



# **L'apprentissage par renforcement**

**Philippe PREUX**

**Laboratoire d'Informatique du Littoral  
Université du Littoral Côte d'Opale  
Calais, France**

**[philippe.preux@lil.univ-littoral.fr](mailto:philippe.preux@lil.univ-littoral.fr)**

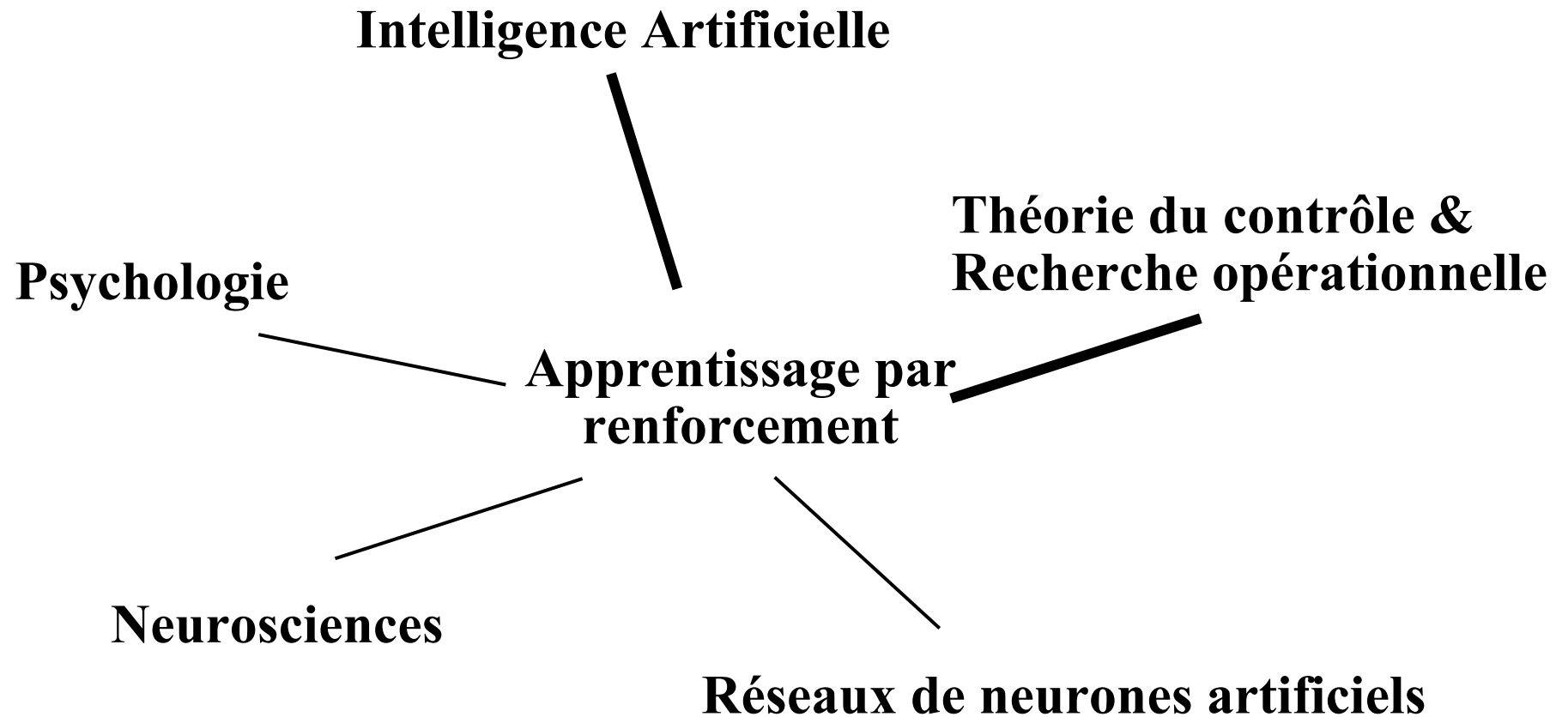
**<http://www-lil.univ-littoral.fr/~preux>**

# Plan

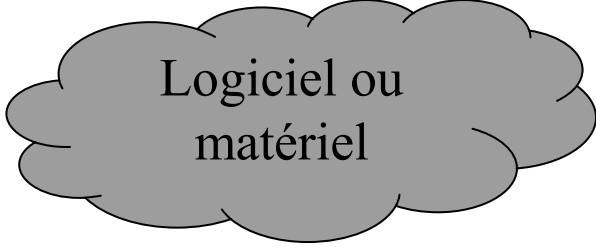
L'apprentissage par renforcement :

- c'est quoi ?
- comment on fait ?
- à quoi ça sert ?
- état des lieux / conclusion.


# **C'est quoi ? (1/14)**



# C'est quoi ? (2/14)



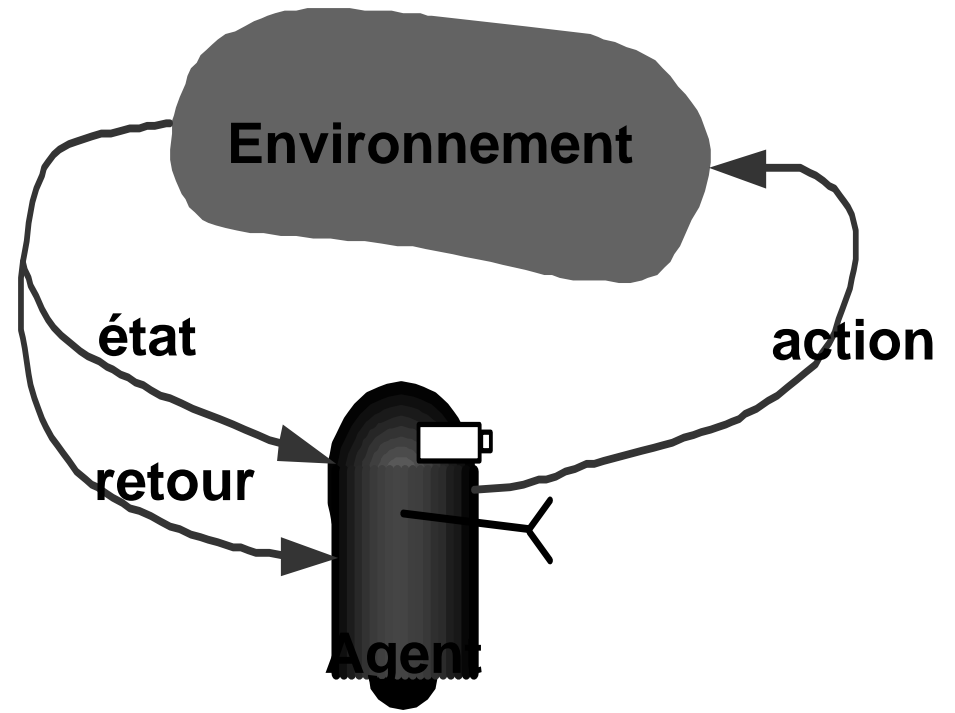
Logiciel ou  
matériel



Qu'est ce qu'un agent autonome peut apprendre, et comment, s'il agit dans un environnement *a priori* inconnu, sans que l'on puisse lui fournir d'aide (telle que des exemples de ce qu'il faut faire ou ne pas faire dans telle ou telle situation) ?

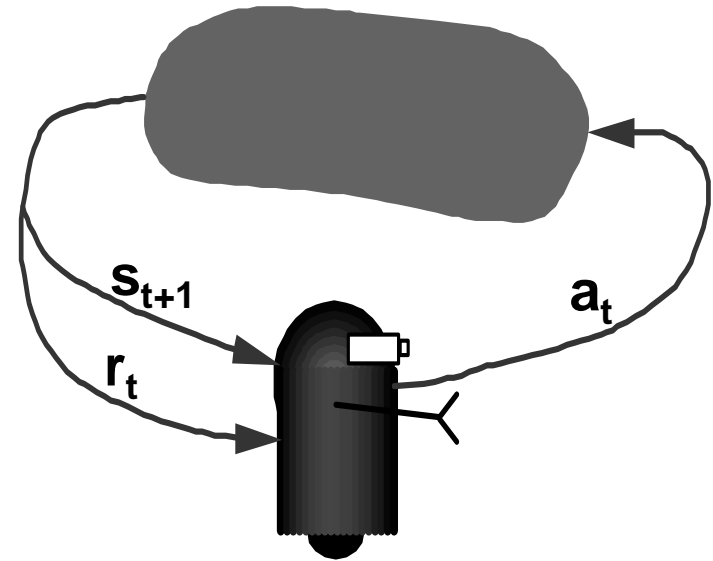
Seule information reçue : un « retour » lui donne une estimation  $\pm$  précise de son comportement.

# C'est quoi ? (3/14)



- un agent situé dans et en interaction avec son environnement
- il est dans un état perçu  $s_t \in S$  à l'instant  $t$
- ensemble d'actions possibles à l'instant  $t$  :  $A_t \subset A$
- l'émission de l'action  $a \in A_t$  entraîne :
  - un retour immédiat  $r_t$
  - le passage dans un état  $s_{t+1}$

# C'est quoi ? (4/14)



Remarques :

- l'état perçu peut être différent de l'état réel
- le retour immédiat reçu et l'état atteint suite à l'émission d'une certaine action dans un certain état peuvent varier au cours du temps
- $S$  peut varier au cours du temps ( $S_t$ )
- $A_t$  peut dépendre de  $s_t$
- le système ne reçoit jamais aucune information qui lui indiquerait :
  - quelle aurait été la meilleure action à effectuer dans un état donné
  - quel meilleur retour il aurait pu recevoir
- le retour (immédiat) perçu peut être la conséquence d'une action émise il y a longtemps
- les conséquences à court ou long terme des actions émises peuvent être contradictoires

# C'est quoi ? (5/14)

## Formalisons un peu :

- $S$  = ensemble des états de l'agent
- $A$  = ensemble des actions que l'agent peut émettre
- dans l'état  $s \in S$ , l'agent peut émettre les actions  $A_s \subset A$ .
- le temps est discrétisé :  $t \in \{0, \dots\}$  ;
- $P[s_{t+1}=s|t+1,s_t,s_{t-1},\dots,s_0,a_t,a_{t-1},\dots,a_0] \in [0, 1]$  : probabilité que l'état à l'étape  $t+1$  soit  $s$  si les états précédents ont été  $s_t,s_{t-1},\dots,s_0$  et que les actions dans chacun de ces états ont été respectivement  $a_t,a_{t-1},\dots,a_0$ .
- $R[s_{t+1}=s|t+1,s_t,s_{t-1},\dots,s_0,a_t,a_{t-1},\dots,a_0]$  réel : espérance de retour à l'étape  $t+1$  pour la transition vers l'état  $s_{t+1}$  si les états précédents ont été  $s_t,s_{t-1},\dots,s_0$  et que les actions dans chacun de ces états ont été respectivement  $a_t,a_{t-1},\dots,a_0$ .

# C'est quoi ? (6/14)

## Formalisons un peu (suite):

Remarques :

- S, A, P et R peuvent varier au cours du temps (non stationnaires) ;
- S et A peuvent être finis ou infinis ;
- l'effet d'une action sur l'environnement n'est pas forcément déterministe (d'où P et R) ;



# C'est quoi ? (7/14)

## les retours

- Le retour fournit une information quant à la qualité de l'action et des actions qui ont été effectuées jusqu'alors ;
- le retour peut être quelconque ; mais c'est très généralement un nombre
- le retour doit spécifier ce que l'on veut obtenir, pas comment l'obtenir
- en général, l'objectif de l'agent est de maximiser ses retours au cours de son fonctionnement :

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^T \gamma^k r_{t+k+1}$$

où  $\gamma \in [0, 1]$  est le facteur de dépréciation

Deux cas :

- la tâche a une fin : tâche épisodique/horizon fini :  $T$  est fini ; il existe des états terminaux
- la tâche n'a pas de fin : tâche en horizon infini :  $T = +\infty$  ; prendre  $\gamma < 1$  pour que  $R_t$  demeure fini.

# C'est quoi ? (8/14)

## les retours (suite)

- une action a des conséquences à court terme et à plus long terme ;
- aussi, on doit tenir compte des conséquences à court terme et des conséquences à long terme ;
- d'autres actions futures peuvent avoir également un effet sur les conséquences à long terme : aussi, il est intuitivement sain de mettre plus l'accent sur les conséquences à court terme que sur les conséquences à long terme , sans toutefois négliger ces dernières totalement ;
- plus  $\gamma$  s'approche de 1, plus l'agent prend en compte les conséquences à long terme de ses actions ;  $\gamma=0$  : agent myope.

# C'est quoi ? (9/14)

## Que cherche-t-on ?

→ une stratégie  $\pi$  qui spécifie l'action à émettre à un moment donné pour maximiser son espérance de retours

# C'est quoi ? (10/14)

## Formalisons un peu (suite):

Définitions :

- valeur d'un état :  $V^\pi(s)$  : espérance de retour  $R$  si l'agent se trouve dans l'état  $s$  et suit la stratégie  $\pi$ . La fonction  $V^\pi$  est dénommée la fonction valeur pour la stratégie  $\pi$  ;
- qualité d'une paire (état, action) :  $Q^\pi(s,a)$  : espérance de  $R$  si l'agent se trouve dans l'état  $s$  et émet l'action  $a$ , puis suit la stratégie  $\pi$ . La fonction  $Q^\pi$  est dénommée la fonction qualité pour la stratégie  $\pi$ .

Il est clair que  $V^\pi(s)$  et  $Q^\pi(s,a)$  représentent la même chose. À un moment donné, on a :

$$V^\pi(s) = \sum_a \pi(s,a) Q^\pi(s,a)$$

# C'est quoi ? (11/14)

## Formalisons un peu (suite):

Propriété de Markov :

un système est markovien si son état courant résume toute son histoire.

Dans un système markovien :

- $P[s_{t+1}=s|t+1,s_t,s_{t-1},\dots,s_0,a_t,a_{t-1},\dots,a_0] = P[s_{t+1}=s|s_t,a_t] = P_{s_{t+1},s_t}^{a_t}$   
probabilité d'atteindre l'état  $s'$  en partant de l'état  $s$  et en émettant l'action  $a$

- $R[s_{t+1}=s|t+1,s_t,s_{t-1},\dots,s_0,a_t,a_{t-1},\dots,a_0] = R[s_{t+1}=s|s_t,a_t] = R_{s_{t+1},s_t}^{a_t}$   
retour moyen si on atteint l'état  $s'$  en partant de l'état  $s$  et en émettant l'action  $a$

# C'est quoi ? (12/14)

## Formalisons un peu (suite):

Si  $S$  et  $A$  sont finis et que le système est markovien, on a un problème de décision de Markov fini (PDMF).

Dans ce cas, il existe une stratégie optimale  $\pi^*$ , une fonction valeur optimale  $V^*$  et une fonction qualité optimale  $Q^*$ .

Et on a alors une jolie équation définissant  $V^*$  par récurrence :

$$V^*(s) = \max_a \sum_{s'} P_{s,s'}^a [R_{s,s'}^a + \gamma V^*(s')]$$

(équation de Bellman, 1957)

On a une équation du même genre pour  $Q^*$  :

$$Q^*(s, a) = \sum_{s'} P_{s,s'}^a [R_{s,s'}^a + \gamma \max_{a'} Q^*(s', a')]$$

# C'est quoi ? (13/14)

## Formalisons un peu (suite):

Disposant de  $V^*$ , on en déduit aisément une stratégie déterministe  $\pi^*$  :

→ dans l'état  $s$ , associer une probabilité non nulle aux actions qui amènent dans un état suivant auquel est associé un maximum de l'équation de Bellman (et seulement à ces actions-là).

→ stratégie gloutonne par rapport à  $V^*$

→ principe des algos pour déterminer  $\pi$  : on calcule  $V^*$  et on en déduit  $\pi^*$

Question : faut-il calculer  $V^*$  pour tous les états avant de pouvoir calculer une bonne stratégie ?

**Réponse : heureusement, non**

# C'est quoi ? (14/14)

Exemples d'application :

Tâches épisodiques :

- jeu constitué de parties (dames, échecs, tarot, ...)
- robot devant accomplir une certaine tâche dans une usine :
  - saisir une pièce,
  - fixer une pièce sur une autre,
  - ramasser des boîtes de boisson vides dans des bureaux,
  - peindre une carrosserie, ...

Tâches non épisodiques :

- contrôleur de systèmes temps réel à longue durée de vie :
  - ascenseurs
  - robot aspirateur
  - robot explorateur
  - ...



# Comment on fait ? (1/)

Face à un problème à résoudre :

1. le mettre sous la forme d'un problème de renforcement
2. appliquer un algorithme de résolution :
  - 2 grandes approches :
    - si on connaît S, A (S et A finis pas trop grands), P et R : programmation dynamique
    - sinon : apprentissage par renforcement : méthodes basées sur la différence temporelle (TD)

# Comment on fait ? (2/)

## Méthodes TD

*(temporal difference learning)*

### Principe :

algorithme itératif d'apprentissage par interaction avec l'environnement :

- à chaque itération, étant dans un certain état  $s$ , l'apprenant agit sur son environnement par l'émission d'une action  $a$  de laquelle il attend un certain retour ;
- un retour (immédiat) est perçu ;
- la différence entre le retour perçu et le retour attendu est utilisée pour modifier ses attentes (*i.e.* l'estimation de la qualité  $Q(s,a)$  ou de la valeur  $V(s)$ ).

Petit à petit, commençant par émettre un comportement aléatoire, l'apprenant apprend quelle action doit être émise dans les différents états.

# Comment on fait ? (3/)

## Méthodes TD

Deux composants essentiels d'un l'algorithme TD :

- sélection d'une action à émettre ;
- mise à jour des attentes.

# Comment on fait ? (4/)

## Méthodes TD

### Sélection de l'action :

étant dans l'état  $s$ , il faut choisir une action à émettre parmi  $A_s \subset A$

On suppose que l'algorithme dispose d'une estimation de la qualité de chaque paire  $(s, a)$  :  $Q(s, a)$ .

Dans ce cas, on peut déterminer la meilleure action :  $\arg \max_{a \in A_s} Q(s, a)$

celle que l'algorithme estime, pour l'instant, comme étant celle qui rapporte la plus grande quantité de retour, donc l'action dont la qualité est la plus grande dans l'état courant.

# Comment on fait ? (5/)

## Méthodes TD

### Mise à jour des attentes :

Attente = retour attendu dans un état donné  $s$  si l'algorithme émet une certaine action  $a$  :  $Q(s,a)$ .

Après l'émission de l'action  $a$  dans l'état  $s$ , l'algorithme reçoit un retour  $r$ .

Rappel :  $Q(s,a)$  = somme pondérée (par  $\gamma$ ) des retours à percevoir dans le futur si l'action  $a$  est émise dans l'état  $s$ .

Donc, on peut produire une nouvelle estimation de  $Q(s,a)$  :

$$Q_{\text{nouvelle valeur}}(s,a) = r + \gamma \max_{a'} Q(s',a')$$

où  $s'$  est l'état atteint effectivement après l'émission de  $a$  dans l'état  $s$ .

# Comment on fait ? (6/)

## Méthodes TD

Structure de données pour représenter cette quantité ?

Nombreuses possibilités :

- la plus simple : une table  $Q[s][a]$  : qualité de la paire (s, a) ;

Problème : si  $|S|$  est grand, cette table est immense !

Solutions plus compactes :

- réseau de neurones : on place (s,a) en entrée, il sort  $Q(s,a)$  ;
- polynôme ;
- arbre de décision ;
- ...

# Comment on fait ? (7/)

## Méthodes TD

### Q-Learning tabulaire [Watkins, 1989] :

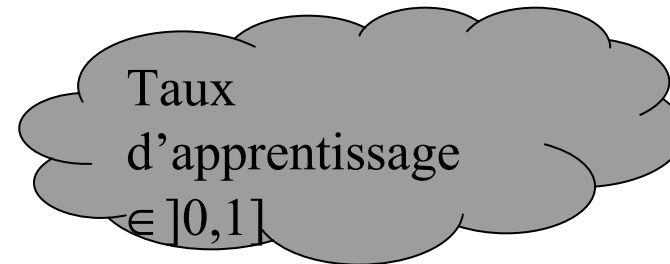
- Initialiser les  $Q(s,a)$  arbitrairement
- **Répéter**
  - $t \leftarrow 0$
  - Initialiser l'état initial :  $s_t$
  - **Répéter** // Effectuer un épisode
    - sélectionner l'action à émettre dans l'état  $s_t$  :  $a_t$
    - émettre cette action
    - observer le retour  $r_t$  et le nouvel état  $s_{t+1}$
    - mettre à jour  $Q(s,a)$  :
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$$
    - $t \leftarrow t+1$
  - **Jusqu'à** ce que  $s_t$  soit un état terminal
- **Jusque** condition d'arrêt remplie

# Comment on fait ? (8/)

## Méthodes TD

Q-Learning tabulaire :

la **mise à jour** de  $Q(s,a)$  :



Estimation  
précédente

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \underbrace{\alpha}_{\text{correction}} \underbrace{[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]}_{\text{Différence entre estimation courante et nouvelle estimation}}$$

Différence entre estimation courante et nouvelle estimation  
Nouvelle estimation des retours à venir.

correction à apporter à  $Q(s,a)$  pour améliorer cette estimation



# Comment on fait ? (9/)

## Méthodes TD

### Sélection de l'action à émettre :

- $\epsilon$ -gloutonne : soit  $\epsilon \in [0,1]$  :
  - prendre  $\arg \max_a Q(s_t, a)$  avec probabilité  $\epsilon$
  - prendre une action au hasard avec probabilité  $1-\epsilon$ .

Encore mieux : faire varier  $\epsilon$  au cours des itérations :  $\epsilon = 1/t$  par exemple

- *softmax* :
  - déterminer pour chaque action une probabilité qu'elle soit émise :
    - proportionnelle à sa qualité  $Q$
    - ou selon une distribution de Boltzmann (prop. à  $e^{Q/\tau}$ )
  - choisir l'action à émettre en fonction de cette probabilité

# Comment on fait ? (10/)

## Méthodes TD

Propriété de convergence :

pour un problème de décision markovien fini, Q-learning converge vers  $Q^*$  si :

- toutes les paires  $(s,a)$  sont visitées une infinité de fois ;
- $\alpha_v(s,a) \in [0, 1[$

- $\sum_{v=1}^{\infty} \alpha_v(s, a) = \infty$

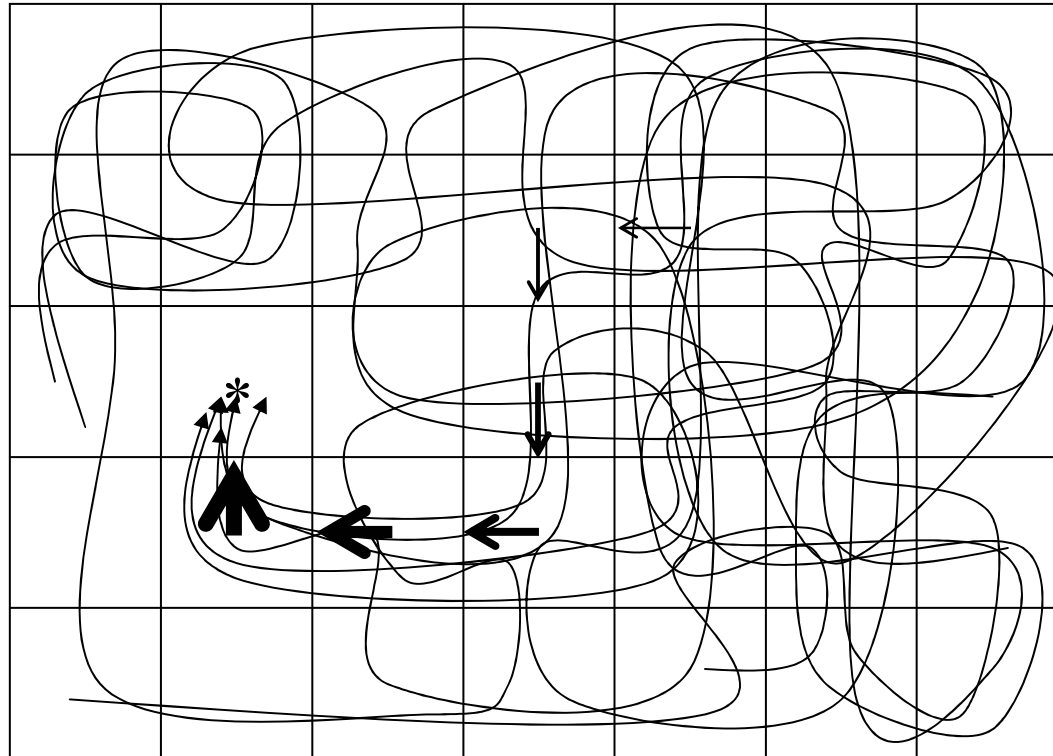
- $\sum_{v=1}^{\infty} [\alpha_v(s, a)]^2 < \infty$

où  $\alpha_v(s,a)$  est la  $v^e$  visite de la paire  $(s,a)$

# Comment on fait ? (11/)

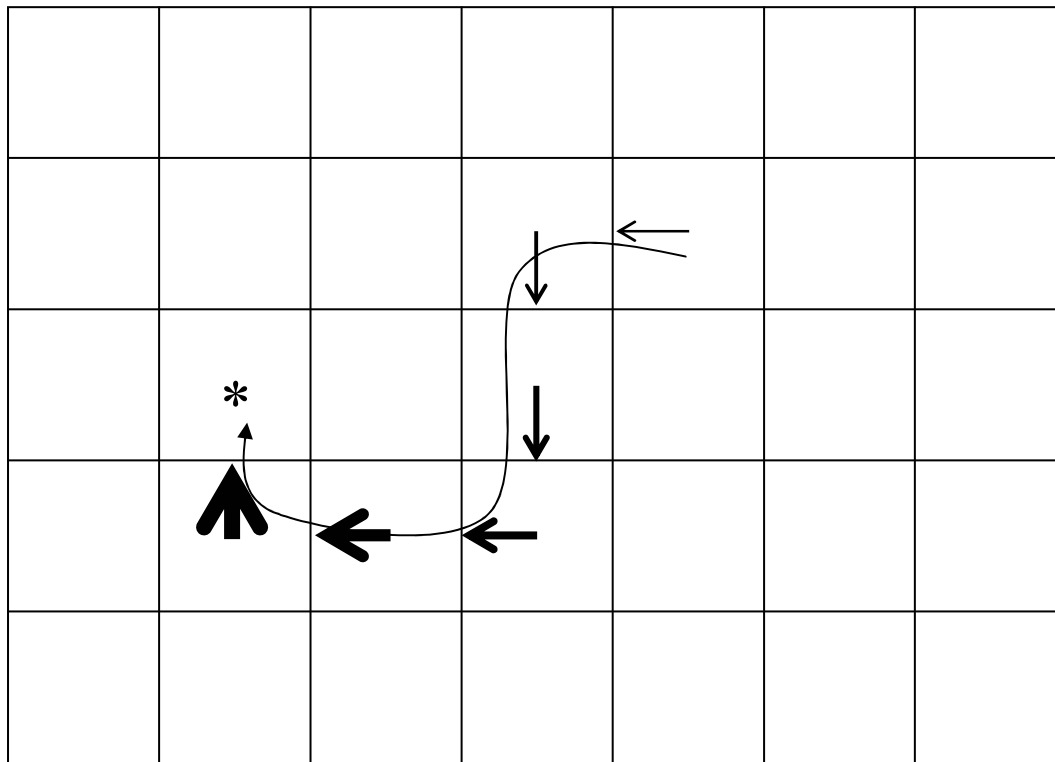
## Méthodes TD

Remarque :  
on suppose que  
 $Q(s,a)=0$  à  $t=0$



# Comment on fait ? (12/)

## Méthodes TD



Intérêt :

apprentissage plus rapide :  
en un seul épisode, on  
apprend une estimation pour  
plusieurs paires (s,a)

Remarque :

plus on s'éloigne de l'état  
terminal, plus la mise à jour  
de  $Q(s,a)$  diminue ;  
elle diminue d'un facteur  $\lambda$

# Comment on fait ? (13/)

## Méthodes TD

Mécanisme de mémorisation des paires  $(s,a)$  parcourues durant l'épisode :  
trace d'éligibilité (*eligibility trace*)

$e(s,a)$  :

- initialisée à 0 pour toutes les paires  $(s,a)$
- à chaque visite à  $(s,a)$  durant l'épisode :  $e(s,a) \leftarrow e(s,a) + 1$
- à chaque mise à jour d'une qualité  $(Q(s_t, a_t))$ , on met à jour en même temps toutes les qualités pour les paires  $(s,a)$  dont  $e(s,a) \neq 0$  et  $e(s,a)$  décroît d'un facteur  $\lambda$ .

→ Algorithme  $Q(\lambda)$

# Comment on fait ? (14/)

## Méthodes TD

### $Q(\lambda)$ tabulaire :

- Initialiser les  $Q(s,a)$  arbitrairement
- **Répéter**
  - $t \leftarrow 0$
  - $e(s,a) \leftarrow 0$
  - Initialiser l'état initial :  $s_t$
  - choisir l'action  $a_t$
  - **Répéter** // Effectuer un épisode
    - émettre l'action  $a_t$
    - observer le retour  $r_t$  et le nouvel état  $s_{t+1}$
    - choisir  $a_{t+1}$  en fonction de  $s_{t+1}$
    - $a^* \leftarrow \arg \max_b Q(s_{t+1}, b)$
    - $\delta \leftarrow r_t + \gamma Q(s_{t+1}, a^*) - Q(s_t, a_t)$
    - $e(s_t, a_t) \leftarrow e(s_t, a_t) + 1$
    - **Pour** toutes les paires  $(s,a)$  **Faire**
      - $Q(s,a) \leftarrow Q(s,a) + \alpha \delta e(s,a)$
      - $e(s,a) \leftarrow \gamma \lambda e(s,a)$
  - **Jusqu'à** ce que  $s_t$  soit un état terminal
- **Jusque** condition d'arrêt remplie
- **Fait**
  - $t \leftarrow t + 1$


# Comment on fait ? (15/)

## Méthodes TD

$Q(\lambda)$  :

Il y a plusieurs versions de cet algorithme :

- la précédente est qualifiée de naïve ;
- celle proposée originalement par [Watkins, 1989] est :
- une autre version a été proposée par [Peng & Williams, 1994].

- 
- émettre l'action  $a_t$
  - observer le retour  $r_t$  et le nouvel état  $s_{t+1}$
  - choisir  $a_{t+1}$  en fonction de  $s_{t+1}$
  - $a^* \leftarrow \arg \max_b Q(s_{t+1}, b)$
  - $\delta \leftarrow r_t + \gamma Q(s_{t+1}, a^*) - Q(s_t, a_t)$
  - $e(s_t, a_t) \leftarrow e(s_t, a_t) + 1$
  - **Pour** toutes les paires  $(s, a)$  **Faire**
    - $Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$
    - **Si**  $a_{t+1} = a^*$  **Alors**  $e(s, a) \leftarrow \gamma \lambda e(s, a)$
    - **Sinon**  $e(s, a) \leftarrow 0$
  - **Fait**
  - $t \leftarrow t + 1$

# Comment on fait ? (16/)

## Méthodes TD

Généralisation de l'apprentissage ?

Très faible ; quelques propositions pour l'améliorer :

idée : quand on met à jour la qualité, on met en même temps à jour la qualité d'autres paires état,action.

Autre approche : utiliser une structure de données qui généralise mieux qu'une table :

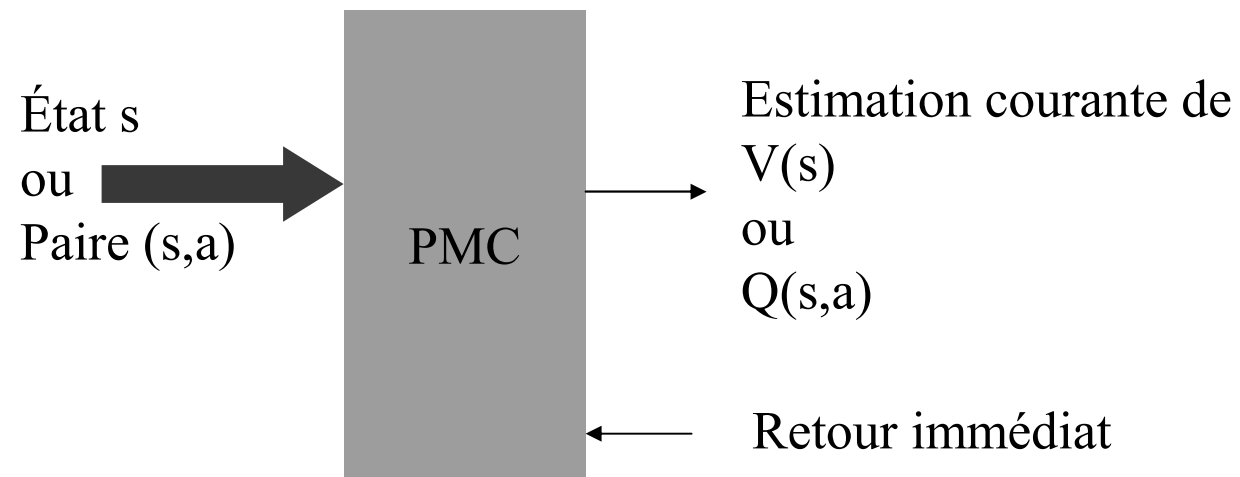
- réseau de neurones ;
- arbre de décision ;
- polynôme ;
- ...



# Comment on fait ? (17/)

## Méthodes TD

Algorithme TD utilisant un réseau de neurones (PMC) pour stocker  $Q(s,a)$  :



# Comment on fait ? (18/)

## Méthodes TD

Les poids des connexions du réseau sont responsables de la valeur en sortie du réseau.  
→ mettre à jour  $Q$  revient à modifier les poids pour que la sortie (estimation de  $Q$ ) prédise mieux le retour effectivement reçu (exactement le même principe que dans la version tabulaire).

Mise à jour des connexions : rétro-propagation du gradient.

Pour la sélection de l'action, c'est exactement comme pour la version tabulaire : on place en entrée les différentes actions possibles pour l'état courant ; on regarde la valeur en sortie (estimation de  $Q(s,a)$ ). On sélectionne la meilleure avec une probabilité  $\epsilon$ .

# Comment on fait ? (19/)

## Méthodes TD

Intérêt d'utiliser un réseau de neurones :

quand on met à jour les poids pour mettre à jour  $Q$ , toutes les estimations de  $Q$  sont modifiées en même temps (et non pas seulement celle pour la paire qui vient d'être visitée dans le Q-learning tabulaire ou les dernières paires visitées dans le  $Q(\lambda)$  tabulaire).

→ généralisation de l'apprentissage importante.

Outre le PMC, les réseaux de Kohonen ont été utilisés.

# Méthodes TD (20/20)

## Quelques sujets chauds

- Apprentissage de modèle de l'environnement :  
(Dyna, Dyna-Q, prioritized sweeping, ...)
- Dans un contexte continu :  
discrétisation de l'espace ; différentes approches pour discrétiser utilement  
(partigame et successeurs : R. Munos par ex.)
- Qu'est ce qu'un état ?  
apprentissage d'états (A. Dutech par ex.)
- Environnement non markovien (POMDP)  
Idée : transformer un problème non markovien en une succession de problèmes markovien  
approche hiérarchique (Dietterich, Wiering, ...)
- Systèmes multi-agents apprenant par renforcement  
N agents coopérant apprennent-ils mieux qu'un seul à résoudre une tâche donnée
- Apprentissage hybride  
Combinaison apprentissage par renforcement et apprentissage avec des exemples

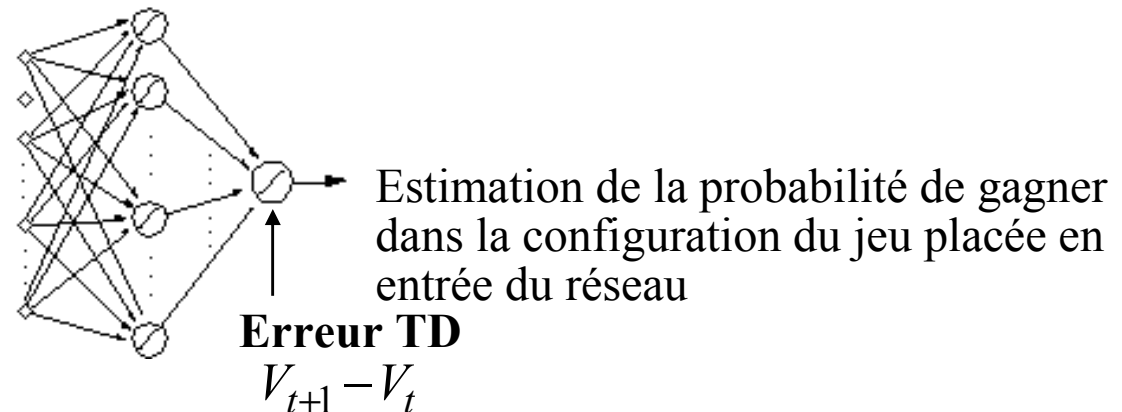
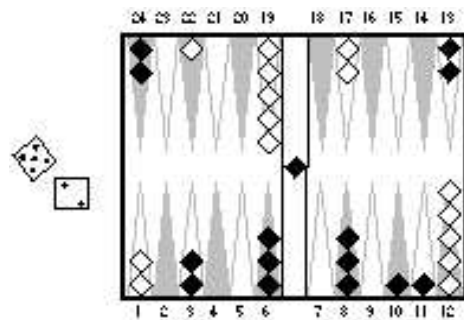
# A quoi ça sert ? (1/6)

- Contextes dans lesquels on ne sait pas très bien ce qu'il faut faire :
  - On ne peut pas donner d'exemples de ce qu'il faut faire, mais on est capable de juger si telle option est plus ou moins correcte.
- Quelques applications :
  - Jeux : TD-Gammon, KnightCap (échecs)
  - Contrôle d'ascenseurs
  - Contrôle de robot mobile
  - Gestion de la réservation de places d'avions
  - Allocation dynamique de canaux téléphone cellulaire
  - Ordonnancement de tâches
  - Vision
  - Système d'aide à l'apprentissage intelligent (ITS)
  - Jeux vidéo
  - Modèle du comportement animal

# A quoi ça sert ? (2/6)

## TD-Gammon

Tesauro, 1992–1995



Initialement : réseau aléatoire

Joue de très nombreuses parties contre lui-même et apprend ainsi une fonction valeur

(1,5x10<sup>6</sup> parties pour la version 3.0, 80 neurones cachés)

Réseau (PMC) ayant : 198 entrées et 40 à 80 neurones cachés.

Versions récentes combinent TD avec un minimax (peu profond).

# A quoi ça sert ? (3/6)

## Le jeu de « checkers »

Samuel, 1959 :

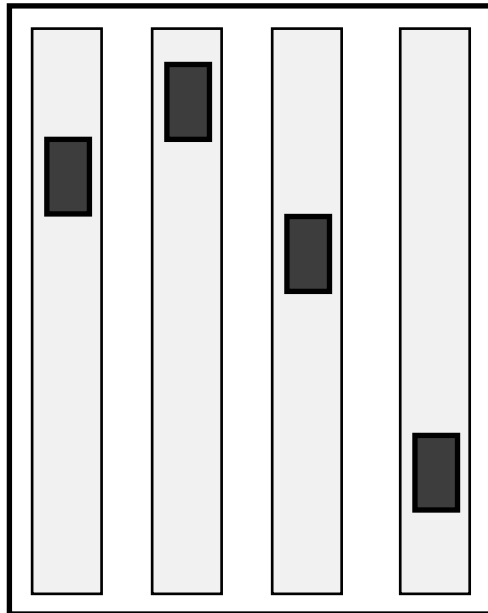
- logiciel apprenant à jouer aux checkers.
- essaie d'associer à la configuration de jeu courante sa valeur.

# A quoi ça sert ? (4/6)

## Contrôle d'ascenseurs

Crites and Barto, 1996

10 étages, 4 ascenseurs couplés



ETAT : état des boutons ;  
position, direction et état de  
déplacement des ascenseurs  
; nombre de passagers dans  
les ascenseurs et en attente

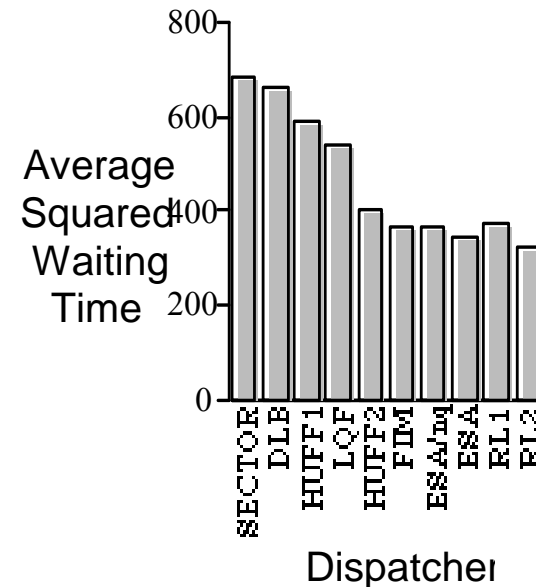
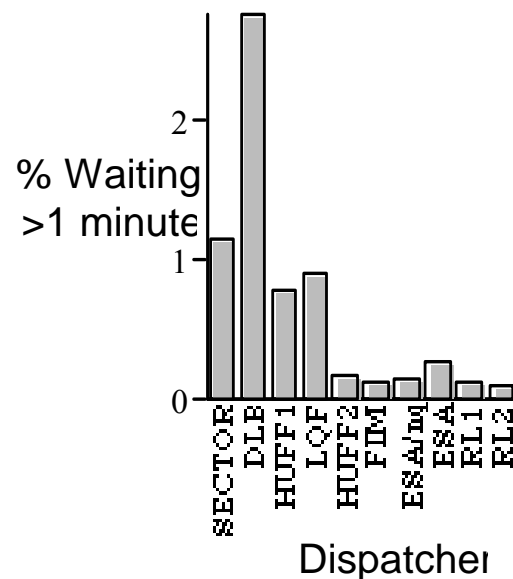
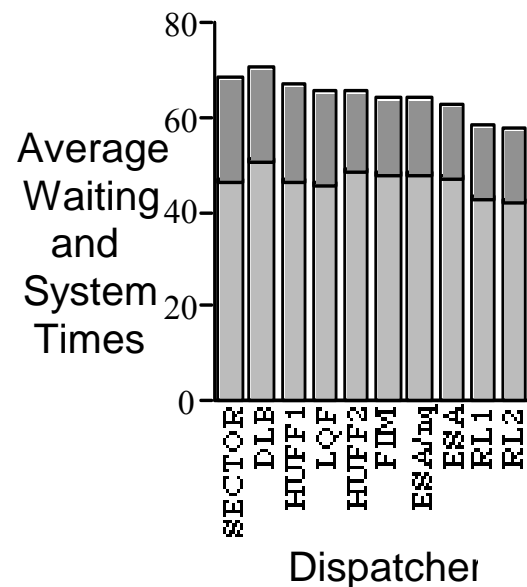
ACTIONS: arrêter à ou passer  
l'étage suivant, ; quand  
arrêté, monter ou descendre

RETOURS: -1 par pas  
d'attente par personne en  
train d'attendre



# A quoi ça sert ? (5/6)

## Contrôle d'ascenseurs



Réseau de neurones composé de 47 entrées, 20 neurones cachés, 1 ou 2 sorties.

# A quoi ça sert ? (6/6)

## Contrôle de robots mobiles

Navigation dans des environnements a priori inconnu.

Voir par exemple :

- travaux de C. Touzet
- projet MARS au MIT (<http://www.ai.mit.edu/people/lpk/mars/>)
  - Combinaison apprentissage par renforcement et supervisé
- robotique collective

# La communauté

- En cours de développement
- Peu développée en Europe
- Groupe PDMIA (<http://www.loria.fr/~buffet/pdm-et-ia>)
- Travaux sur les robots autonomes (logiciel ou matériel)

# Conclusion (1/2)

- L'apprentissage par renforcement peut trouver sa place dans de très nombreuses situations réelles qui sont peu ou mal formalisées
- Proximité avec AG et algo. en essaim qui sont également des algorithmes reposant sur des notions d'essai/erreur et de retour (fitness, phéromone)
- Liens forts avec la programmation dynamique  
(cf. prog. dyn. asynchrone et la prog. dyn. temps réel)

# Conclusion (2/2)

L'apprentissage par renforcement est un peu dans la situation des AG il y a encore peu !

- mal connu en dehors de son cercle
- apprendre à maîtriser ce type d'apprentissage
- de très nombreuses questions plus ou moins techniques restent à étudier :
  - paramètres, problèmes de représentation, fonction de renforcement, ...
  - méthodologie : comment formuler un problème sous la forme qu'il faut pour qu'il soit traité au mieux par ce genre d'algos ?
  - préciser sa niche
  - étudier l'hybridation de ce type d'apprentissage avec d'autres, voire avec d'autres algorithmes de recherche (pas nécessairement d'apprentissage)
  - formaliser ses performances (convergence, apprenabilité, dim. VC ?, ...)
  - adapter des techniques de l'apprentissage supervisé à l'apprentissage par renforcement (*boosting*, ...)
  - ...

# Conclusion

- L'apprentissage par renforcement peut trouver sa place dans de très nombreuses situations réelles qui sont peu ou mal formalisées
- Proximité avec AG et algo. en essaim qui sont également des algorithmes reposant sur des notions d'essai/erreur et de retour (fitness, phéromone)
- Liens forts avec la programmation dynamique  
(cf. prog. dyn. asynchrone et la prog. dyn. temps réel)

# Où démarrer pour en savoir plus ?

- Dépôt RL : <http://www-anw.cs.umass.edu/rlr>
- Tutoriels sur le *web* :
  - Moore, Littman, Kaelbling :  
<http://www.cs.washington.edu/research/jair/volume4/kaelbling96a-html/rl-survey.html>
  - C. Touzet :  
[http://saturn.epm.ornl.gov/~touzetc/Publi/Bq\\_Jutten.pdf](http://saturn.epm.ornl.gov/~touzetc/Publi/Bq_Jutten.pdf)
  - Sur les POMDP :  
<http://www.cs.brown.edu/research/ai/pomdp/tutorial/index.html>
- Des livres :
  - Sutton, Barto, *Reinforcement Learning*, MIT Press, 1998
  - Bertsekas, Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, 1996
  - Bertsekas, *Dynamic Programming and Optimal Control*, Athena Scientific, 2000 (2 vol.)
  - Les actes des conférences ICML, ECML et SAB.
  - Les revues Machine Learning, Adaptive Behavior, JAIR, JMLR