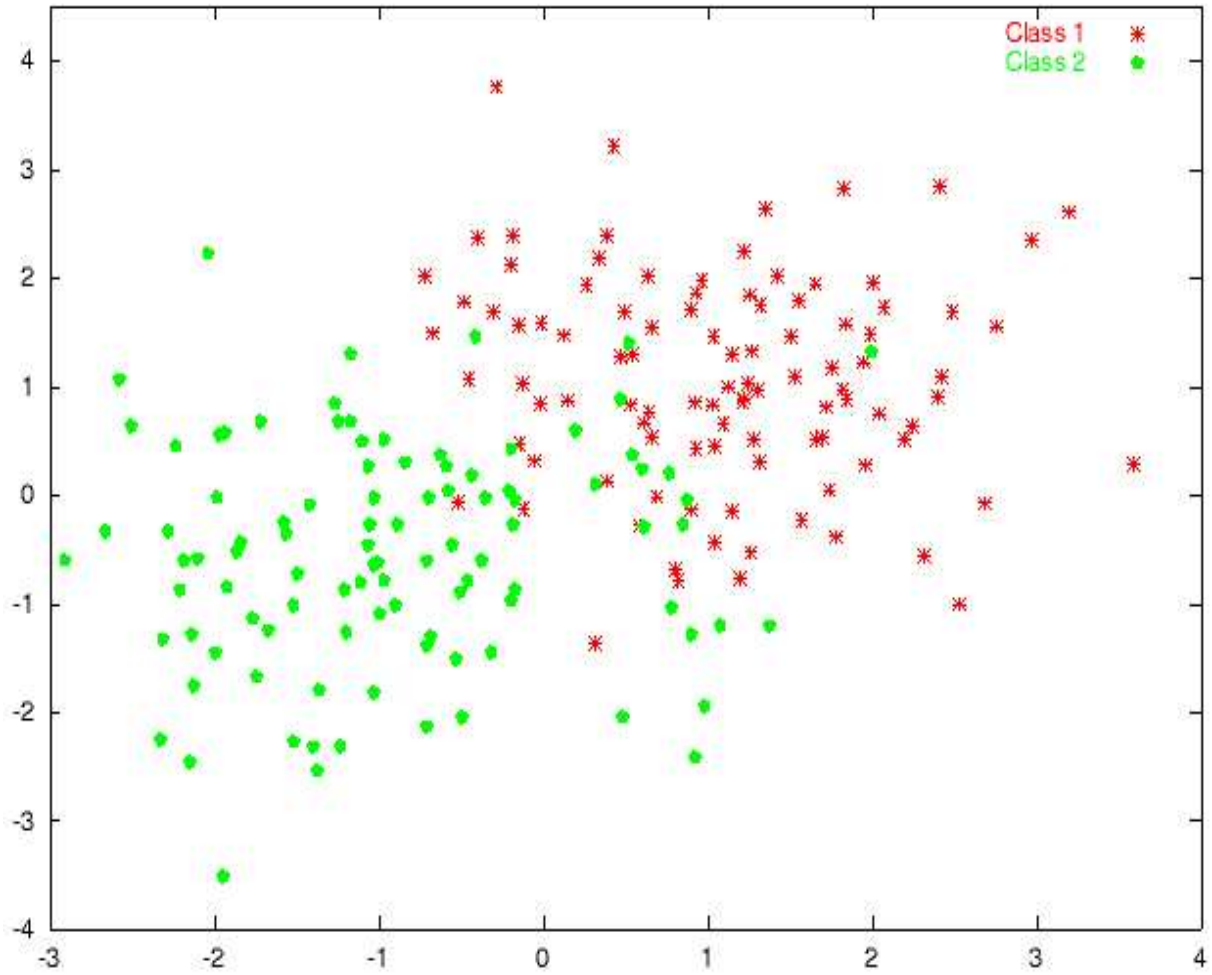
Réseau à 4 neurones cachés

Exemple: régression simple

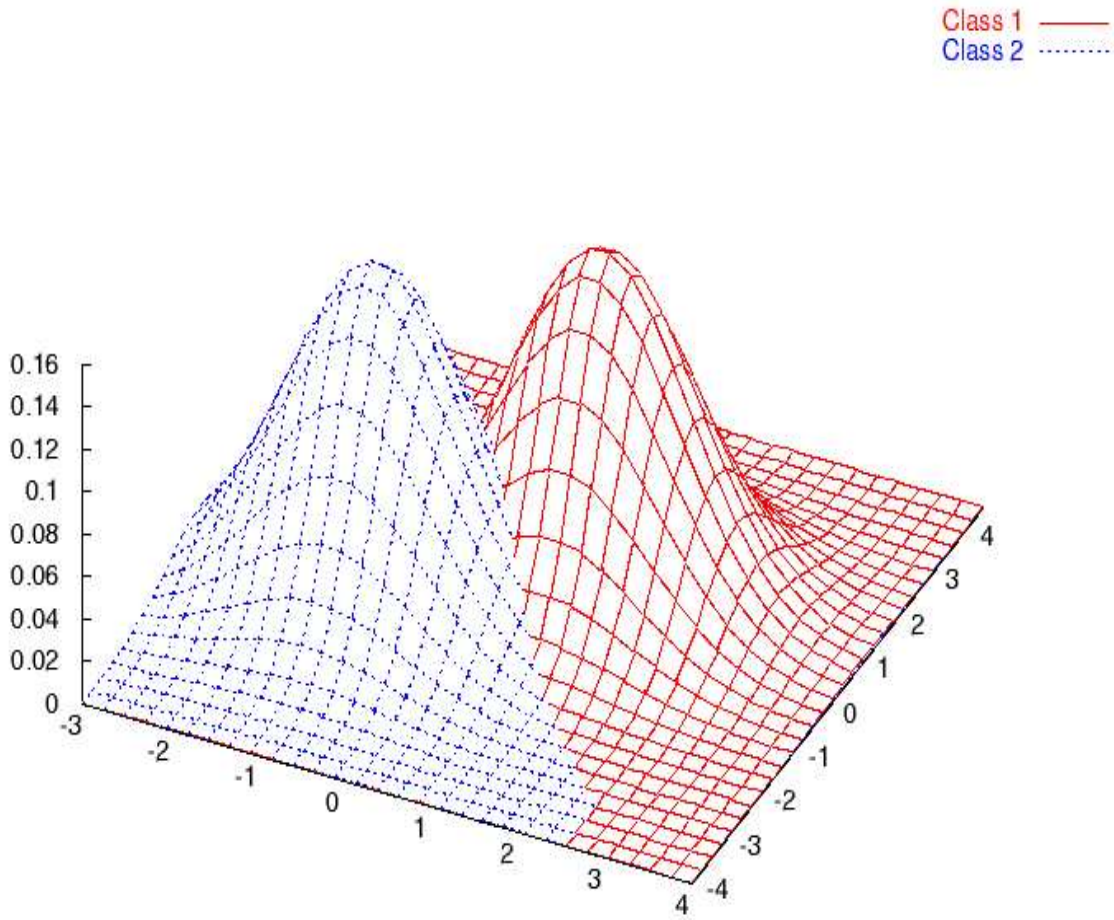


Réseau à 32 (!) neurones cachés

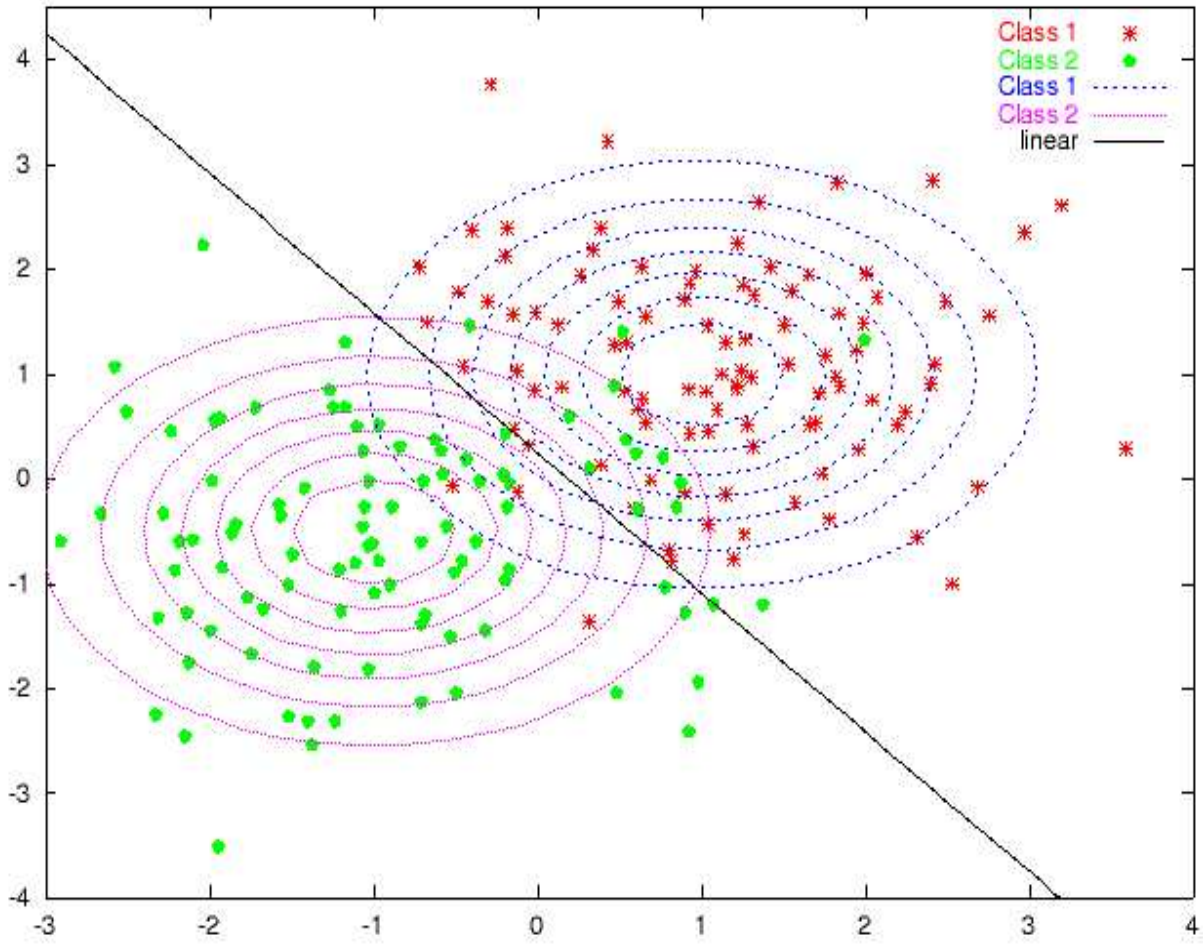
Exemple: dicrimination



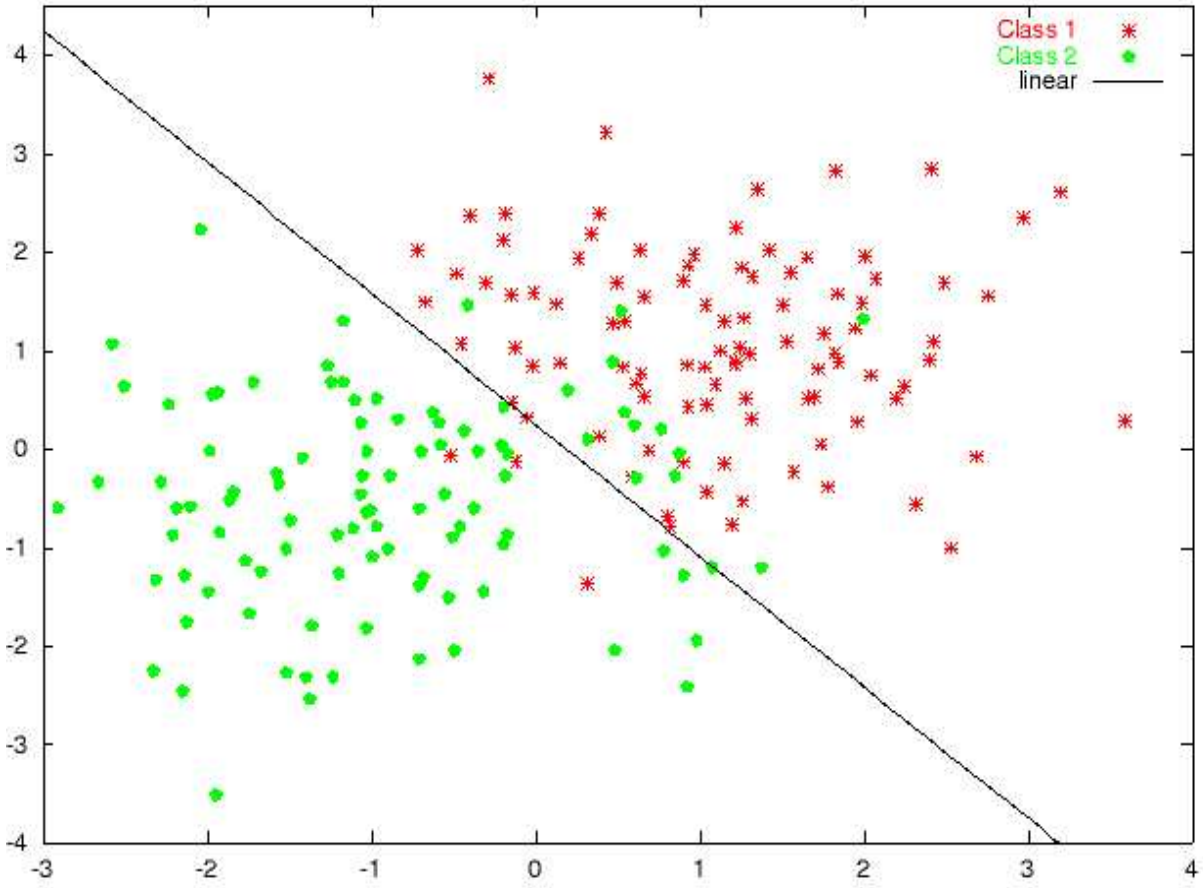
Exemple: dicrimination



Exemple: dicrimination

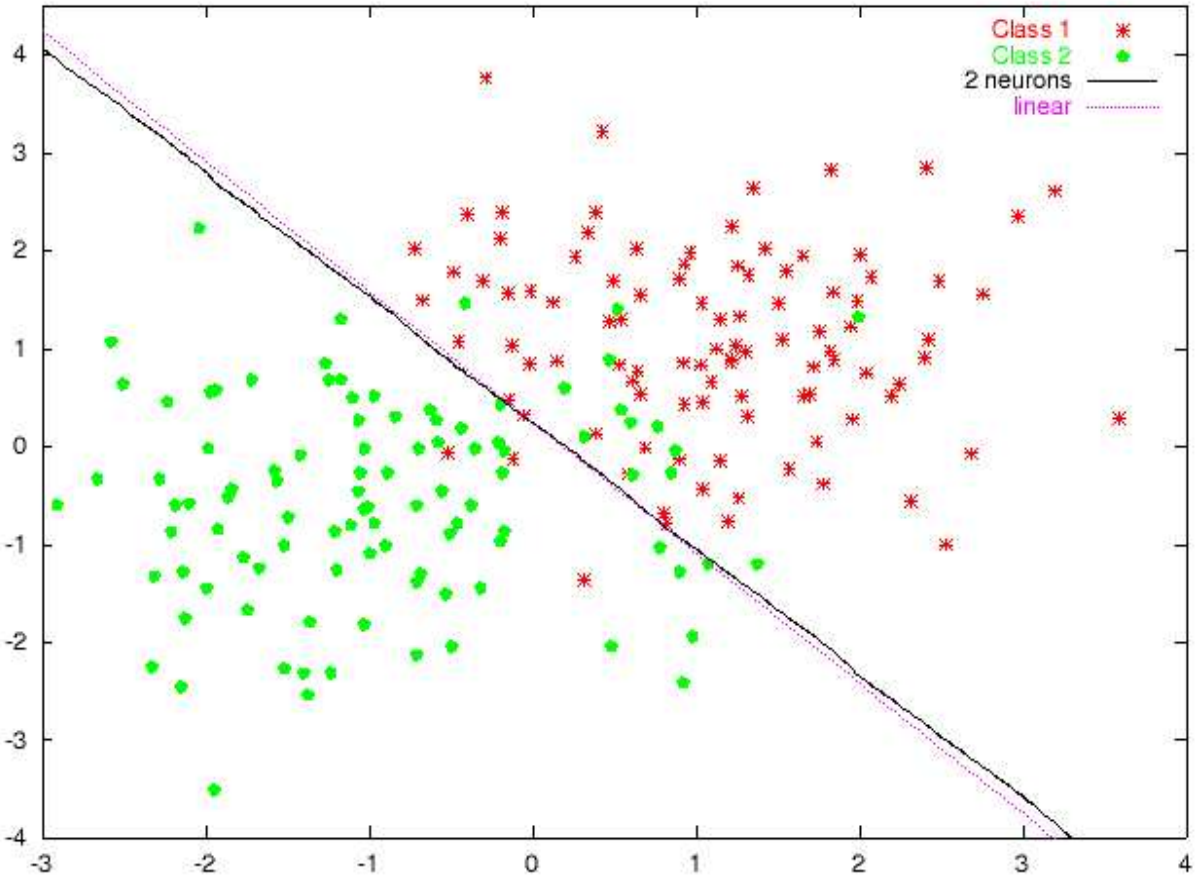


Exemple: dicrimination



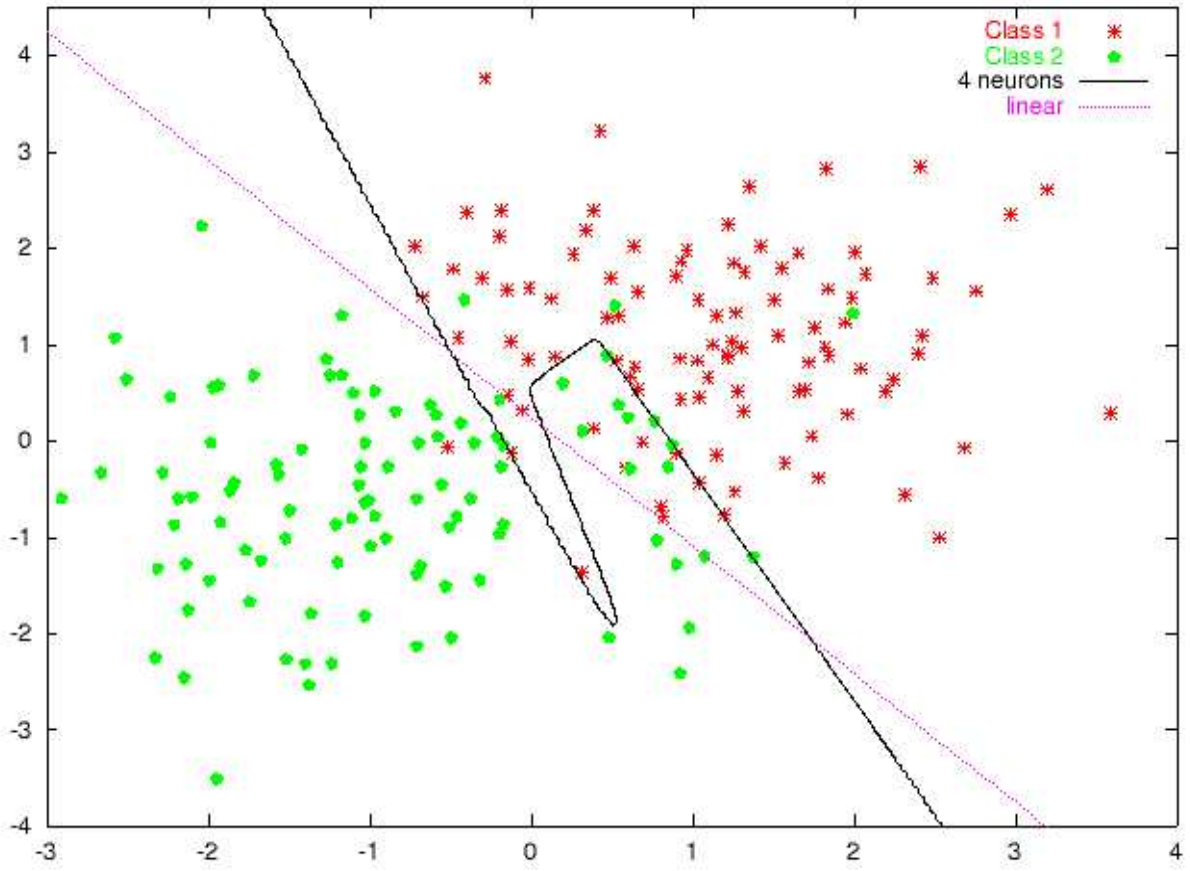
Modèle linéaire (optimal), 16 erreurs

Exemple: dicrimination



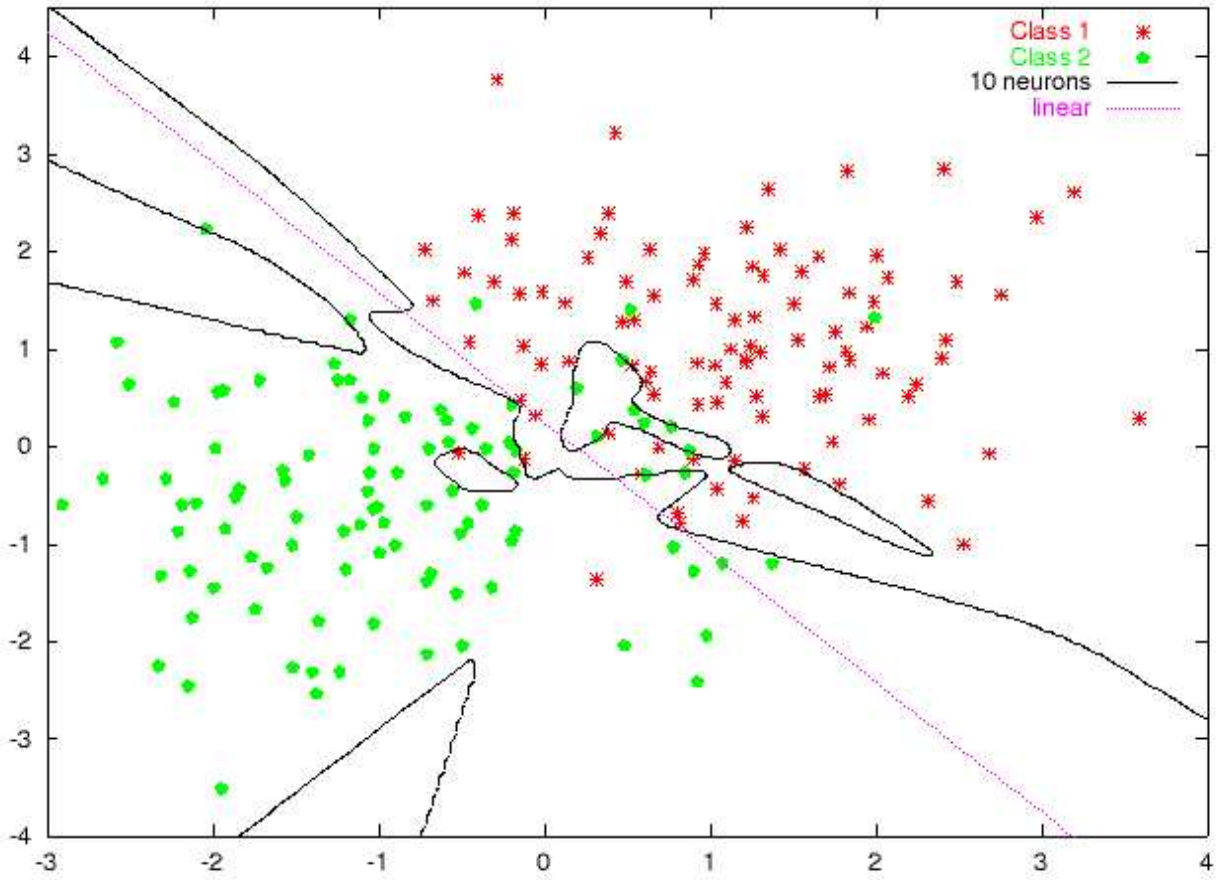
Réseau à 2 neurones cachés, 16 erreurs

Exemple: dicrimination



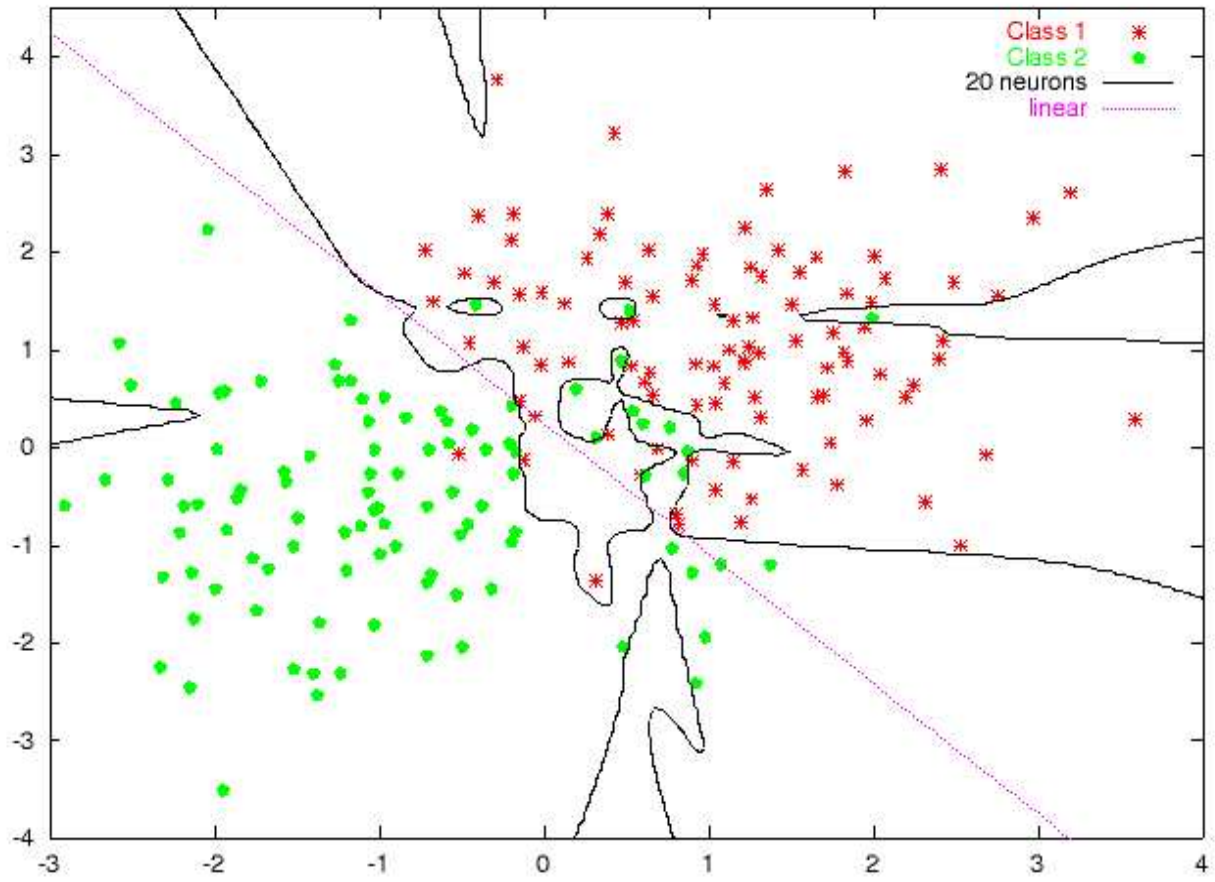
Réseau à 4 neurones cachés, 12 erreurs

Exemple: dicrimination



Réseau à 10 neurones cachés, 4 erreurs

Exemple: dicrimination

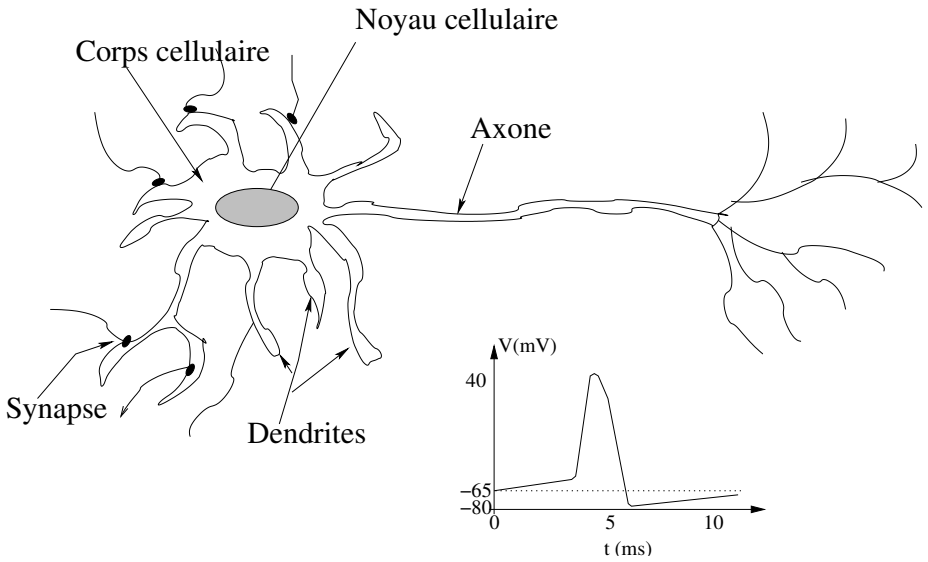


Réseau à 20 (!) neurones cachés, 1 erreur

Histoire

- 1940 : La machine de Turing
- 1943 : Le neurone formel (McCulloch & Pitts)
- 1948 : Les réseaux d'automates (Von Neuman)
- 1949 : Première règle d'apprentissage (Hebb)
- 1958-62 : Le Perceptron (Rosenblatt)
- 1960 : L'Adaline (Widrow & Hoff)
- 1969 : Perceptrons (Minsky & Papert)
  - les limites du Perceptron, besoin d'architectures + complexes
- 1986 : Rétropropagation (Rumelhart & McClelland)
  - nouvelles architectures de Réseaux de Neurones
  - applications : reconnaissance de l'écriture/parole, vision, ...

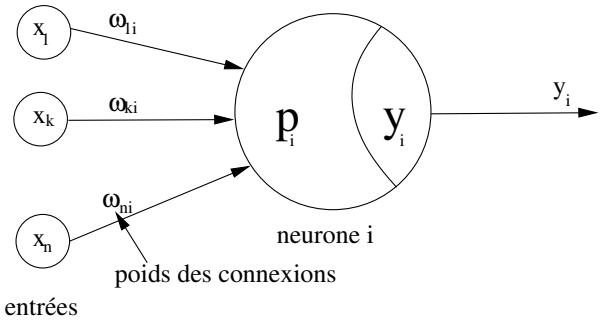
Le neurone biologique



- 10 à 100 milliards de neurones chacun avec 10000 synapses (50 types de neurotransmetteurs)
- L'activité = $f\,(frequence\ du\ train\ de\ potentiels\ d'action)$

Le neurone formel

- Le neurone artificiel = modélisation très simplifiée du neurone biologique (McCulloch & Pitts - 1943)



- Potentiel

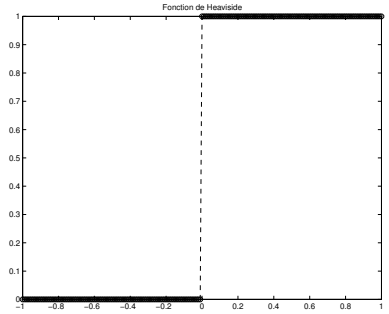
$$p_i = \sum_{k=1}^{N} x_k \cdot w_{ki} + \theta = \sum_{k=0}^{N} x_k \cdot w_{ki} = \vec{\mathbf{X}} \cdot \vec{\mathbf{W}}^T$$

- Sortie

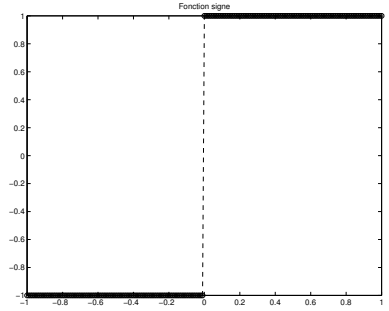
$$y_i = f_{act}\left(p_i\right) = f_{act}\left(\sum_{k=0}^{N} x_k \cdot w_{ki}\right)$$

La fonction d'activation f_{act}

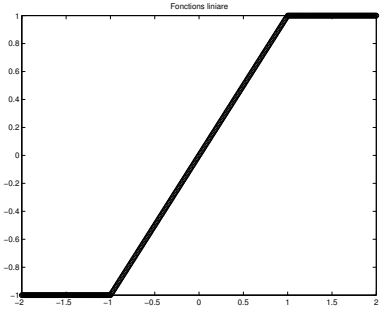
- Heaviside : $f_{act}(x) = \begin{cases} 1, & si\ x \geq 0 \\ 0, & si\ x < 0 \end{cases}$



- Signe : $f_{act}(x) = \begin{cases} 1, & si\ x \geq 0 \\ -1, & si\ x < 0 \end{cases}$

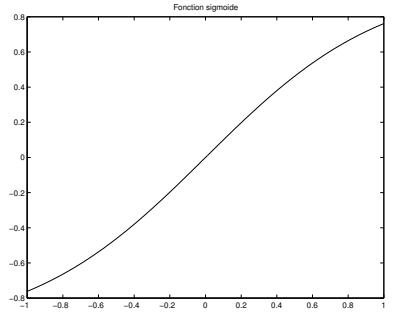


- linéaire à seuil :$f_{act}(x) = \begin{cases} 1 & si\ x > a \\ \frac{1}{a}x & si\ x \in [-a, a] \\ -1 & si\ x < a \end{cases}$

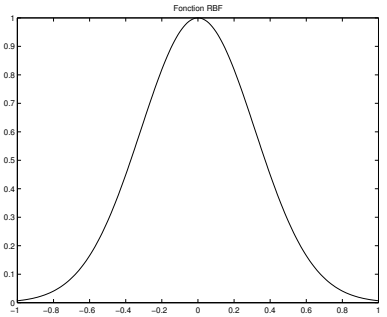


La fonction d'activation f_{act}

- sigmoïde : $f_{act}(x) = \tanh(kx)$ ou $\frac{1}{1+e^{-kx}}$



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

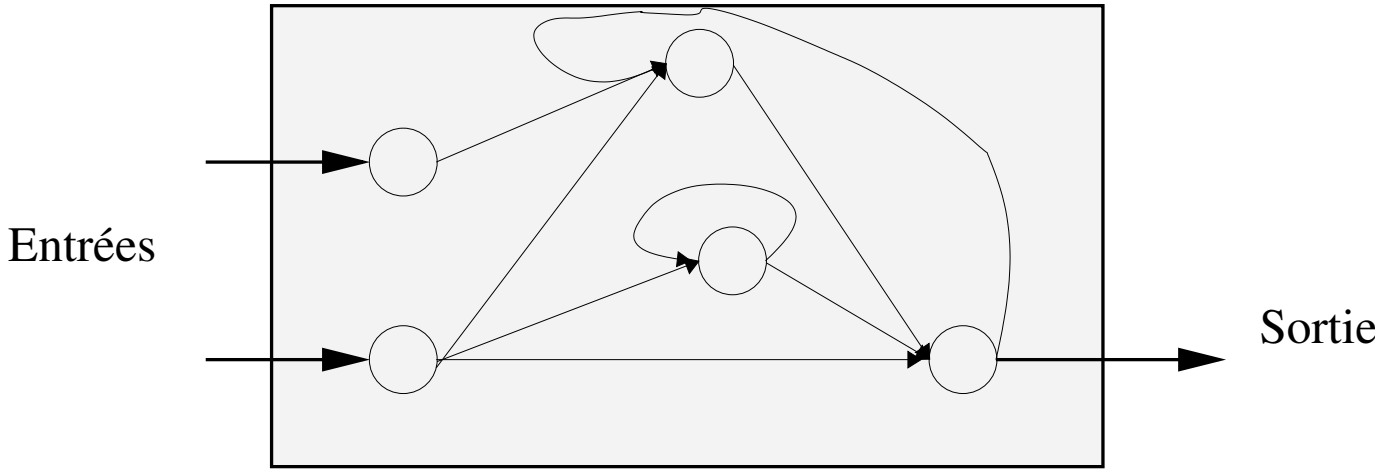


- stochastique :

$$f_{act}(x) = \begin{cases} 1 \;\; avec\ la\ probabilite\ P(x) = \frac{1}{1+e^{-\frac{x}{t}}} \\ 0 \;\; sinon \\ avec\ x = \prod_{i=1}^{n} \omega_i \cdot x_i \end{cases}$$

Réseaux de neurones

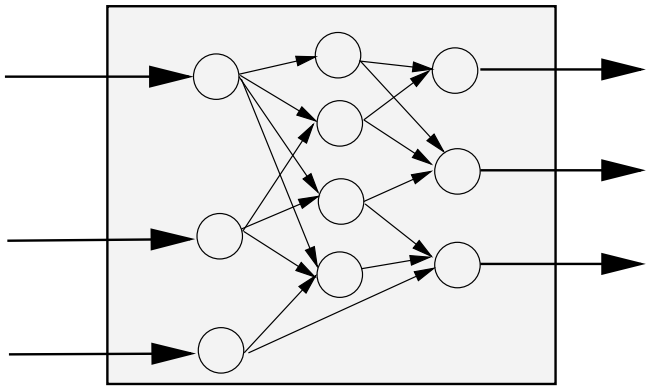
- Principe :
  - graphe orienté : interconnexion d'éléments simples (les neurones)
  - échange d'informations via les connexions
  - calcul distribué et coopératif
- boite noire ...



- Types d'architectures : réseaux feed-forward, réseaux récurrents, cartes topologiques

Réseaux feed-forward

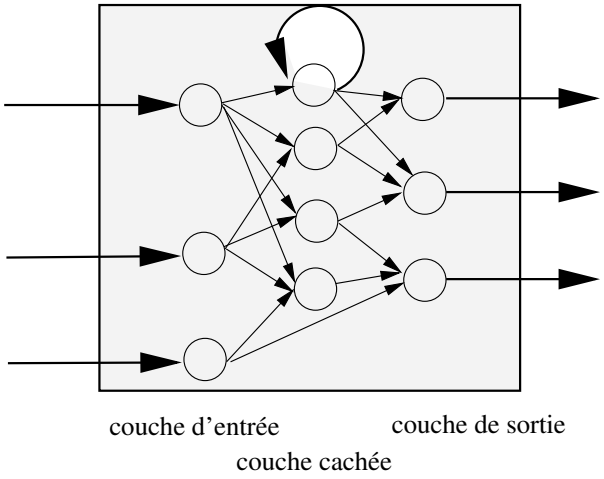
- Organisation des neurones en couches successives

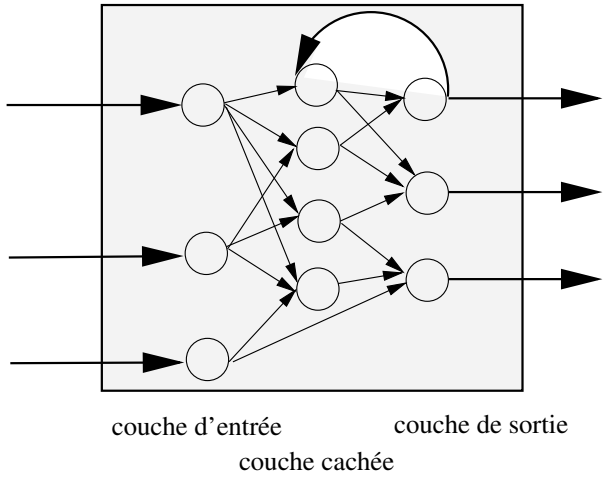


- Le calcul des sorties se fait en propageant les calculs de gauche à droite
- + connexions directes linéaires : $y = a \cdot x + FW(x)$

Réseaux récurrents

- Présence d'au moins une boucle de rétroaction au niveau des neurones ou des couches

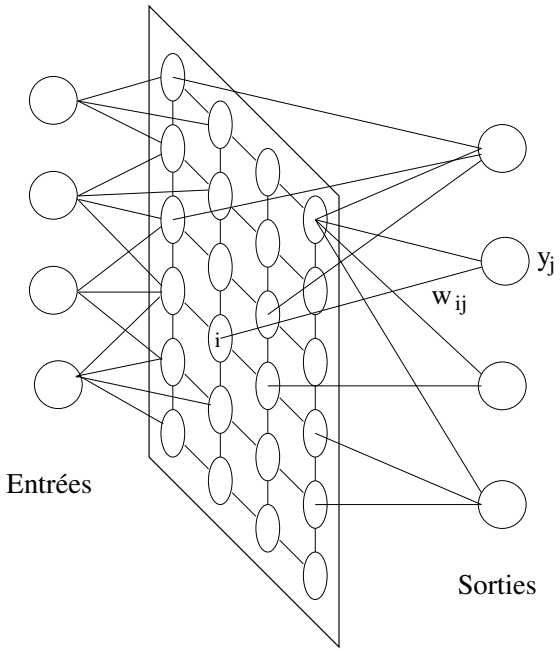


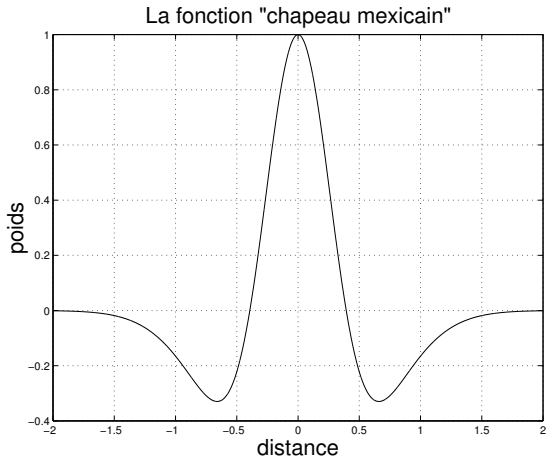


- Prise en compte d'aspects temporels et de mémoire
- Modèles plus difficile à mettre en oeuvre (convergence, stabilité)

Réseaux topologiques

- Les neurones des couches cachées sont totalement interconnectés et les interactions sont “modulés” par la topologie





- Peu ou pas d’apprentissage des interactions latérales
- Problèmes de convergence, temps de calcul, ... mais biologiquement plausibles

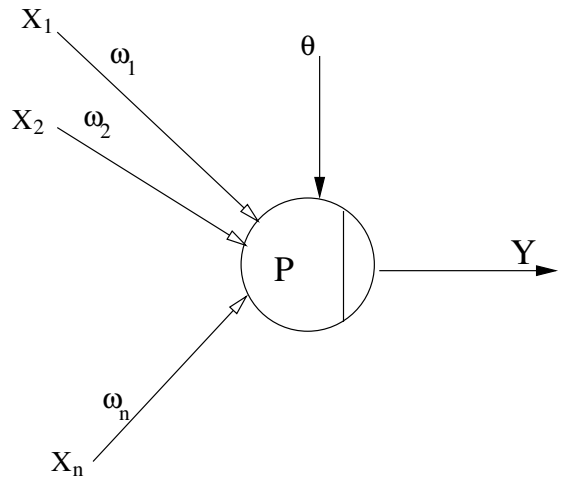
Différents types d'apprentissage

- Apprentissage supervisé :
  - On fournit aux réseaux une série d'exemples (x) et de résultats (y)
  - Il faut trouver les poids (W) tel que $y = F_W(x)$ (+ bonne généralisation)
- Apprentissage par renforcement (semi-supervisé) :
  - On fournit des exemples et des indications sur le résultat (vrai—faux)
- Apprentissage non supervisé :
  - On fournit seulement les exemples (x)
  - trouver les poids (W) tel que les x soient correctement groupés selon F_W (+ bonne généralisation)

Réseaux de neurones “usuels”

- Réseaux feedforward + apprentissage supervisé :
  - Multilayer Perceptron (MLP) / Perceptron Multi-Couches (PMC)
  - Radial Basis Function network (RBF) / Réseaux à fonctions de base radiale (RBR)
- Réseaux récurrents + apprentissage supervisé :
  - Time Delay Neural Networks (TDNN), NARMAX
- Apprentissage non supervisé
  - Winner Take All (WTA)
  - Adaptative Resonance Theory (ART)
  - Self-Organizing Maps (SOM) / Cartes auto-organisatrices

Le Perceptron/ADALINE



- Entrées binaires : $x_i \in [-1, 1]$
- Sortie binaire : $Y \in [-1, 1]$
- Neurone formel : $P = \sum_{i=1}^{N} w_i \cdot x_i - \theta$
- Fonction d'activation : $Y = signe(P)$
- Mise à jour : $\Delta w_i(t) = \alpha \left(D(t) - Y(t)\right) \cdot x_i(t)$

Exemple

Question : Sachant que les poids du Perceptron à deux entrées sont les suivants : $w_1 = 0.5$, $w_2 = 0.2$ et que la valeur de seuil est $\theta = 0.4$, déterminez son comportement, sachant que les comportements du ET logique, OU logique et OU exclusif sont rappelés ci-dessus :

		ET			OU			OU Exc
x_1	x_2	Y	x_1	x_2	Y	x_1	x_2	Y
1	1	1	1	1	1	1	1	1
1	-1	-1	1	-1	1	1	-1	-1
-1	1	-1	-1	1	1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	1

La règle d'apprentissage du Perceptron

1. Initialisation des poids
2. Présentation d'une entrée $X(t) = (x_1, ..., x_n)$
3. Calcul de la sortie obtenue S pour cette entrée :

$$P = \sum_{i=1}^{n} w_i \cdot x_i - \theta = \sum_{i=0}^{n} w_i \cdot x_i, \quad Y = signe\,(P) = \begin{cases} +1, \text{ si } P > 0, \\ -1, \text{ si } P \leq 0. \end{cases}$$

4. Si la sortie Y du Perceptron est différente de la sortie désirée $D(t)$ pour cet exemple d'entrée $X(t)$, alors modification des poids

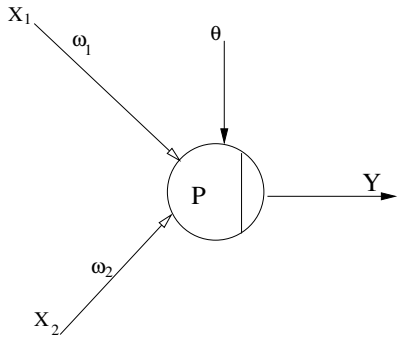
   $w_i(t+1) = w_i(t) + \alpha \cdot ((D(t) - Y) \cdot x_i)$, α le pas de modification

   Rappel : $D(t) = +1$ si $X(t)$ est de la classe 1, $D(t) = -1$ si $X(t)$ est de la classe 2 et $(D(t) - x_i)$ est une estimation de l'erreur.

5. Tant que tous les exemples de la base d'apprentissage ne sont pas traités correctement (i.e. modif. de poids), retour à l'étape 2.

Exercice

Trouvez les poids pour un Perceptron avec 2 entrées qui modélise la fonction ET logique.



		ET
x_1	x_2	Y
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

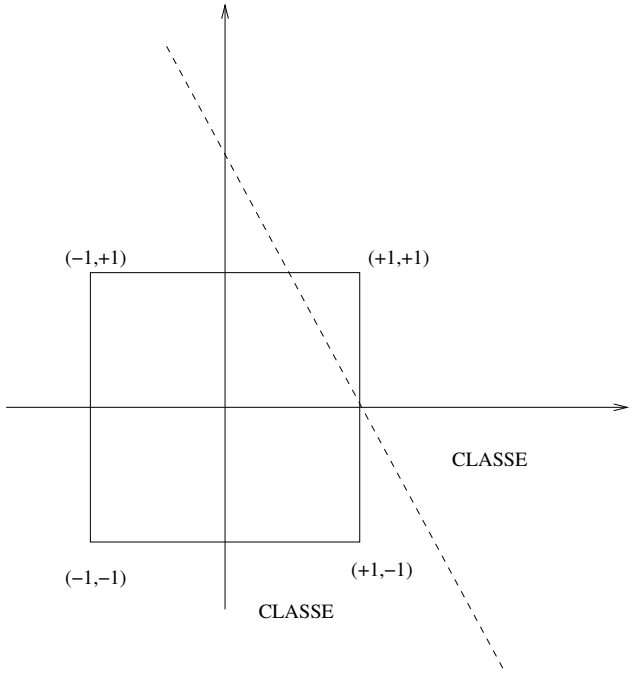
Exercice

- Condition initiales : $w_1 = 0.1$, $w_2 = -0.2$, $\theta = 0.2$ et $\alpha = 0.1$
- $P(0) = 1 \cdot 0.1 + 1 \cdot -0.2 + -1 \cdot 0.2 = -0.3 \rightarrow Y(0) = -1.0 \rightarrow$ modif.
  - $w_1 = 0.1 + 0.1 \cdot (1 - -1) \cdot 1 = 0.3$
  - $w_2 = -0.2 + 0.1 \cdot (1 - -1) \cdot 1 = 0.0$
  - $\theta = 0.2 + 0.1 \cdot (1 - -1) \cdot -1 = 0.0$
- $P(1) = 1 \cdot 0.3 + -1 \cdot 0.0 + -1 \cdot 0.0 = 0.3 \rightarrow Y(1) = 1.0 \rightarrow$ modif.
  - $w_1 = 0.3 + 0.1 \cdot (-1 - 1) \cdot 1 = 0.1$
  - $w_2 = 0.0 + 0.1 \cdot (-1 - 1) \cdot -1 = 0.2$
  - $\theta = 0.0 + 0.1 \cdot (-1 - 1) \cdot -1 = 0.2$
- $P(2) = -1 \cdot 0.1 + 1 \cdot 0.2 + -1 \cdot 0.2 = -0.1 \rightarrow Y(2) = -1.0$
- $P(3) = -1 \cdot 0.1 + -1 \cdot 0.2 + -1 \cdot 0.2 = -0.5 \rightarrow Y(3) = -1.0$
- $P(0) = 1 \cdot 0.1 + 1 \cdot 0.2 + -1 \cdot 0.2 = 0.1 \rightarrow Y(0) = 1.0$
- $P(1) = 1 \cdot 0.1 + -1 \cdot 0.2 + -1 \cdot 0.2 = -0.3 \rightarrow Y(1) = -1.0$

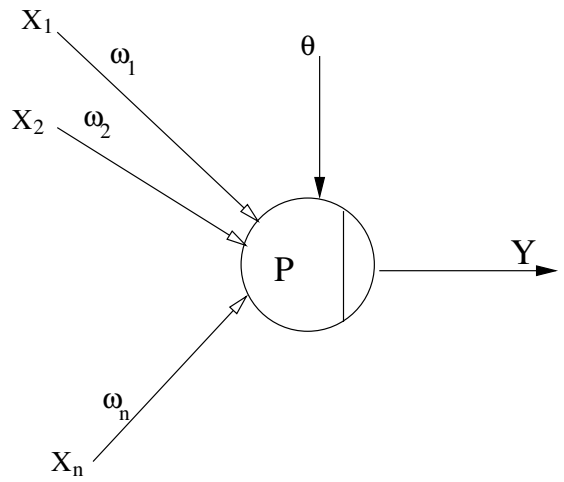
Exercice

Le Perceptron réalise une partition de son espace d'entrée en 2 classes (1 et 2) selon la valeur de sa sortie (+1 ou -1). La séparation de ces deux zones est effectuée par un hyperplan. L'équation de la droite séparatrice est :

$$x_1 \cdot w_1 + x_2 \cdot w_2 + \theta = x_1 \cdot 0.1 + x_2 \cdot 0.2 - 0.2 = 0$$



Le Perceptron / Généralisation



- Entrées réelle : $x_i \in \Re$
- Sortie réelle : $Y \in \Re$
- Neurone formel : $P = \sum_{i=1}^{N} w_i \cdot x_i - \theta$
- Fonction d'activation :
  $Y_i = f_{act}(P_i) = f_{act}\left(\sum_{j=0}^{N} x_j \cdot w_{ji}\right) = f_{act}\left(\vec{\mathbf{X}} \cdot \vec{\mathbf{W}}^T\right)$
- Mise à jour : minimiser l'erreur (moindres carrés)

Minimiser l'erreur (moindres carrés)

$$J = \sum_{i=0}^{N} err_i^2 = \sum_{i=0}^{N} (y_i - d_i)^2$$

$$\text{minimiser l'erreur} \Longleftrightarrow \frac{\nabla J}{\nabla \vec{\mathbf{W}}} = 0$$

$$\frac{\nabla J}{\nabla \vec{\mathbf{W}}} = 2 \cdot \sum_{i=0}^{N} \left(f_{act} \left(\vec{\mathbf{X}} \cdot \vec{\mathbf{W}}^T \right) - d_i \right) \cdot \frac{\partial f_{act} \left(\vec{\mathbf{X}} \cdot \vec{\mathbf{W}}^T \right)}{\partial \vec{\mathbf{W}}}$$

$$\frac{\nabla J}{\nabla \vec{\mathbf{W}}} = 2 \cdot \sum_{i=0}^{N} \left(f_{act} \left(\vec{\mathbf{X}} \cdot \vec{\mathbf{W}}^T \right) - d_i \right) \cdot f'_{act} \left(\vec{\mathbf{X}} \cdot \vec{\mathbf{W}}^T \right) \cdot \vec{\mathbf{X}}$$

Le gradient - solutions

Le gradient :

$$\frac{\partial J\left(\vec{\mathbf{W}}\right)}{\partial \vec{\mathbf{W}}} = 2 \cdot \sum_{i=0}^{N} \left(f_{act}\left(\vec{\mathbf{X}} \cdot \vec{\mathbf{W}}^T\right) - d_i \right) \cdot f_{act}^{'}\left(\vec{\mathbf{X}} \cdot \vec{\mathbf{W}}^T\right) \cdot \vec{\mathbf{X}}$$

- Si $f_{act}^{'} = 1$

$$\frac{\partial J\left(\vec{\mathbf{W}}\right)}{\partial \vec{\mathbf{W}}} = 2 \cdot \sum_{i=0}^{N} \left(f_{act}\left(\vec{\mathbf{X}} \cdot \vec{\mathbf{W}}^T\right) - d_i \right) \cdot \vec{\mathbf{X}}$$

Le perceptron mono-couche

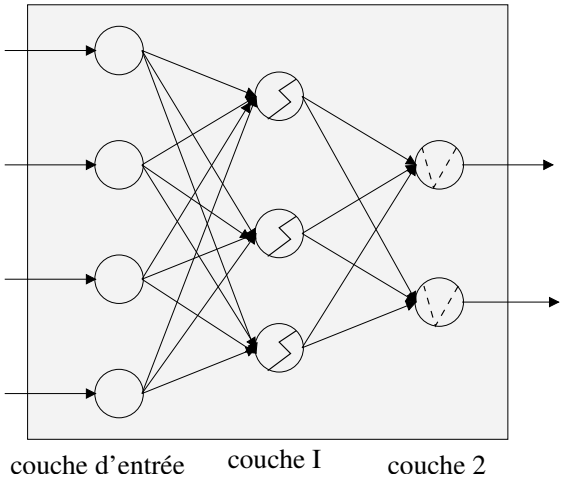
Réseaux à une couche avec neurones avec fonction d'activation croissante :

- On peut montrer que l'algorithme converge si le problème est linéairement séparable.
- permet une régression linéaire et une discrimination linéaire
- le modèle est simple mais utile

A toujours utiliser comme référence avant un traitement par un réseau de neurones plus puissant.

Perceptron multi-couche

- Réseau feedforward (1986)



- Fonction de transfert $\tanh(\cdot)$ (sauf couche de sortie $\longrightarrow$ linéaire)
- Méthode d'apprentissage (supervisé) usuelle : la rétropropagation du gradient

Perceptron multicouche - notations

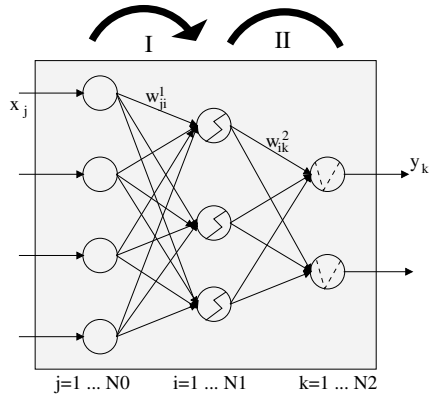
- Soit $y_i^{(k)}$ la sortie du neurone i de la couche k.

$$y_i^k = f_{act}\left(\sum_{j=0}^{N_1} y_j^{(k-1)} \cdot w_{ji}^{(k)}\right)$$

- Soit $w_{ji}^{(k)}$ le poids entre le neurone j de la couche $k-1$ et le neurone i de la couche k.
- Soit $w_{ji}^{(1)} = w_{ij} = \mathbf{\vec{W}_1}$ les poids de la première couche
- Soit $w_{ik}^{(2)} = w_{ik} = \mathbf{\vec{W}_2}$ les poids de la seconde couche

Perceptron multicouche - propagation

- Calcul des sorties du réseaux en propageant les valeurs des entrées de couche en couche



- Etape I :

$$y_i^{(1)} = y_i = f_{act}\left(\sum_{j=0}^{N_0} w_{ji}^{(1)} \cdot x_j\right)$$

- Etape II :

$$y_k^{(2)} = y_k = f_{act}\left(\sum_{i=0}^{N_1} w_{ik}^{(2)} \cdot y_i^{(1)}\right)$$

Calcul de l'erreur

- Fonction de coût :
  - on présente un exemple $\vec{\mathbf{X}} = [x_1 ... x_{N_0}]$ avec $\vec{\mathbf{D}} = [d_1 ... d_{N_2}]$ la sortie désirée
  - on calcule la sortie correspondante $\vec{\mathbf{Y}} = [y_1 ... y_{N_2}]$
  - erreur : $err_k = d_k - y_k$
  - coût associé à l'exemple :

$$J_{exemple} = \frac{1}{2} \cdot \sum_{k=1}^{N_2} err_k^2$$

  - coût global :

$$J = \sum_{m=1}^{N_m} J_{exemple\ m}$$

Calcul du gradient

- Mise à jour de $w_{ji}^{(1)}$ et de $w_{ik}^{(2)}$ selon une règle delta :

$$w(t) = w(t-1) + \Delta w$$

avec

$$\Delta w = \eta \cdot \frac{\partial J}{\partial w}$$

avec η le pas d'apprentissage

- Il faut calculer

$$\frac{\partial J}{\partial w_{ji}^{(1)}}, \quad \frac{\partial J}{\partial w_{ik}^{(2)}}$$

Couche de sortie - I

Calcul de $\frac{\partial J}{\partial w_{ik}^{(2)}}$ pour un exemple donné :

$$\frac{\partial J}{\partial w_{ik}^{(2)}} = \frac{\partial J}{\partial y_k^{(2)}} \cdot \frac{\partial y_k^{(2)}}{\partial p_k^{(2)}} \cdot \frac{\partial p_k^{(2)}}{\partial w_{ik}^{(2)}}$$

Avec :

$$J = \frac{1}{2} \cdot \sum_{k=1}^{N_2} \left(d_k - y_k^{(2)} \right)^2 \Longrightarrow \frac{\partial J}{\partial y_k^{(2)}} = - \left(d_k - y_k^{(2)} \right)$$

$$y_k^{(2)} = f_{act} \left(p_k^{(2)} \right) \Longrightarrow \frac{\partial y_k^{(2)}}{\partial p_k^{(2)}} = f_{act}^{'} \left(p_k^{(2)} \right)$$

$$p_k^{(2)} = \sum_{i=0}^{N_1} y_i^{(1)} \cdot w_{ik} \Longrightarrow \frac{\partial p_k^{(2)}}{\partial w_{ik}^{(2)}} = y_i^{(1)}, \quad (i = k)$$

Couche de sortie - II

- Soit

$$Err_k \equiv \left(d_k - y_k^{(2)}\right) \cdot f_{act}^{'}\left(p_k^{(2)}\right)$$

- Mais pour la couche de sortie f_{act} linéaire $\longrightarrow f_{act}^{'} = 1$
- Donc :

$$\frac{\partial J}{\partial w_{ik}^{(2)}} = -\left(d_k - y_k^{(2)}\right) \cdot f_{act}^{'}\left(p_k^{(2)}\right) \cdot y_i^{(1)}$$

$$\Longrightarrow \frac{\partial J}{\partial w_{ik}^{(2)}} = Err_k \cdot y_i^{(1)}$$

- Le calcul du gradient de la couche cachée ne dépend pas que des informations “amont”.

Couche cachée

Calcul de $\frac{\partial J}{\partial w_{ji}^{(1)}}$ pour un exemple donné :

$$\frac{\partial J}{\partial w_{ji}^{(1)}} = \frac{\partial J}{\partial y_i^{(1)}} \cdot \frac{\partial y_i^{(1)}}{\partial p_i^{(1)}} \cdot \frac{\partial p_i^{(1)}}{\partial w_{ji}^{(1)}}$$

Couche cachée $\longrightarrow$ l'erreur est "cachée"

Soit

$$J = \frac{1}{2} \cdot \sum_{k=1}^{N_2} \left(d_k - y_k^{(2)} \right)^2 \Longrightarrow \frac{\partial J}{\partial y_i^{(1)}} = \sum_{k=1}^{N_2} \left(d_k - y_k^{(2)} \right) \cdot \frac{\partial \left(d_k - y_k^{(2)} \right)}{\partial y_i^{(1)}}$$

$$\Longleftrightarrow \frac{\partial J}{\partial y_i^{(1)}} = \sum_{k=1}^{N_2} \left(d_k - y_k^{(2)} \right) \cdot \frac{\partial \left(d_k - y_k^{(2)} \right)}{\partial p_k^{(2)}} \cdot \frac{\partial p_k^{(2)}}{\partial y_i^{(1)}}$$

Couche cachée

$$\Longleftrightarrow \frac{\partial J}{\partial y_i^{(1)}} = \sum_{k=1}^{N_2} \left(d_k - y_k^{(2)}\right) \cdot \frac{\partial \left(d_k - f_{act}\left(p_k^{(2)}\right)\right)}{\partial p_k^{(2)}} \cdot \frac{\partial \left(\sum_{i=0}^{N_1} y_i^{(1)} \cdot w_{ik}\right)}{\partial y_i^{(1)}}$$

$$= \sum_{k=1}^{N_2} \left(d_k - y_k^{(2)}\right) \cdot f_{act}^{'}\left(p_k^{(2)}\right) \cdot w_{ik} = \sum_{k=1}^{N_2} \left(d_k - y_k^{(2)}\right) \cdot w_{ik} \quad (f_{act}^{'} = 1)$$

Mais

$$y_i^{(1)} = f_{act}\left(p_i^{(1)}\right) \Longrightarrow \frac{\partial y_i^{(1)}}{\partial p_i^{(1)}} = f_{act}^{'}\left(p_i^{(1)}\right)$$

$$p_i^{(1)} = \sum_{j=0}^{N_0} x_j \cdot w_{ji} \Longrightarrow \frac{\partial p_i^{(1)}}{\partial w_{ji}^{(1)}} = x_j, \quad (j = i)$$

Couche cachée

Donc

$$\frac{\partial J}{\partial w_{ji}^{(1)}} = \sum_{k=1}^{N_2} \left(d_k - y_k^{(2)} \right) \cdot w_{ik} \cdot f_{act}^{'} \left(p_i^{(1)} \right) \cdot x_j$$

Soit

$$Err_i = \sum_{k=1}^{N_2} \left(d_k - y_k^{(2)} \right) \cdot w_{ik} \cdot f_{act}^{'} \left(p_i^{(1)} \right)$$

Donc

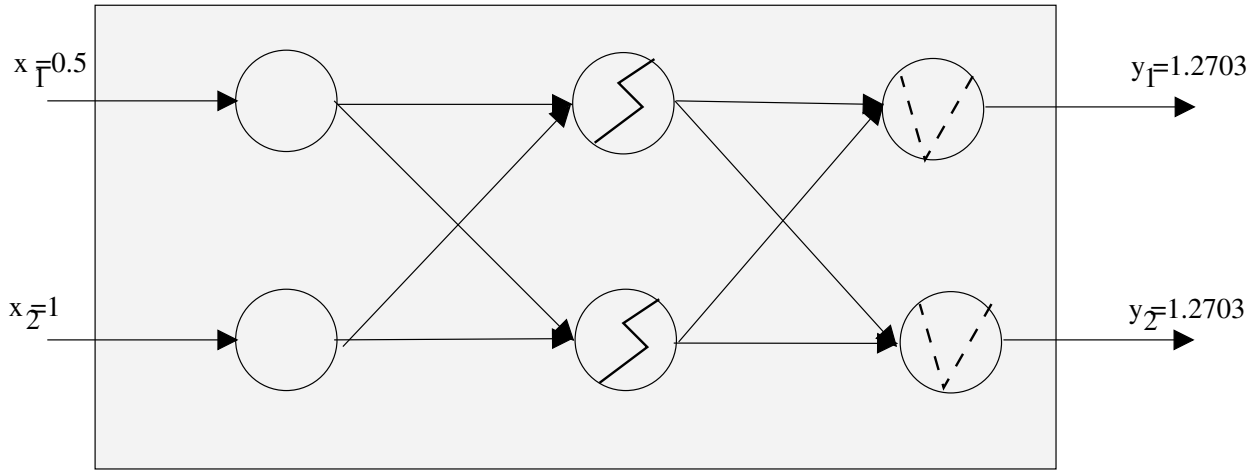
$$\frac{\partial J}{\partial w_{ji}^{(1)}} = Err_i \cdot x_j$$

Algorithme d'apprentissage

1. Initialisation des poids w_{ji} et w_{ik} avec de faibles valeurs
2. Pour chaque vecteur d'entrée $\vec{\mathbf{X}}_m$ faire :
   (a) calculer l'entrée de la couche cachée
   (b) calculer la sortie de la couche cachée
   (c) calculer la sortie de la couche de sortie
   (d) calculer les modifications des poids de la couche de sortie
   (e) calculer les modifications des poids de la couche cachée
3. vérifier si le nombre d'itérations est atteint
4. figer les poids
5. utiliser le réseaux

Exemple

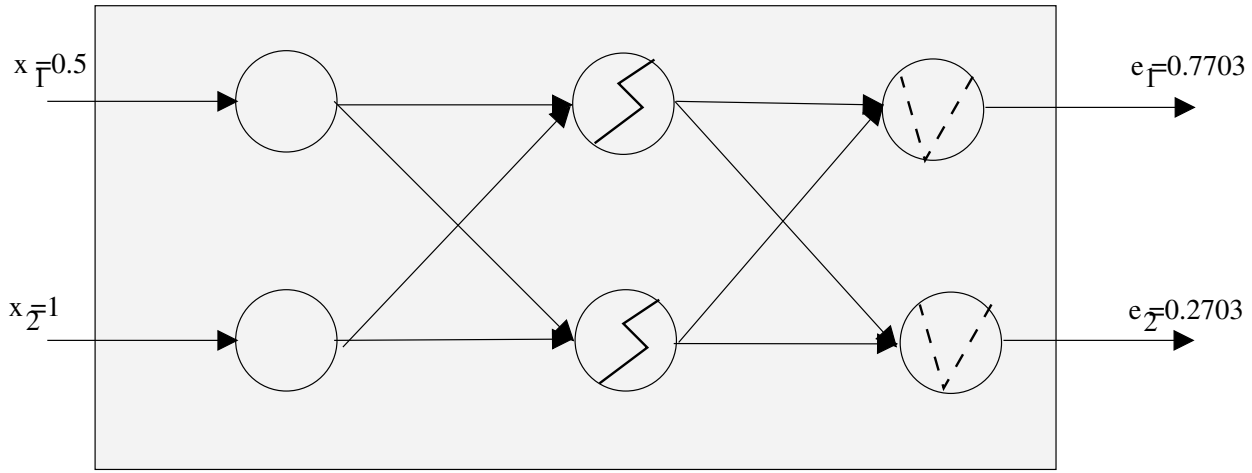
- $\vec{\mathbf{x}} = [1 \quad 0.5]$
- $\vec{\mathbf{d}} = [1 \quad 0.5]$
- $\vec{\mathbf{W}}_1 = [0.5 \quad 0.5 \; ; \; 0.5 \quad 0.5]$
- $\vec{\mathbf{W}}_2 = [1 \quad 1 \; ; \; 1 \quad 1]$



- $\vec{\mathbf{p}}^{(1)} = [0.75 \quad 0.75]$ et $\vec{\mathbf{y}}^{(1)} = [0.6351 \quad 0.6351]$
- $\vec{\mathbf{p}}^{(2)} = [1.2703 \quad 1.2703]$ et $\vec{\mathbf{y}}^{(2)} = [1.2703 \quad 1.2703]$

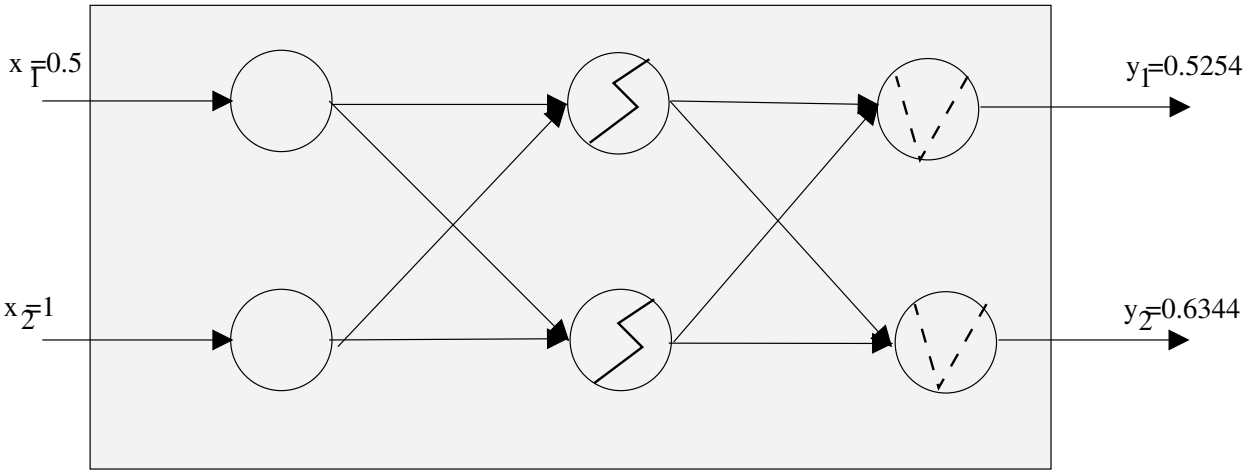
Exemple

- $\vec{\mathbf{Err}}_k = [0.7703 \;\; 0.2703]$
- $\vec{\mathbf{Grad}}_{W_2} = [0.4893 \;\; 0.1717 \; ; \; 0.4893 \;\; 0.1717]$
- $\vec{\mathbf{Err}}_i = [0.6208 \;\; 0.6208]$
- $\vec{\mathbf{Grad}}_{W_1} = [0.3104 \;\; 0.3104 \; ; \; 0.6208 \;\; 0.6208]$



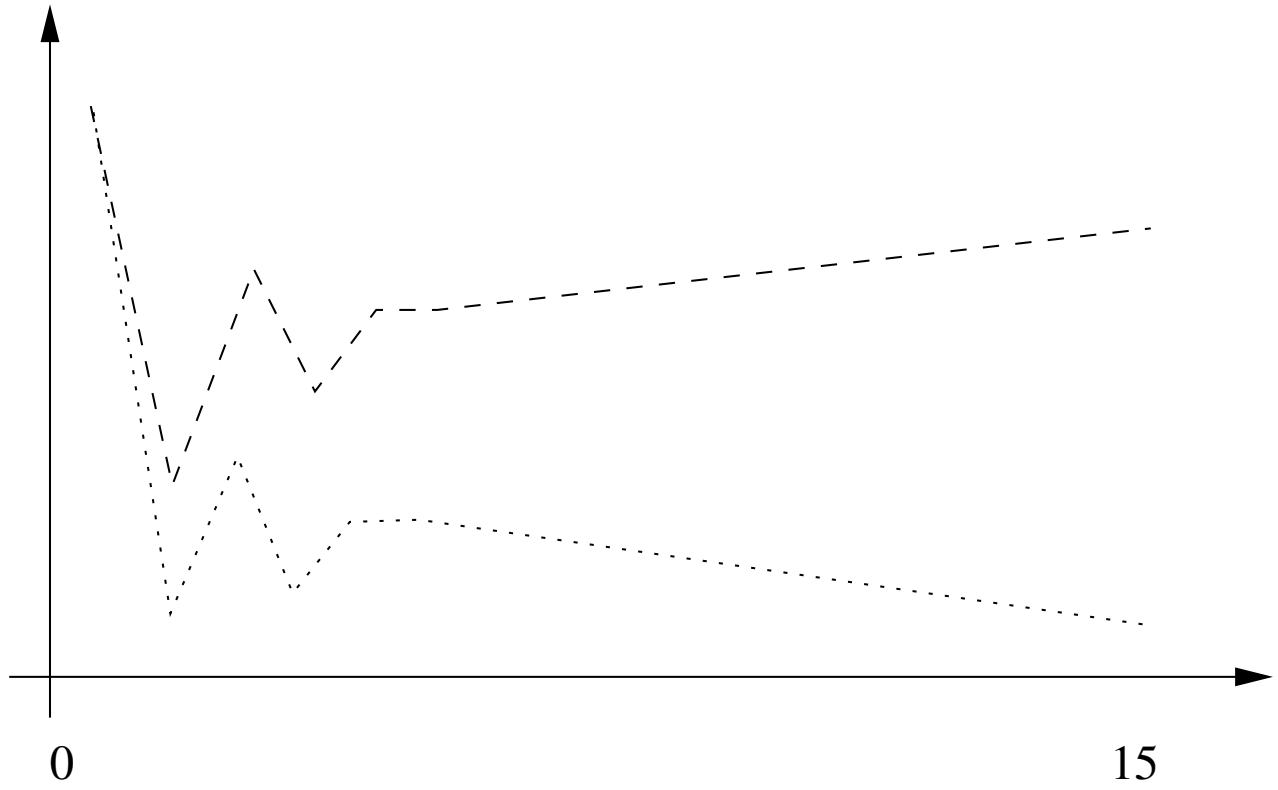
Exemple

- Mise à jour des poids $\vec{\mathbf{W}}_2 = [0.7554 \;\; 0.9142 \;;\; 0.7554 \;\; 0.9142]$
- Mise à jour des poids $\vec{\mathbf{W}}_1 = [0.3448 \;\; 0.3448 \;;\; 0.1896 \;\; 0.1896]$
- Nouvelle propagation, etc ... $\vec{\mathbf{y}} = [0.5242 \;\; 0.6344]$



Exemple

Evolution de $\vec{\mathbf{y}}$



Algorithme apprentissage MADALINE

Apprentissage par perturbation minimum.

1. Application d'un vecteur d'apprentissage
2. Calcul de l'erreur (nb. de sorties incorrectes/couche de sortie)
3. Pour tous les neurones de la couche de sortie :
   (a) Choisir un neurone non encore sélectionné dont la sortie analogique est proche de zéro (ce neurone peut provoquer une commutation du système à seuil par une perturbation minimum de ses poids)
   (b) Modifier les poids de ce neurone de manière à faire commuter sa sortie
   (c) Propager de nouveau le vecteur d'apprentissage
   (d) Si l'erreur est réduite alors valider la modification des poids, sinon revenir à la configuration initiale
4. Répéter l'étape 3 pour toutes les couches sauf la couche d'entrée

Bases fonctionnelles

La puissance du modèle vient de la notion de base :

- dans un espace vectoriel, on peut toujours trouver une base des vecteurs linéairement indépendants $(e_i)_{i \in N}$
- tout vecteur s'écrit $x = \sum_{i \in N} a_i \cdot e_i$ (avec un nombre fini de a_i non nuls)
- dans un espace hilbertien, on peut définir des bases "approximatives"

Ces résultats s'appliquent aux espaces fonctionnels $\Rightarrow$ approximation universelle.

Approximation universelle

En choisissant bien une suite de fonctions $(\phi_i)_{i \in N}$, on peut représenter approximativement toute fonction. Exemples : polynômes (Bernstein, Lagrange, etc.), B-splines, fonctions trigonométriques (séries de Fourier), etc.

On a une propriété d'approximation universelle si la suite $(\phi_i)_{i \in N}$ est bien choisie : pour toute fonction "régulière" f et toute précision $\epsilon > 0$, il existe un réseau de neurones à deux couches basés sur la suite $(\phi_i)_{i \in N}$ (pour la première couche) et utilisant des neurones linéaires dans la seconde couche, qui *calcule* une fonction proche de f à ϵ près.

Conséquences pratiques

A priori, on a que des avantages :

- Le perceptron multi-couche est strictement plus puissant que le modèle mono-couche (approximation universelle)
- l'apprentissage reste rapide

Il y a quand même des inconvénients :

- le temps de calcul augmente avec le nombre de neurones (temps de calcul fixé par les dimensions du problème)
- choix difficile de la puissance du modèle (i.e., du nombre de neurones)

Problème de dimension

Première apparition de la malédiction des grandes dimensions (curse of dimensionality). Polynômes :

- dans $\Re$, $a_0 + a_1 x + a_2 x^2 + ...$: le degré k correspond à $k+1$ coefficients
- dans $\Re^2$, $a_{0,0} + a_{1,0} x + a_{1,0} y + a_{1,1} xy + a_{2,0} x^2 + a_{0,2} y^2 ...$: le degré k correspond à $\frac{(k+1)(k+2)}{2}$ coefficients
- dans $\Re^n$, le degré k correspond à $\frac{(k+n)!}{k!n!}$, soit $O(n^k)$ coefficients pour n grand.

Pour toutes les bases classiques (B-splines, séries de fourier, etc.), on a des résultats similaires.

Problème de dimension (2)

Traduction pratique :

- plus la dimension d'entrée (le nombre de variables) est élevée, plus l'augmentation de puissance du modèle est coûteuse
- la croissance est exponentielle

Conséquences :

- les bases classiques sont délicates à utiliser pour $n > 2$
- le réglage de la puissance pose problème

On peut cependant revenir à une croissance linéaire.

Fonction à base radiale

Une solution simple pour éviter l'explosion du nombre de paramètres

$$\phi_j(x) = \rho_j\left(\|x - \mu_j\|^2\right)$$

où ϕ_j est une fonction de $\Re$ dans $\Re$. On connaît des conditions très larges sur ρ_j qui permettent de conserver la propriété d'approximation universelle. En général, on prend la fonction gaussienne :

$$\rho_j(x) = e^{-\frac{x}{2\sigma_j^2}}$$

Augmentation de puissance : ajout d'un neurone, i.e. de n à n+1 paramètres numériques (μ et σ) linéaire.

Réseaux à fonctions de base radiale

- Réseau feedforward à une couche cachée (Broomhead & Lowe 1988) (Moody & Darken 1989)
- Fonction de transfert :
  - gaussienne
$$f_{act} = \exp\left(\frac{\| x - \mu \|^2}{2\sigma^2}\right)$$
  - toute autre fonction noyau
- Méthode d'apprentissage usuelle :
  - Nb et paramètres des gaussiennes (apprentissage non supervisé) :
    * Clustering : regroupement des points en classes (LVQ, K-means). Nb gaussiennes = nb de clusters trouvés
    * Paramètres de chaque gaussienne = (μ, σ) de chaque cluster
  - Couche de sortie (apprentissage supervisé) :

Conclusions

- Avantages :
  - Le PMC utilise une méthode de gradient stochastique
  - C'est simple, calculs locaux
  - Tolérance aux défaillances hardware (dégradation locale)
  - Lissage de l'erreur moyenne (modif. du pas d'apprentissage)
- Problèmes :
  - Minima locaux (initialisation des poids)
  - On n'a pas d'indications sur M, le nombre de neurones sur la couche cachée !
  - L'apprentissage supervisé

Bibliographie

Livres :

- *Neural Networks : a comprehensive foundation - S. Haykin (Macmillan Publishing)*
- Neural Networks for Pattern Recognition - C. M. Bishop (Clarendon Press)
- The Handbook of Brain Theory and Neural Networks - M.A. Arbib (MIT Press)
- Réseaux Neuronaux et Traitement du Signal - J. Hérault & C. Jutten (Hermès)

Sites :

- http://www.dontveter.com
- http://www.emsl.pnl.gov:2080/proj/neuron/neural/systems

L'apprentissage non-supervisé

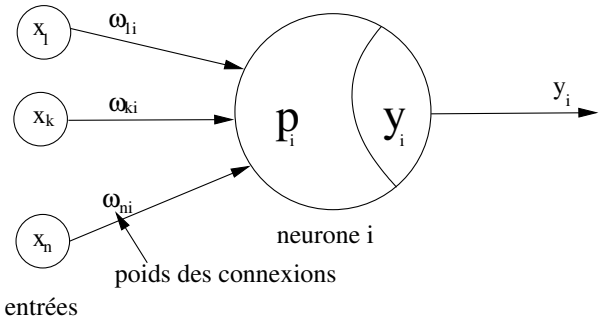
Définition : Algorithmes d'apprentissage en sens large pour lesquels nous ne disposons pas d'une indication explicite du résultat attendu pour chaque exemple.

Permet aux réseaux de neurones capables de s'auto-organiser.

Plus "biologiquement plausibles".

Rappel - le neurone formel

- Le neurone artificiel = modélisation très simplifiée du neurone biologique (McCulloch & Pitts - 1943)



- Potentiel

$$p_i = \sum_{k=1}^{N} x_k \cdot w_{ki} + \theta = \sum_{k=0}^{N} x_k \cdot w_{ki} = \vec{\mathbf{X}} \cdot \vec{\mathbf{W}}^T$$

- Sortie

$$y_i = f_{act}\left(p_i\right) = f_{act}\left(\sum_{k=0}^{N} x_k \cdot w_{ki}\right)$$

Plan

- La règle de Hebb
- Winner Take All
- Les cartes auto-organisatrices (Kohonen)
- Adaptative Resonance Theory (ART)
- Renforcement
- Conclusions

La règle de Hebb

La plus ancienne loi d'apprentissage (1949).

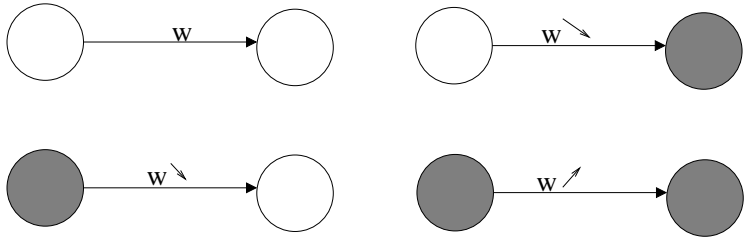
> *"Quand une cellule A excite par son axone une cellule B et que, de manière répétée et persistante, elle participe à la genèse d'une impulsion dans B, un processus de croissance ou un changement métabolique a lieu dans l'une ou dans les deux cellules, de telle sorte que l'efficacité de A à déclencher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

Dans le cas de neurones formels, on peut décrire la règle de Hebb par l'équation :

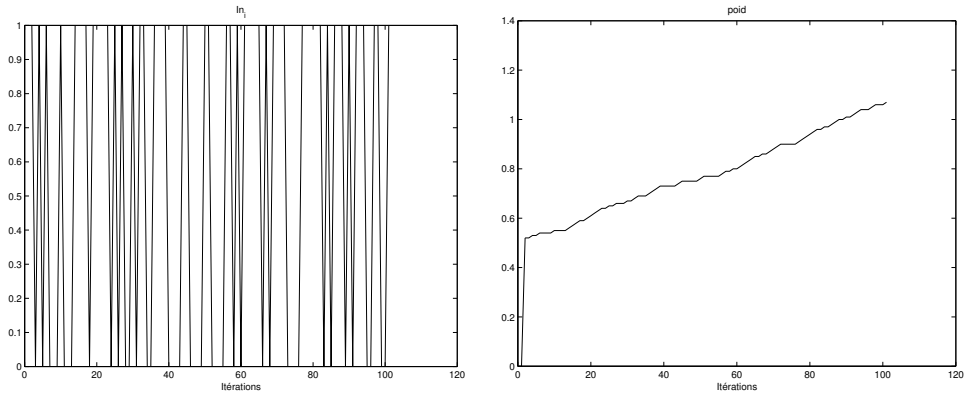
$$\omega_{ij}\left(t+\delta t\right) = \omega_{ij}\left(t\right) + \epsilon \cdot E_i \cdot S_j$$

Hebb

Règle de modification des poids de Hebb selon 4 cas possibles :



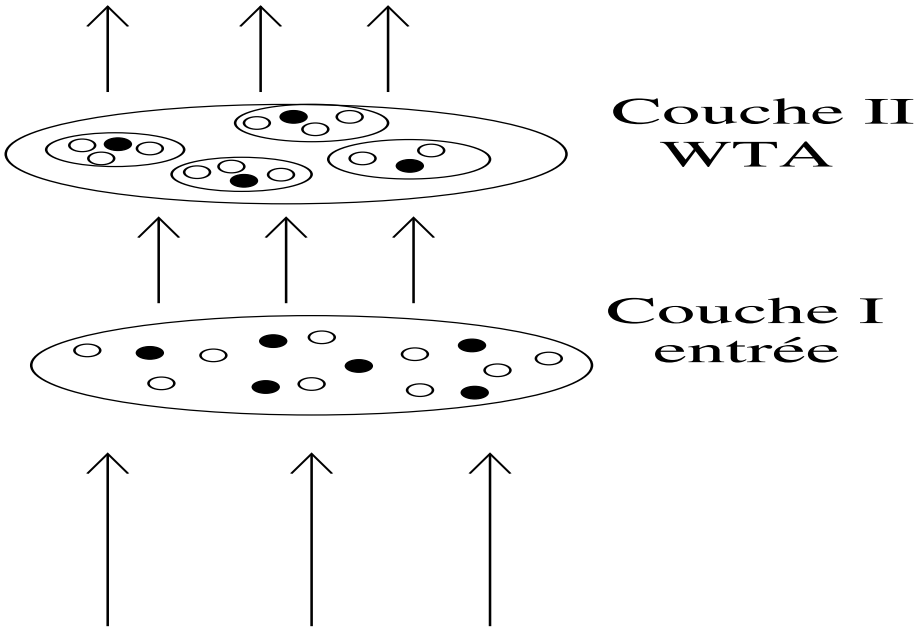
Exemple avec 2 neurones :



Problème de stabilité : + d'activation + d'augmentation des poids = explosion! Solution : la normalisation, le seuillage, l'inhibition

Winner Take All

- Permet de modéliser les mécanismes de compétition
- Utilisation de connexions latérales
- Fin: un seul neurone actif

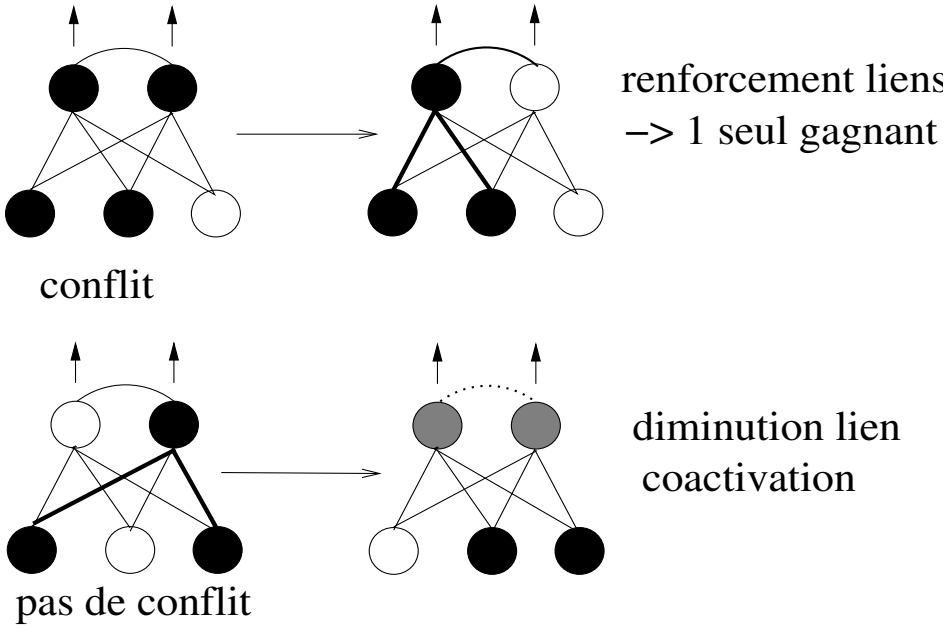


- Quel mécanisme de compétition utiliser?

Winner Take All - apprentissage

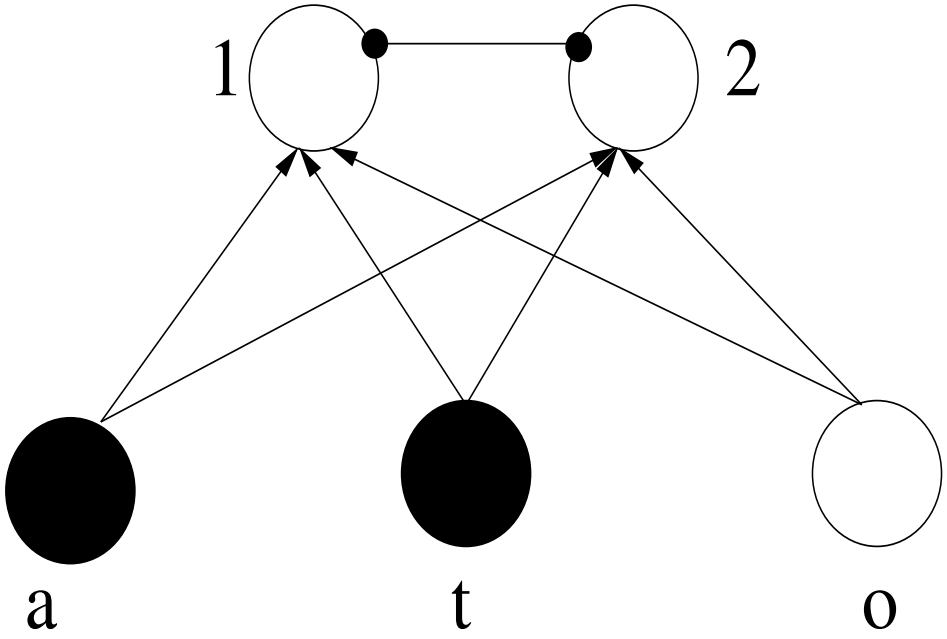
- connexions excitatrices : Hebb
- connexions inhibitrices :

  A chaque fois qu'un neurone est actif, ses connexions inhibitrices (sorties) vers d'autres neurones actifs sont renforcées. Ses connexions inhibitrices vers des neurones inactifs sont diminuées.



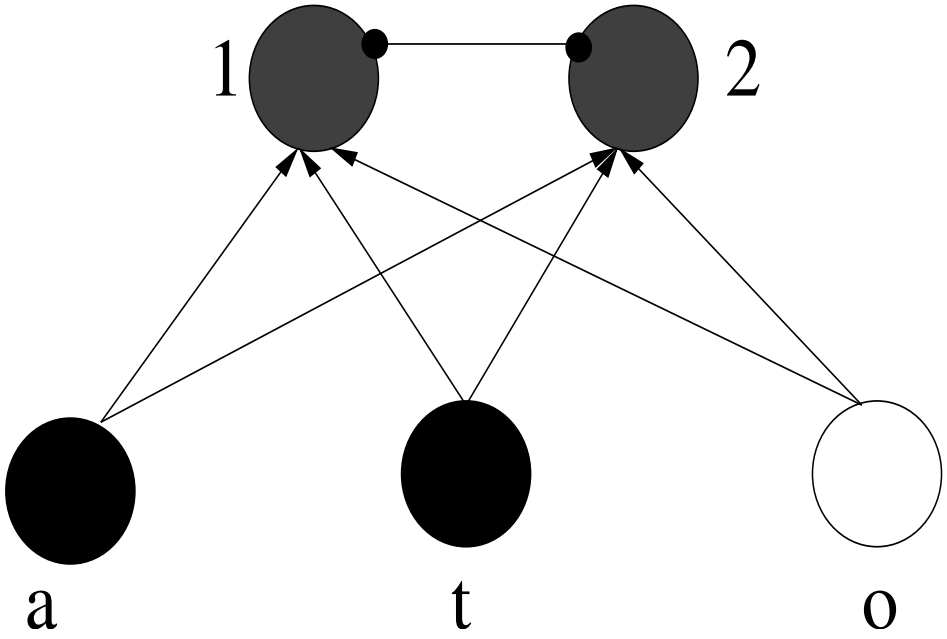
Winner Take All - exemple

Les entrées “a” et “t” sont activées.



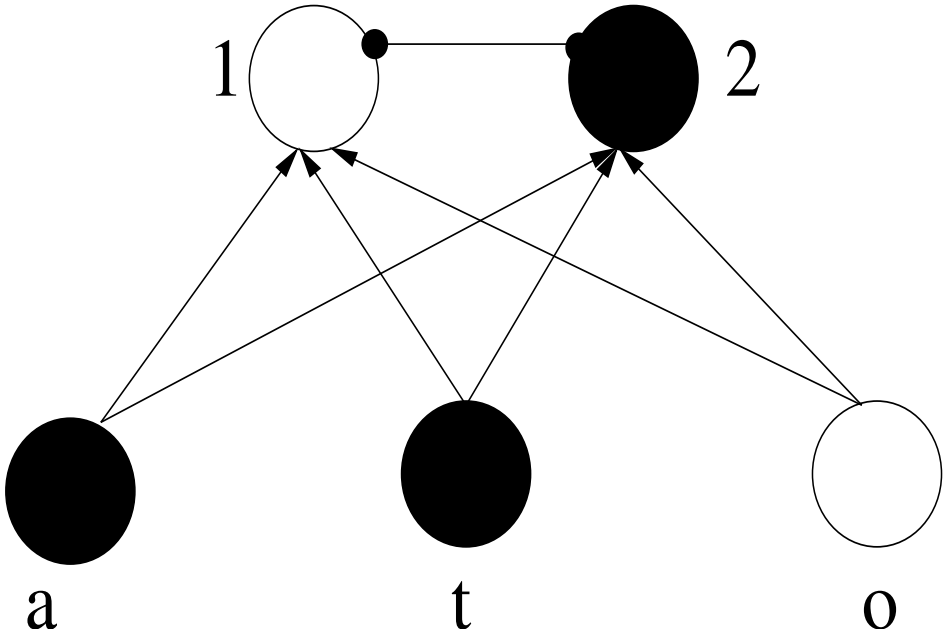
Winner Take All - exemple

(que) La “compétition” commence ...



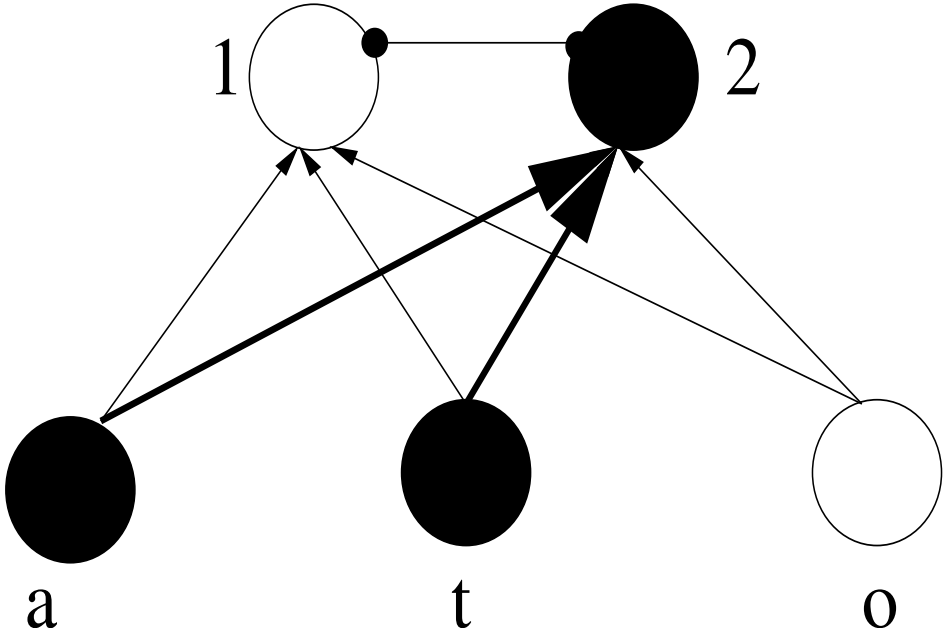
Winner Take All - exemple

Le “plus fort” à gagné ...



Winner Take All - exemple

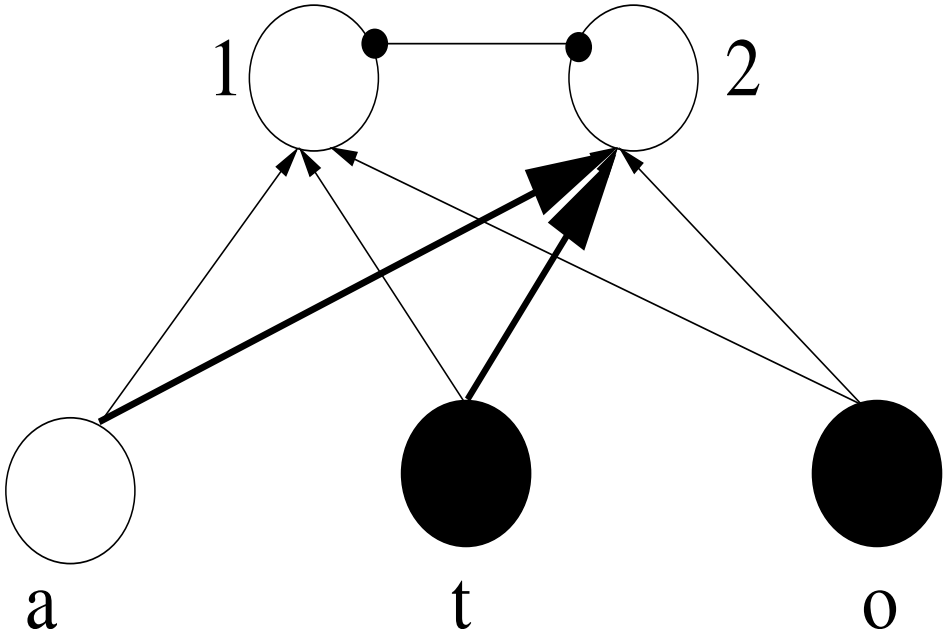
Apprentissage hebbien ...



L'entrée "at" est associée à la catégorie "2".

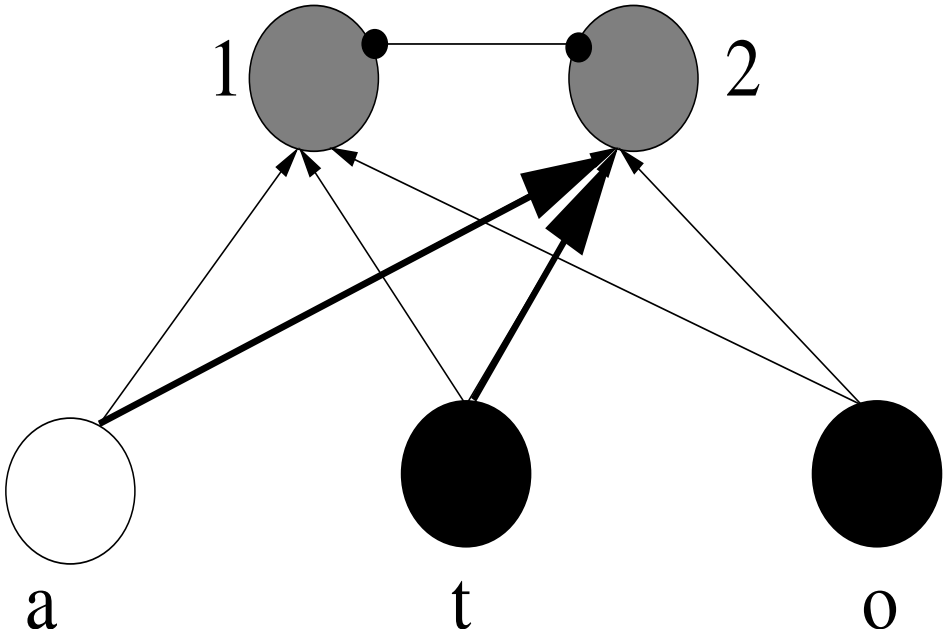
Winner Take All - exemple

Les entrées “t” et “o” sont actives.



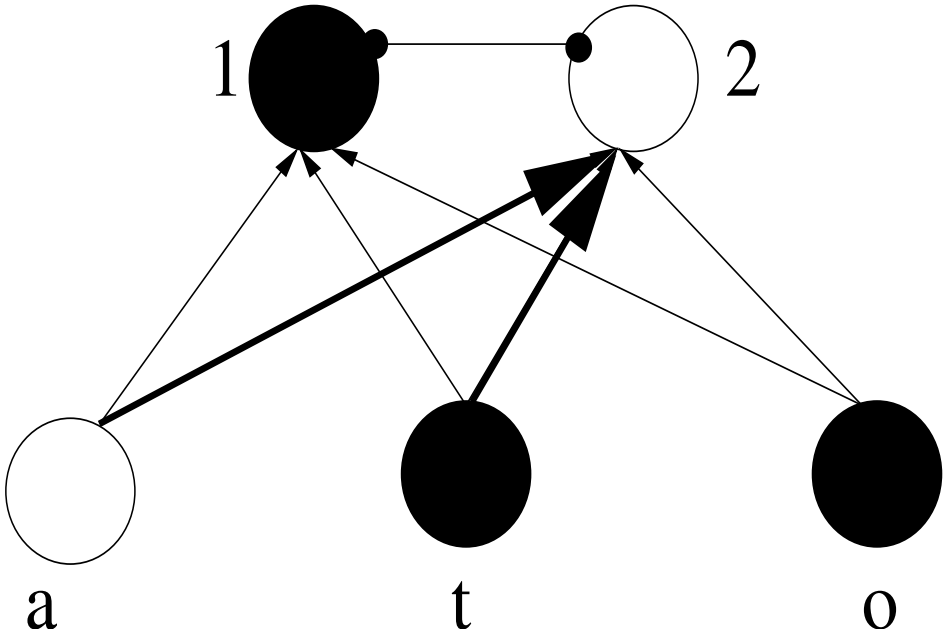
Winner Take All - exemple

(que) La “compétition” commence ...



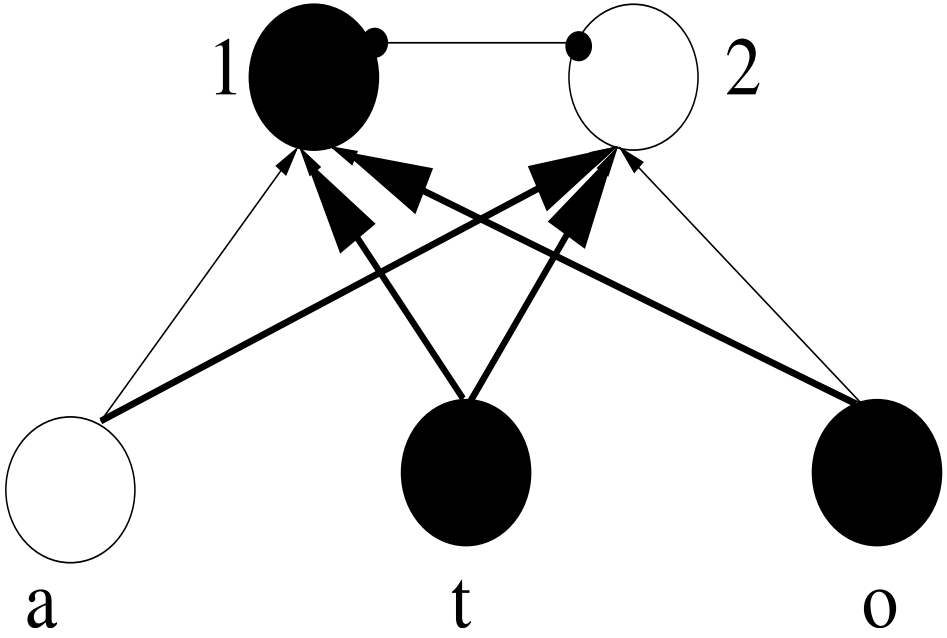
Winner Take All - exemple

Le “plus fort” à gagné ...



Winner Take All - exemple

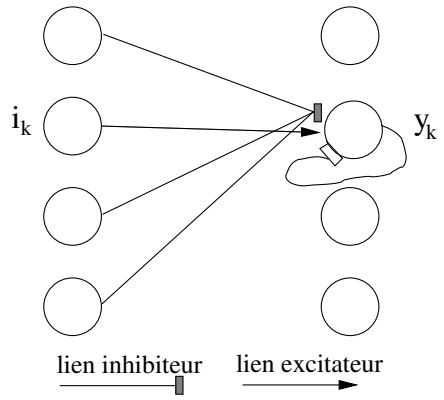
Apprentissage hebbien ...



L'entrée "to" est associée à la catégorie "1".

WTA - Instar

- Utilisé comme mécanisme de rehaussement de contraste



- mise à jour dynamique

$$\frac{dy_k}{dt} = -A \cdot y_k + (B - y_k) \cdot i_k - y_k \sum_{l \neq k} i_l$$

- avec y_k valeur du potentiel du neurone k, i_l valeur sur la couche I, A, B constantes

WTA - Instar

- Après stabilisation :

$$\frac{dy_k}{dt} = 0 \Longleftrightarrow y_k \cdot (A + I) = B \cdot i_k, \quad I = \sum_l i_l$$

- La sortie est proportionnelle à l'entrée :

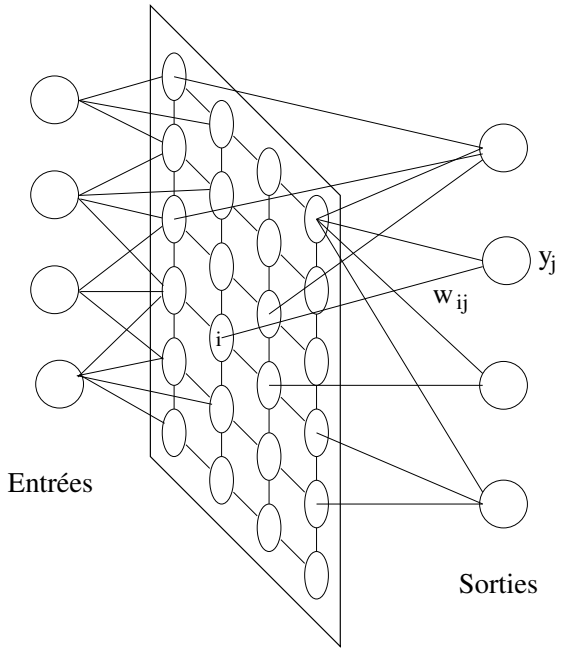
$$y_k = i_k \cdot \frac{B}{A + I}$$

- la somme des sorties est normalisée :

$$y_{total} = \sum_{k=1}^{N} y_k = \sum_{k=1}^{N} i_k \cdot \frac{B}{A + I} = \frac{I \cdot B}{A + I} = constante \leq B$$

Cartes topologiques

- Observation biologique : les neurones ont un rôle spécifique + neurones voisins réagissent à des entrées qui se ressemblent.
- Kohonen (1977) introduit le concept de cartes topologiques auto adaptatives.



Kohonen

- Kohonen part de l'équation dynamique :

$$\frac{\Delta S}{\Delta t} = E - p\,(S)$$

avec E l'entrée totale, $p\,(S)$ terme de perte non-linéaire +saturation

- stabilisation

$$\frac{\Delta S}{\Delta t} = 0 \longleftrightarrow S = p^{-1} \cdot \sum_i w_i \cdot e_i$$

avec p^{-1} une sigmoïde.

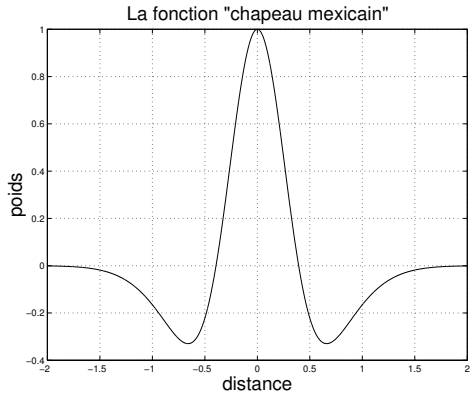
Kohonen - apprentissage

- Kohonen introduit l'oubli

$$\frac{\Delta S}{\Delta t} = k \cdot S \cdot e_i - r\left(S\right) \cdot w_i$$

  avec $r\left(S\right)$ fonction d'oubli

- le mécanisme d'apprentissage est :
  - Si le neurone reçoit un potentiel d'action par sa connexion i et qu'il s'active, il y a renforcement de cette connexion
  - sinon, le poids de cette connexion est diminué
- interactions latérales



Kohonen - algorithme

Pour N neurones avec M entrées et le rayon r :

- *Les poids sont initialisés aléatoirement.*
- *Tant que le réseau ne converge pas :*
  - *Choisir un vecteur d'entrée* $I = [I_1, I_2, ..., I_M]$ *dans l'ensemble des vecteurs d'entrée à apprendre.*
  - *Trouver le neurone gagnant (le neurone* j *dont le vecteur poids* $W_j = [W_{j1}, W_{j2}, ..., W_{jM}]$ *est le plus proche, selon une distance* d*, du vecteur d'entrée).*

$$j = Arg\left(\min_{i=1..N} d\left(I, W_i\right)\right)$$

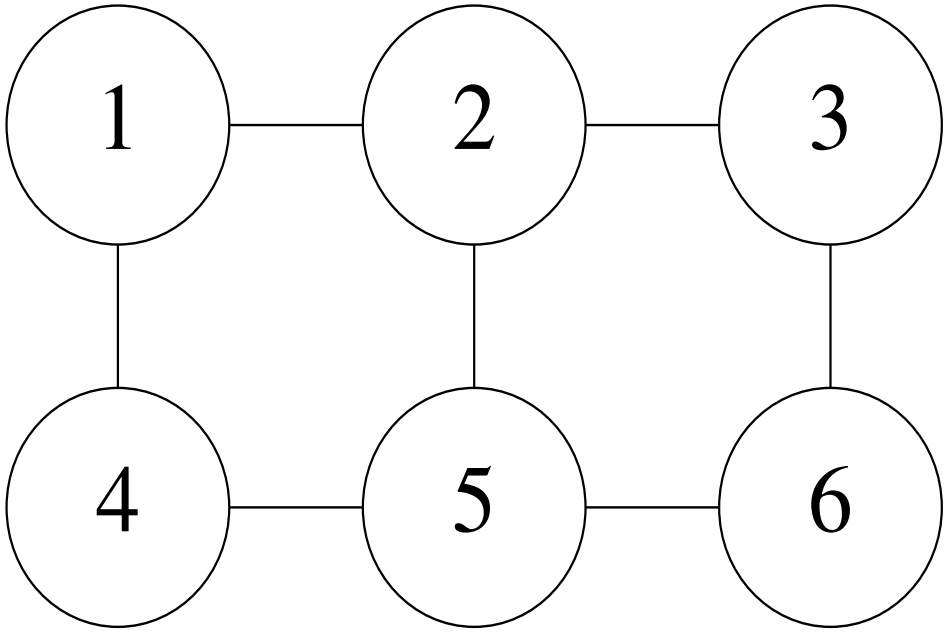
  - *Modifier les poids du vecteur gagnant selon l'équation :*

$$si\ |k-j| \leq r \rightarrow W_{kj}(t+1) = W_{kj}(t) + \eta(t) \cdot (I_j(t) - W_{kj}(t))$$

- *Fin tant que*

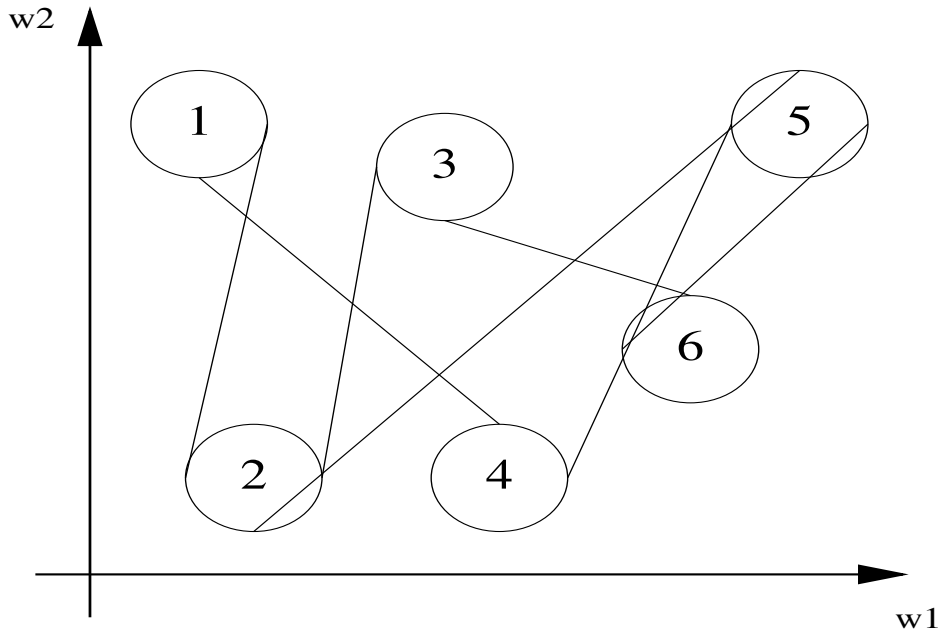
SOM - exemple

Considérons les entrées dans l'espace $\Re^2$ et soit la carte topologique suivante (6 neurones sur une grille) :



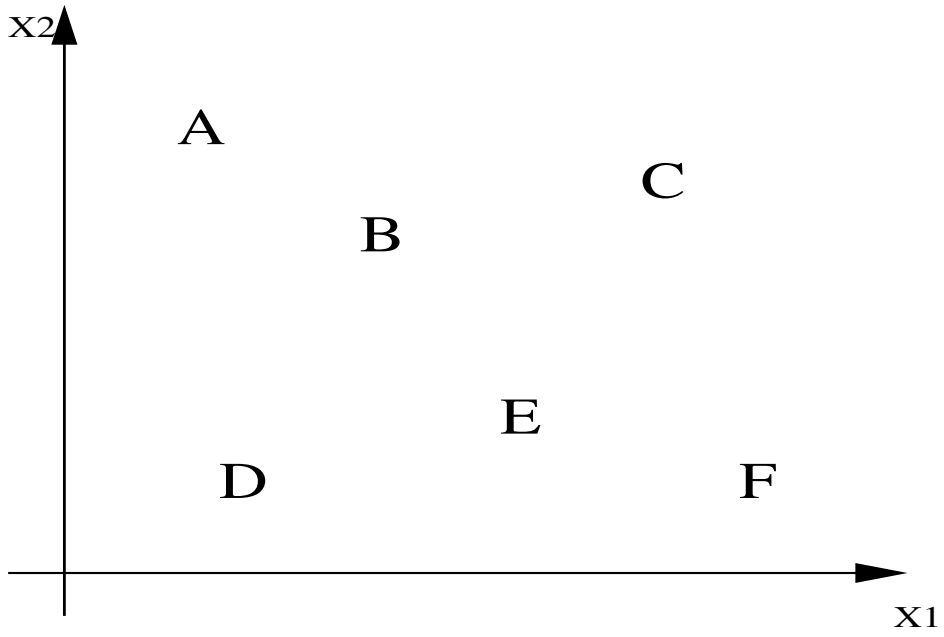
SOM - exemple

Après initialisation des poids, passons dans l'espace des poids (chaque neurone n'as que 2 voisins).



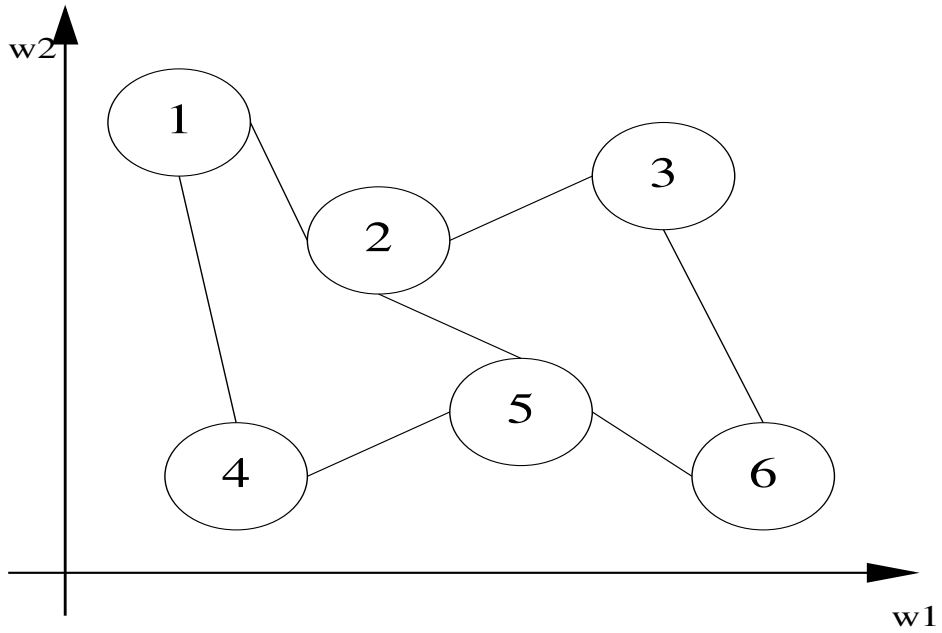
SOM - exemple

Considérons que les vecteurs d'entrée dont nous disposons (ils sont tous en $\Re^2$!) peuvent être représentes par la figure suivante :



SOM - exemple

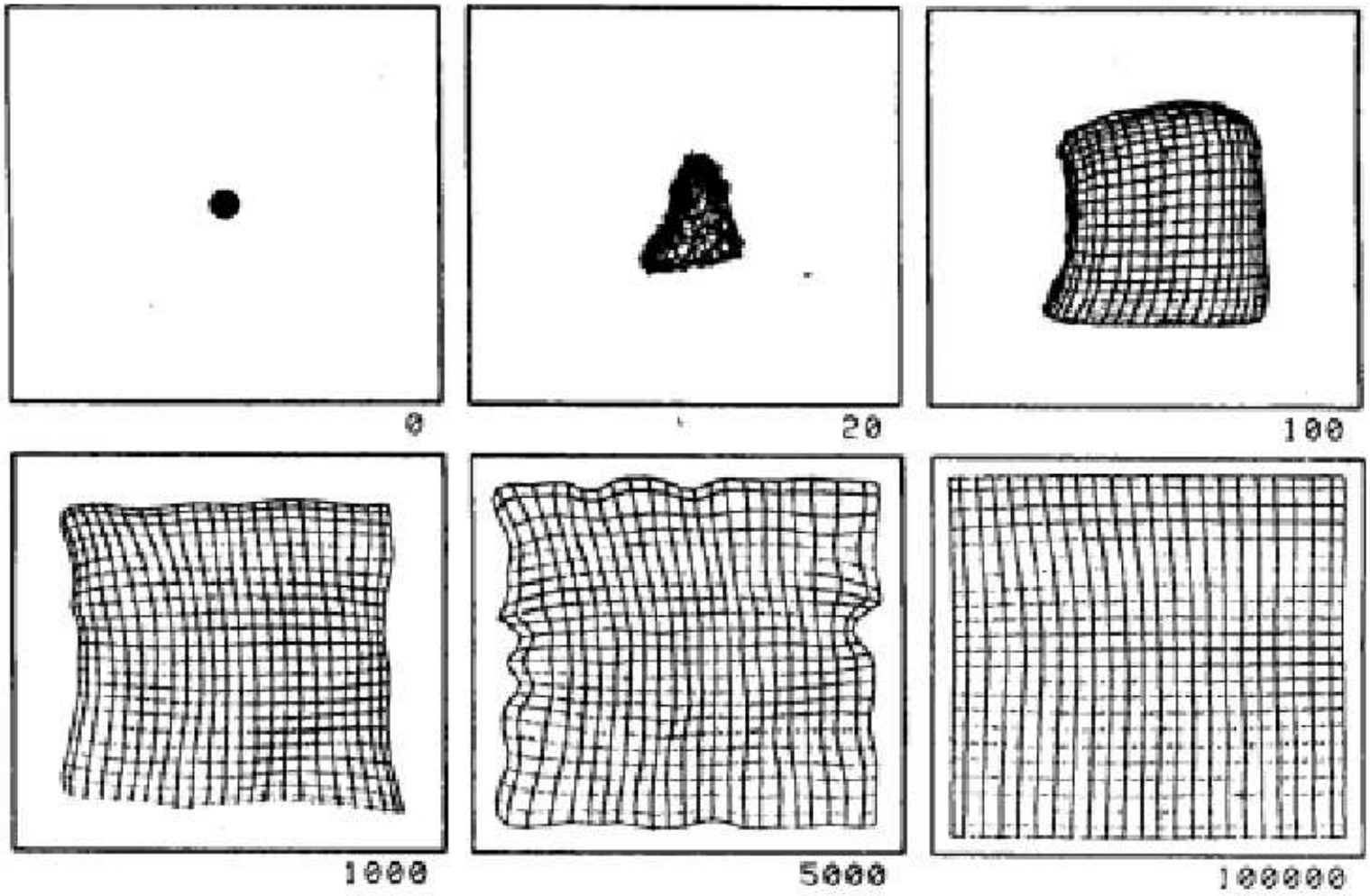
Après apprentissage, les connexions entre les neurones voisins sont :



La même topologie que celle des entrées !!!

SOM - exemple

Considérons maintenant le cas ou les entrées sont dans $\Re^2$ ...



Kohonen - conclusion

- présentation de tous les vecteurs d'entrée
- réalisation d'une projection d'un espace N à un espace M
- avantages :
  - l'espace de sortie est un espace de représentations
    $\rightarrow$ on peut visualiser les sorties
  - représentation des données de grande dimension
- inconvénients :
  - le temps de convergence
- ???
  - pas de preuve de convergence
  - pas d'unicité de la représentation

Kohonen - exemple

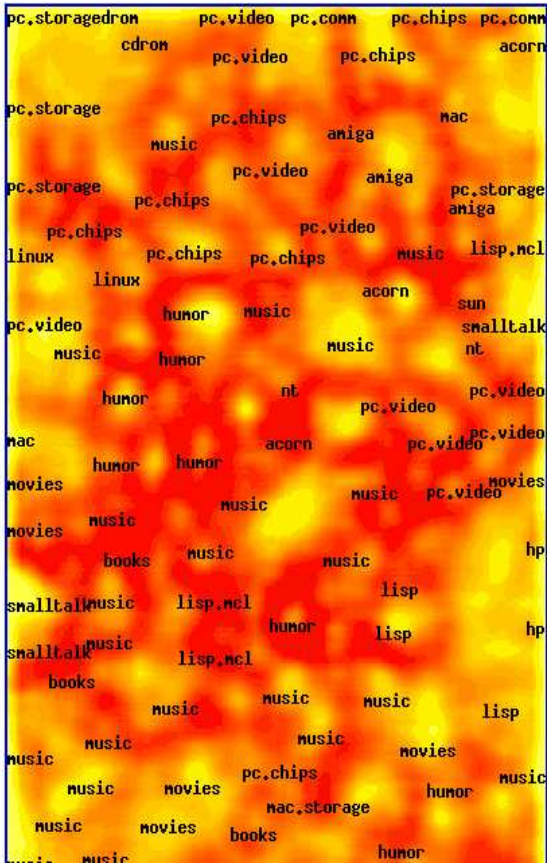
Websom (websom.hut.fi)

– représentation de documents issus du web

– projection du contenu des 7 millions de messages (newsgroups)

– le textes se resemblant sont représentés dans des endroits proches

– la culeur traduit le nombre de textes projétes dans la zone (rouge=faible)

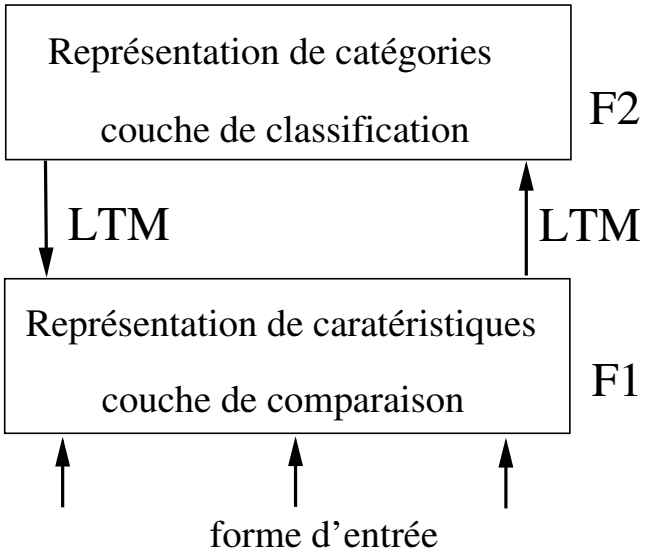


Adaptive Resonance Theory - ART

- famille de modèles de RN inspirés de la psychologie visuelle (Grossberg - 1980) :
  - *la normalisation* de l'activité totale du réseau.
  - *le rehaussement de contraste*
  - *la mémorisation* à court terme de la forme
- harmoniser le dilemme entre l'adaptation (plasticité synaptique) et la stabilité (rigidité synaptique) de l'apprentissage.
  - Système trop plastique : les informations sont apprises même si elles ne sont pas pertinentes, ou les informations apprises sont oubliées.
  - système trop rigide : les informations pertinentes ne sont pas apprises.
- introduction d'un terme de *vigilance* permet de gérer de manière autonome le passage d'un mode "plastique" à un mode "rigide".

ART - schéma

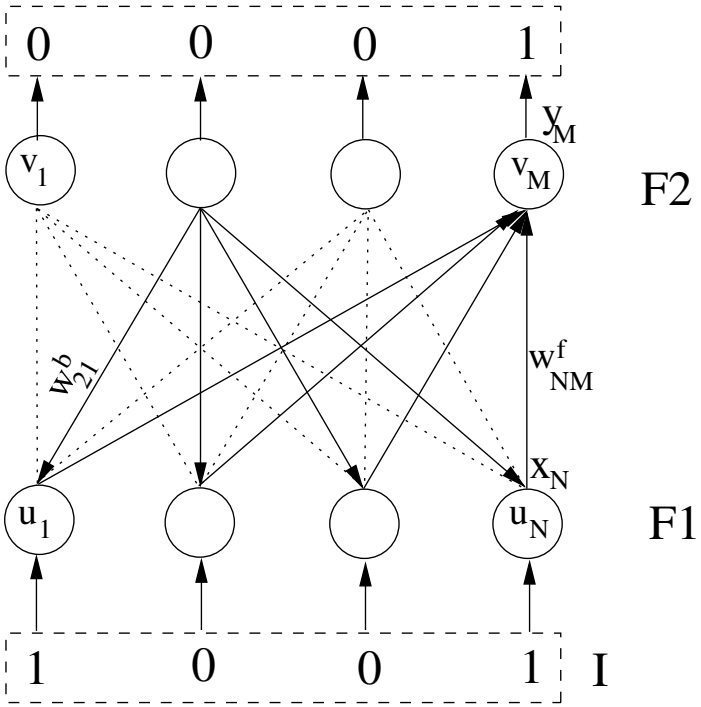
- Réseau à 2 couches (STM)
- Connexions récurrentes entre les 2 couches



- F1 : couche de comparaison / représentation de caractéristiques
- F2 : couche de classification / représentation de catégories
- Principe : on apprend une nouvelle forme ssi elle est très différentes de celles qui sont déjà apprises.

ART - Algorithme

Schéma du modèle ART1:



1. Initialiser les poids des connexions :

$$w_{ji}^b(0) = 1 \quad w_{ij}^f(0) = \frac{L}{L-1+N};$$

ART - Algorithme

2. *Présenter la forme d'entrée I et la propager vers F1. Calculer les activités U et les sorties X pour F1 :*

$$U = X = I$$

3. *Propager les sorties de F1 vers F2 :*
   - *Calculer les activités v_j pour F2.*

$$v_j = \sum_{i=1}^{N} w_{ij}^f \cdot x_i$$

   - *Calculer les sorties y_j pour F2.*
     Soit $J = \arg\left(\max_k(v_k)\right)$ l'indice du neurone gagnant.

$$y_j = \begin{cases} 1 & si\ j = J \\ 0 & sinon \end{cases}$$

4. *Rétro-propager les sorties de F2 vers F1.*

ART - Algorithme

- *Calculer les entrées t_i de F1 provenant de F2.*

$$t_i = \sum_{j=1}^{M} y_j \cdot w_{ji} = w_{Ji}$$

l'indice J est celui du neurone gagnant à l'étape précédente.

- *Calculer les nouvelles activités et sorties pour F1.*

$$U = X = I \bigwedge T$$

5. *Déterminer le degré de correspondance.*

$$\frac{|X|}{|I|} = \frac{\sum_i x_i}{\sum_i I_i}$$

6. *Comparer le degré de correspondance avec la vigilance ρ.*
   - *Si $|X|/|I| < \rho$, inhiber le neurone J pour éviter qu'il gagne à nouveau et retourner à l'étape 3 avec le même vecteur d'entrée.*

ART - Algorithme

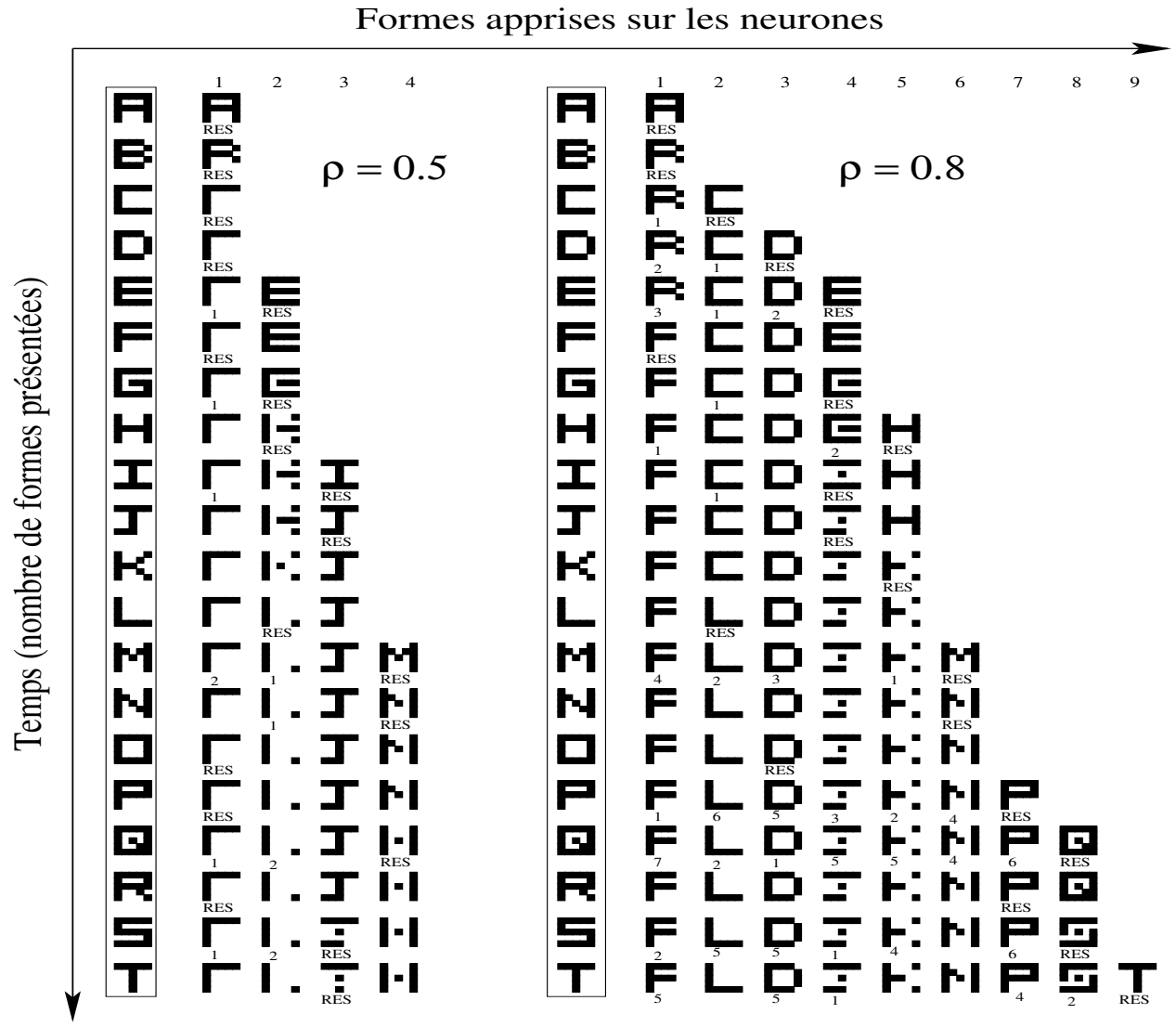
- $\|X\|/\|I\| > \rho$, *continuer*

7. *Mettre à jour les poids de connexions reliées au neurone gagnant* v_J.

$$w_{iJ}^f = \begin{cases} \frac{L}{L-1+|X|} & si \;\; x_i \; actif \\ 0 & si \;\; x_i \; inactif \end{cases} \qquad w_{Ji}^b = \begin{cases} 1 & si \;\; x_i \; actif \\ 0 & si \;\; x_i \; inactif \end{cases}$$

8. *Réactiver tous les neurones de F2 inhibés à l'étape 6 et répéter l'algorithme à partir de l'étape 3 avec un nouveau vecteur d'entrée.*

ART - Exemple


Formes apprises sur les neurones
Temps (nombre de formes présentées)
ρ = 0.5
ρ = 0.8
RES

ART - Conclusions

- Catégorisation enligne
- Apprentissage top/down sur des informations down/top.
- ART = système physique avec un fréquence de résonance + un mécanisme d'amortissement.

Le renforcement

- Les techniques de renforcement = apprentissage par essai et erreur
- Utilisent des signaux “critiques” pour apprendre
  - les actions dans “la bonne direction” sont récompensés
  - les actions dans “la mauvaise directions” sont punies
  - que faire des actions “sans effet immédiat” ?
    - les punir : représentation action/pénalité
    - les ignorer : représentation but/récompense

Le conditionnement instrumental

- Sutton & Barto 1981
- Fonction d'activation :

$$y_j = f_{Heaviside}\left(\sum_i w_{ij} \cdot y_i + bruit\right)$$

- le bruit ajouté en sortie est introduit pour permettre au réseau de choisir une action alors qu'aucune association sensori-motrice n'existe pas au préalable !
- règle d'apprentissage :

$$\frac{\Delta w_{ij}(t+1)}{\Delta t} = \epsilon \cdot \frac{\Delta R(t)}{\Delta t} \cdot \frac{\Delta y_j(t-1)}{\Delta t} \cdot x_i(t-1)$$

- il n'y a pas de retard entre l'apparition du stimulus et la récompense

Le conditionnement instrumental

- variation du renforcement :

Variation renforcement	Variation de la sortie	Entrée	Δw
$\nearrow$	$\nearrow$	Active	$\nearrow$
$\searrow$	$\nearrow$	Active	$\searrow$
0	$\nearrow$ ou $\searrow$	Active	0
$\nearrow$ ou $\searrow$	0	Active	0
$\nearrow$ ou $\searrow$	$\nearrow$ ou $\searrow$	Inactive	0

- Le mécanisme n'est possible que s'il y a concomitance de la réponse et du signal de renforcement!

Les techniques du renforcement

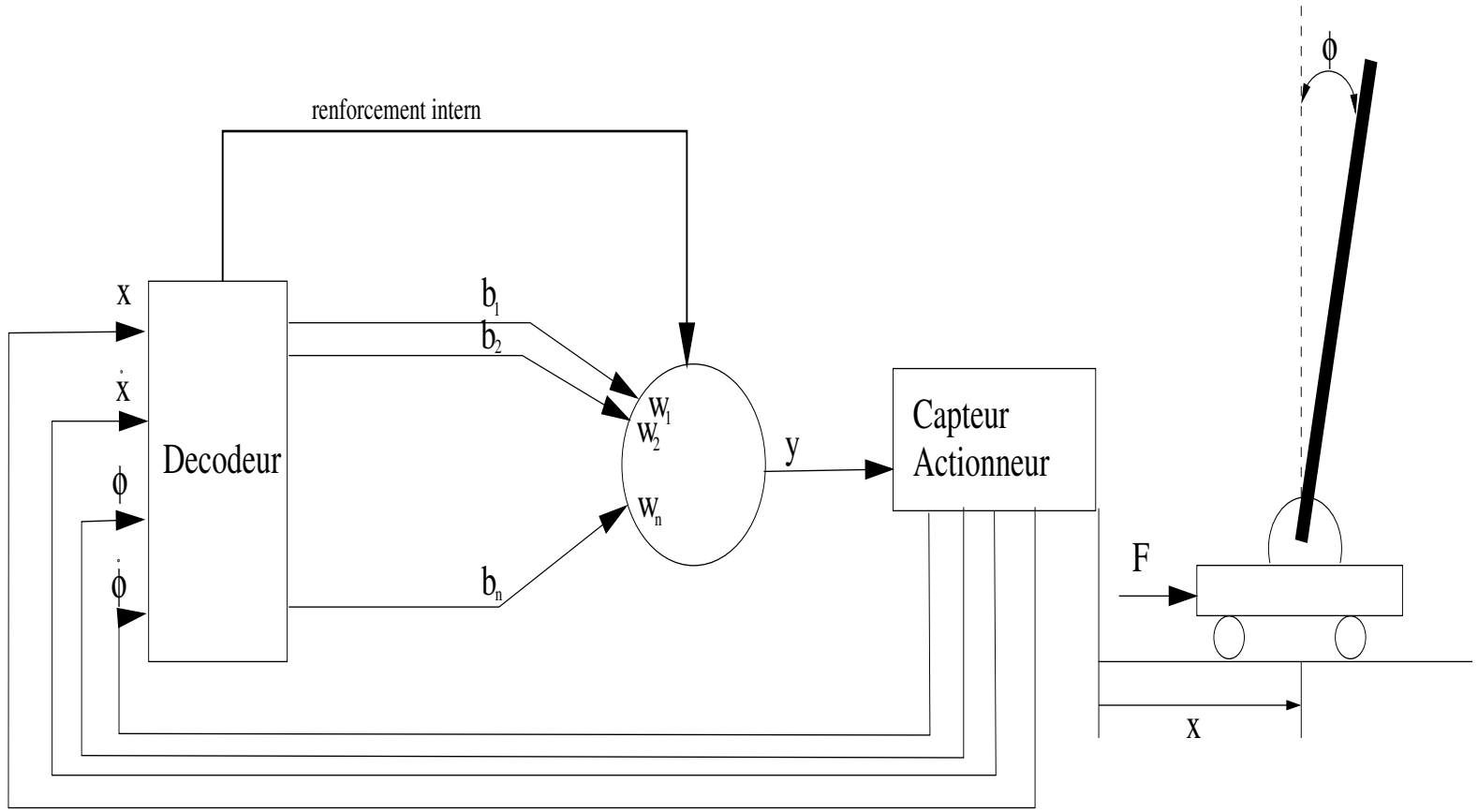
Schéma de fonctionnement général :

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.

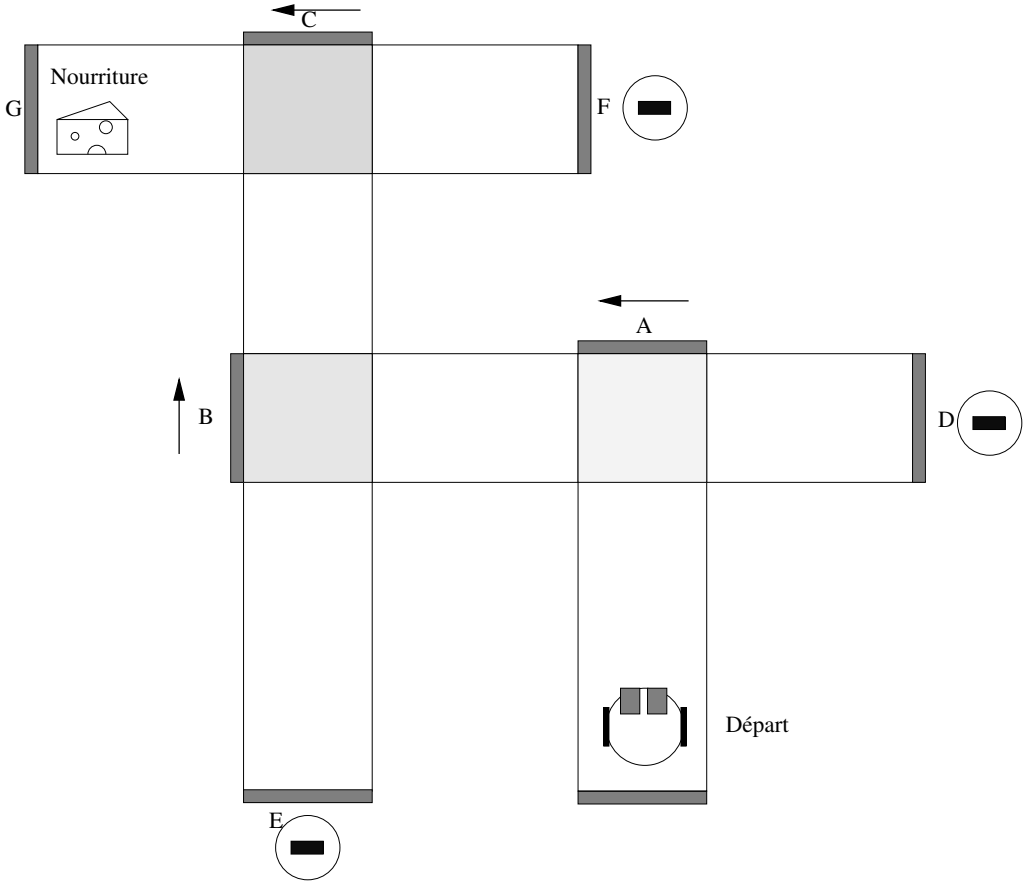
Exemple

Le chariot :

- signal renforcement : il tombe ou pas
- action possible : bouger (à gauche ou à droite)



Application possible



L'algorithme $TD(\lambda)$

- Principe : utiliser les prédictions successives effectuées au cours du temps et non l'erreur entre la prédiction actuelle et la récompense réellement reçue
- la mise à jour :

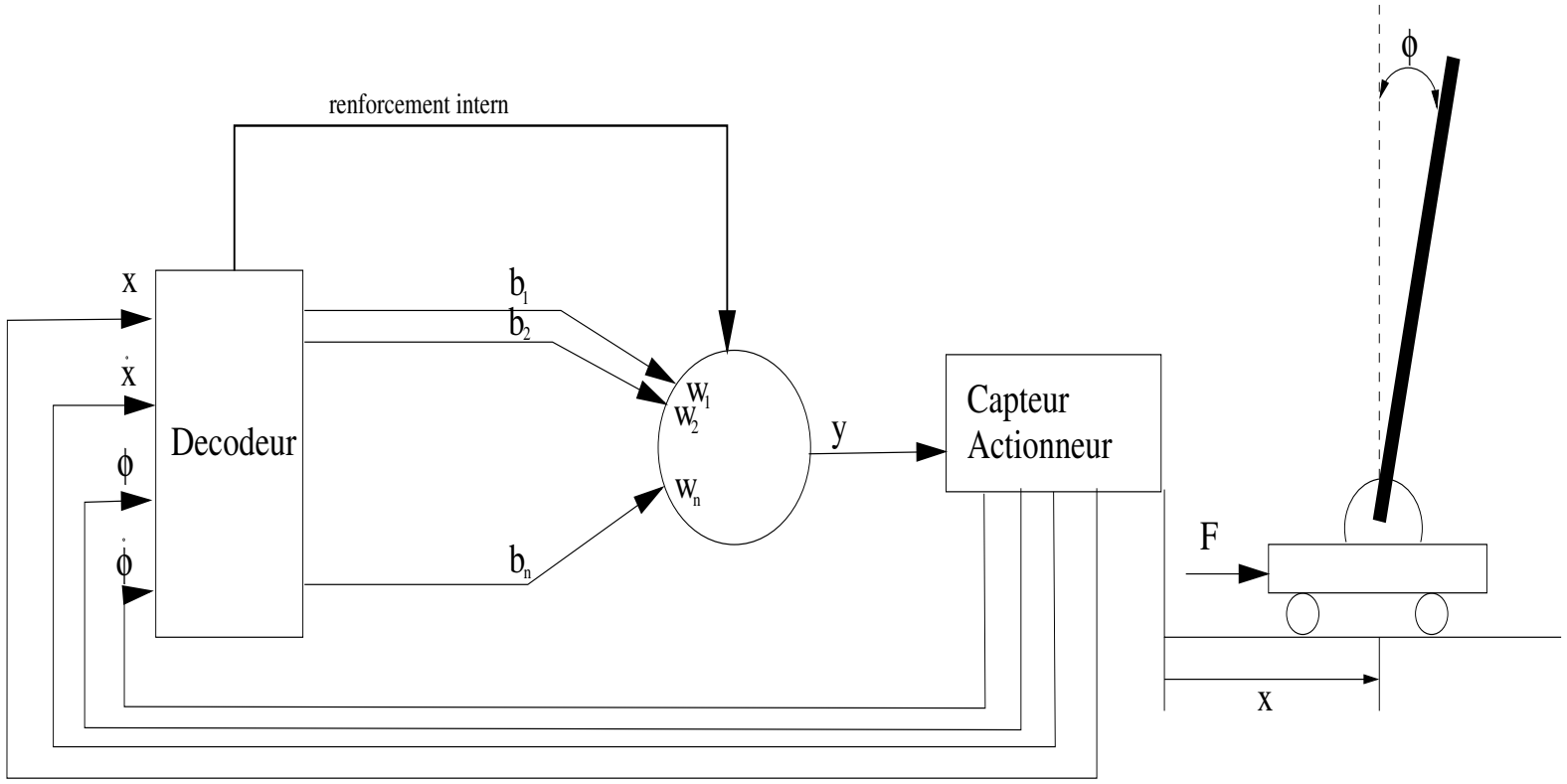
$$V(S) \leftarrow V(S) + \alpha \cdot \Big(r + \gamma \cdot V(S^{'}) - V(S) \Big) \cdot e\,(S)$$

  avec $e\,(S)$ une fonction d'éligibilité
- la convergence de cet algorithme à été montré!
- il nécessite un temps de calcul important.

Exemple

Le chariot :

- signal renforcement : proportionnellement avec le temps depuis la dernière fois quand il est tombé.
- action possible : bouger (à gauche ou à droite)



L'algorithme Q-lerning

- Principe :introduction d'un paramètre $Q(S, A)$ qui, pour chaque état S, donne la valeur estimée de la récompense totale pouvant être obtenue en effectuant l'action A.
- la mise à jour :

$$Q(S, A) \leftarrow (1 - \alpha) \cdot Q(S, A) + \alpha \cdot \left(r + \gamma \cdot Q_{max}(S^{'}) \right)$$

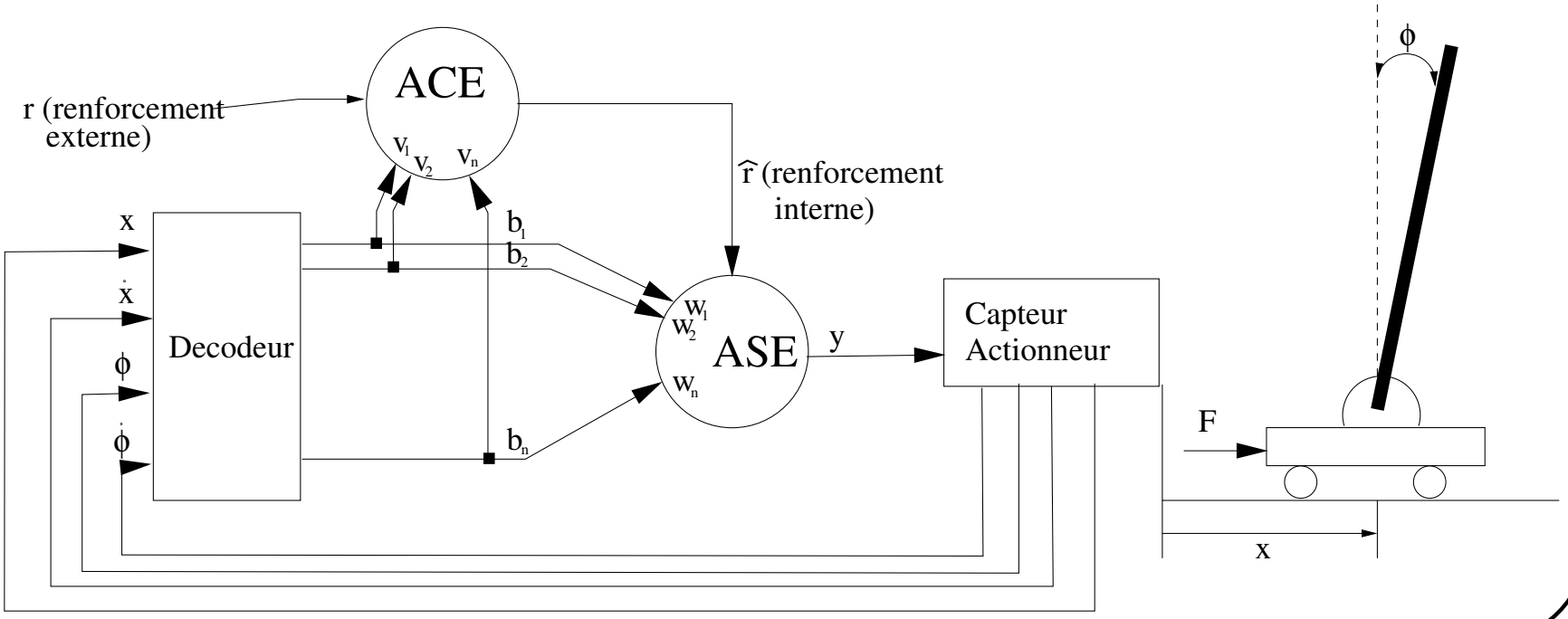
avec

$$Q_{max}(S^{'}) = \max_{A^{'} \in A_i} Q(S^{'}, A^{'})$$

- démonstration du fait que $Q(S, A)$ tend vers l'optimum
- facile d'implantation
- convergence modérée

Application : le chariot ...

- principe :
  - partitionnement de l'espace $\longrightarrow$ espace d'états
  - choix de la fonction d'éligibilité $\longrightarrow$ pour décider quelle action passée est responsable de l'échec
  - apprentissage par renforcement $\longrightarrow$ max temps avant l'échec



Le neurone ASE

- décision :

$$y(t) = f_{act}\left(\sum_{i=1}^{n} w_i(t) \cdot b_i(t) + bruit(t)\right)$$

- activation :

$$f_{act}(x) = \begin{cases} 1, & si\ x \geq 0 \longrightarrow (\ VERS\ LA\ DROITE) \\ -1, & si\ x < 0 \longrightarrow (\ VERS\ LA\ GAUCHE) \end{cases}$$

- mise à jour :

$$w_i(t+1) = w_i(t) + \alpha \cdot r(t) \cdot e_i(t)$$

- fonction d'éligibilité :

$$e_i(t+1) = \delta \cdot e_i(t) + (1-\delta) \cdot y(t) \cdot b_i(t)$$

avec $r(t) \in \{0, -1\}$, $\delta \in (0, 1)$ et $\alpha = ct.$

Le neurone ACE

Prédiction du signal de renforcement qui renforce ASE selon la fréquence de l'échec.

- activation :

$$p(t) = \sum_{i=1}^{n} v_i(t) \cdot b_i(t)$$

- sortie :

$$\hat{r}(t) = r(t) + \gamma \cdot p(t) - p(t-1)$$

- mise à jour :

$$v_i(t+1) = v_i(t) + \beta \cdot \hat{r}(t) \cdot \bar{b}_i(t)$$

avec

$$\bar{b}_i(t) = \lambda \cdot \bar{b}_i(t) + (1-\lambda) \cdot b_i(t)$$

Choix du modèle

- choix du modèle
  - choix du réseaux
  - choix du nombre de neurones
  - choix des paramètres
- choix de la méthode
  - découpage des données
  - validation croisée (leave-one-out)
  - re-simulation (bootstrap)
  - contrôle de complexité
  - dimension de Vapnik-Chervonenkis

Découpage des données

Si on a beaucoup de données, on coupe l'ensemble en deux, apprentissage et test :

- on utilise les données de l'ensemble d'apprentissage pour estimer les paramètres du modèle
- on utilise les données de l'ensemble de test pour estimer la qualité du modèle optimal

Validation croisée

Quand on a peu de données, on ne peut pas découper l'ensemble. On introduit alors du hasard artificiellement en engendrant de nouveaux ensembles d'exemples à partir des données d'origine (ré-échantillonnage). Validation croisée :

- on coupe les données en n sous-ensembles $D_1, ..., D_n$
- pour tout i :
  - on entraîne sur les D_j avec $j \neq i$
  - on évalue sur D_i
- on somme les évaluations pour obtenir une évaluation globale

Dans le cas limite où n = N , on parle de leave-one-out.

Bootstrap

Une autre méthode de ré-échantillonnage :

- répéter n fois :
  - engendrer un nouvel ensemble de données de taille N par choix aléatoire avec remise dans les données d'origine
  - entraîner le modèle sur les nouvelles données
  - estimer le modèle sur les données restantes
- l'ensemble des erreurs obtenues donne une vision empirique de la distribution de l'erreur réelle

On peut, par exemple, calculer le biais et la variance de la distribution.

Contrôle de complexité

L'idée de base est d'étudier une combinaison :

$$O = \epsilon + C$$

ϵ désigne l'erreur de modélisation obtenue, alors que C mesure la complexité effective du modèle. Par exemple :

$$O = \frac{E}{N} + \frac{W}{N} \cdot \sigma^2$$

où E désigne l'erreur quadratique de modélisation, W le nombre de paramètres du modèle, N le nombre de données et σ^2 la variance du bruit.

Dimension de Vapnik-Chervonenkis

Très schématiquement :

- théorie spécifique à la discrimination
- mesure la puissance d'un modèle à l'aide d'un nombre dV_C, la dimension de Vapnik-Chervonenkis
- donne une borne sur l'écart maximal (en valeur absolue) entre l'erreur observée sur un ensemble de données et l'erreur réelle, en fonction de dV_C et de N.
- grande force : résultat uniforme
- grande faiblesse : pas d'hypothèse sur la distribution des données bornes très pessimistes

Sorte de généralisation uniforme des grandes déviations.

Résumé

Méthode	Inconvénient
Découpage	Beaucoup de données
Ré-échantillonnage	Temps de calcul
Complexité	Fortes hypothèses
Vapnik	Pessimiste

La situation pratique reste délicate car il faut esquiver les inconvénients. Il n'existe pas de méthode complètement automatique pour l'instant (ni en théorie, ni en pratique).

Conclusions

- L'apprentissage remplace une grande partie du développement : tests sur méthodes bayesienne, rétropopagation du gradient, 5 ppv, Kalman → résultats comparables.
- approche neuronale :
  - les résultats sont toujours bons au début
  - il est difficile à améliorer
  - il faut comprendre le problème à résoudre et de trouver une solution
- l'intelligence d'un réseau est surtout fonction de son architecture
- *Quand le seul outil dont tu disposes est un marteau, chaque problème que tu rencontreras aura tendance à ressembler à un clou.*

  "Biologie de la conscience" – G. M. Edelman