

COURS SUR
L'APPRENTISSAGE ARTIFICIEL
COURS MASTER IFI 2010/2011



JEAN-DANIEL ZUCKER

DR À L'IRD UR GEODES
 (MODÉLISATION MATHÉMATIQUES ET INFORMATIQUES DES SYSTÈMES COMPLEXES)
 UMMISCO UMI 209



COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

Administratif: 1/2 Module Apprentissage (18ECTS²)

- **Séance 1: Jeudi 25 Novembre – INTRO GÉNÉRALE**
 - Introduction, principe inductif, historique, formulation
 - Quelques mots sur l'apprentissage statistique
 - Espace des versions et algorithme
- **Séance 2: Lundi 6 Décembre – APPRENTISSAGE SUPERVISÉ**
- **Séance 3: Mardi 7 Décembre – APPRENTISSAGE NON-SUPERVISÉ**
- **Séance 4: Mardi 11 Janvier 2011 – ALGORITHMES ÉVOLUTIONNAIRES**
- **Séance 5: Vendredi 14 Janvier 2011 – ALGO. PAR RENFORCEMENT**
- **Séance 6: Lundi 17 Janvier 2011 – MINI-PROJET**

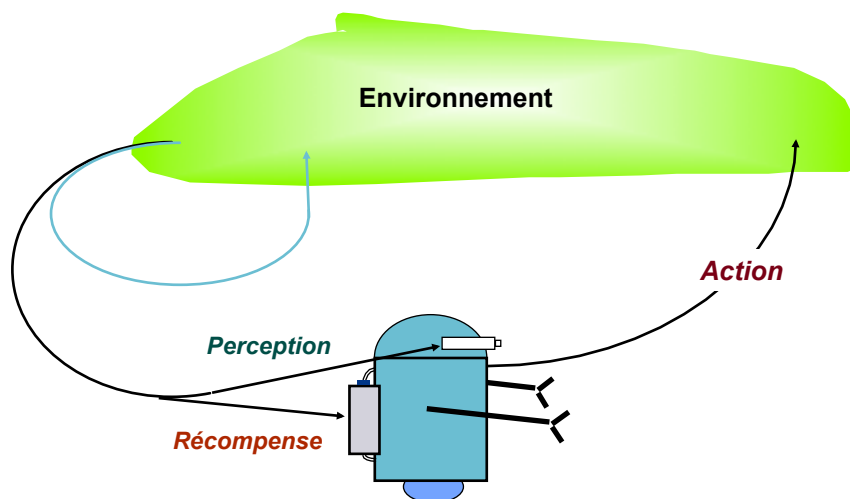
COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

Plan du cours

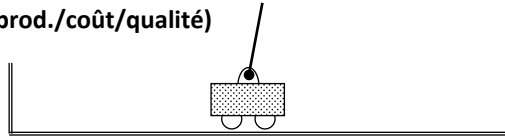
- 1- **Introduction** : motivation, problèmes, notions et principes
- 2- **La notion d'utilité**
- 3- **Apprentissage des fonctions d'utilité en environnement connu**
- 4- **Univers inconnu : méthodes de différences temporelles**
 - Principes
 - Méthode du Q-Learning
 - Extension à plusieurs pas : le TD-Learning
- 5- **La généralisation dans l'apprentissage par renforcement**
- 7- **Exemples d'applications**
- 8- **Bilan et perspectives**

1.1 Introduction : schéma général

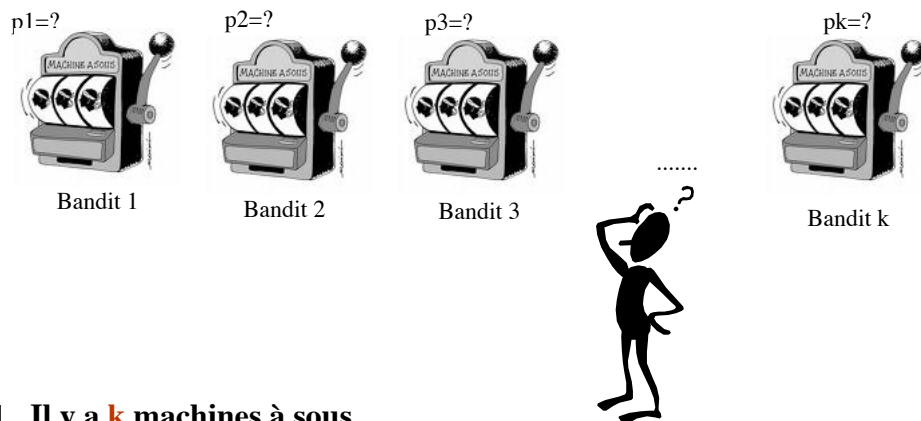


I.2. Exemples illustratifs

- ✓ Les jeux (dans certains cas): jugements intuitifs, blitz...
- ✓ A la naissance, une gazelle tient à peine debout...
- ✓ Attraper son paquet de cérééal favori, un ballon,...
- ✓ Stratégie d'un robot (se recharger)/(continuer)
- ✓ Contrôleurs adaptatifs temps réel (prod./coût/qualité)
- ✓ Equilibre
- ✓ Point communs:
 - ✓ interactions, sous-buts, incertitude de l'environnement.
 - ✓ l'expérience permet d'apprendre...



Exemple 1: Le problème du "bandit à k bras"



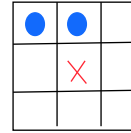
- Il y a k machines à sous
- Chacune donne 0 ou 1€ avec une loi de probabilité cachée
- On peut jouer h coups.
- Comment choisir les machines pour optimiser le gain?

Exemple 2: Le jeu de Tic-Tac-Toe

7

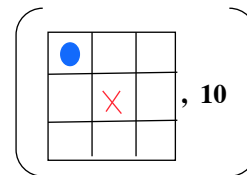
Soit

- Un TicTacToe à 9 cases
- Comment apprendre à évaluer une position ?



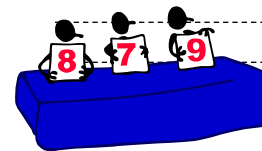
Apprentissage Supervisé:

- Un ensemble de couple (positions notes):



Apprentissage par renforcement:

- On apprend la valeur des positions en fonction des parties jouées.

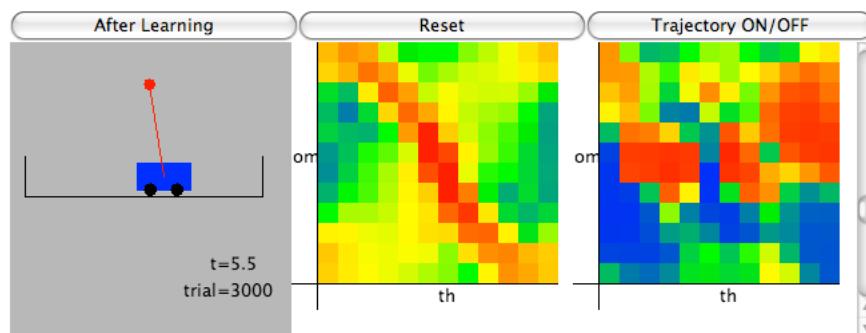


COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

Exemple 3 : le contrôle de l'équilibre

8



From left: State of the pendulum / Critic / Actor
<http://brain.cc.kogakuin.ac.jp/~kanamaru/NN/CPRL/>

<http://www.bovine.net/~jlawson/hmc/pole/sane.html>

COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

Autres Applications

- **Computer games** (Schaeffer et al. 01)
- **Robotics** (Kohl and Stone 04)
- **Marketing** (Abe et al 04)
- **Power plant control** (Stephan et al. 00)
- **Bio-reactors** (Kaisare 05)
- **Vehicle Routing** (Proper and Tadepalli 06)
- **Allocation de fréquences pour téléphone...**

Dès que l'on veut optimiser une séquence de décisions

Quand doit-on faire appel à l'app. par renf. ?

- ✓ Une tâche en plusieurs étapes où la récompense ne vient qu'à la fin d'une succession de choix (un état final)
e.g. Recherche dans un labyrinthe
- ✓ La récompense peut venir plus fréquemment (perdre une pièce aux échec) mais celle-ci ne donne pas d'indication sur la solution optimale
e.g. Prise de pièces (attention un sacrifice peut mener à la victoire)
- ✓ On ne sait pas quelle récompense attribuer à quelle action
credit assignment problem

1.2 Introduction : Les notations de base

- **Temps discret:** t
- **États :** $s_t \in \mathcal{S}$
- **Actions :** $a_t \in \mathcal{A}(s_t)$
- **Récompenses :** $r_t \in \mathcal{R}(s_t)$
- **L'agent :** $s_t \rightarrow a_t$
- **L'environnement :** $(s_t, a_t) \rightarrow s_{t+1}, r_{t+1}$
- **Politique :** $\pi_t : \mathcal{S} \rightarrow \mathcal{A} \quad T, R$
 - Avec $\pi_t(s, a) = \text{Prob que } a_t = a \text{ si } s_t = s$
- Les transitions et récompenses ne dépendent que de l'état et de l'action précédents : **processus Markovien**

Processus de décision Markovien

- **Propriété de Markov**

$$P(s_t | s_{t-1}, a_{t-1}) = P(s_t | s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}, \dots)$$

- **Alors**

$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) \max_{a' \in \mathcal{A}} Q(s', a')$$

récompense immédiate taux d'intérêt prochain état espéré Valeur future

1.2 Introduction : Eléments de base

13

- **Politique** : π

ensemble d'associations *situation* \rightarrow *action* (une application)

Une simple table ... un algorithme de recherche intensive

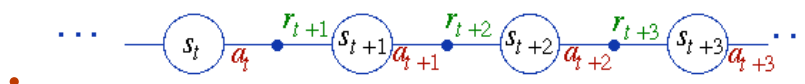
Eventuellement stochastique

- **Fonction de renforcement** :

- Définit implicitement le but poursuivi

- Une fonction : $(\text{état}, \text{action}) \rightarrow \text{récompense} \in \mathbb{R}$

- **Fonction d'évaluation** $V(s)$ ou $Q(s,a)$: gain cumulé



- Fonctions T et R : $(\text{état}(t), \text{action}) \rightarrow (\text{état}(t+1), \text{récompense})$

COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

2- La notion d'utilité

14

Principe :

- Choisir une action sans avoir besoin de faire une exploration (simulée) en avant

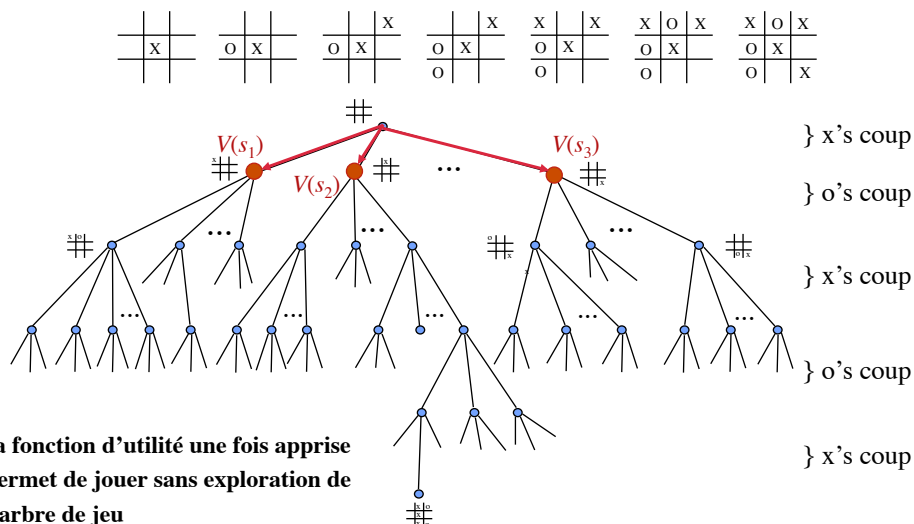
- Il faut donc disposer d'une fonction d'évaluation locale résumant une espérance de gain si l'on choisit cette action : *fonction d'utilité*

- Il faut apprendre cette fonction d'utilité : *apprentissage par renforcement*

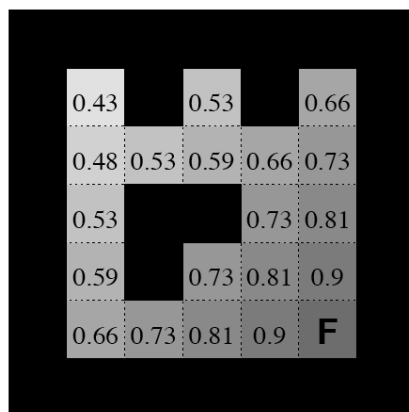
COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

2- Notion d'utilité. Exemple : Tic-Tac-Toe



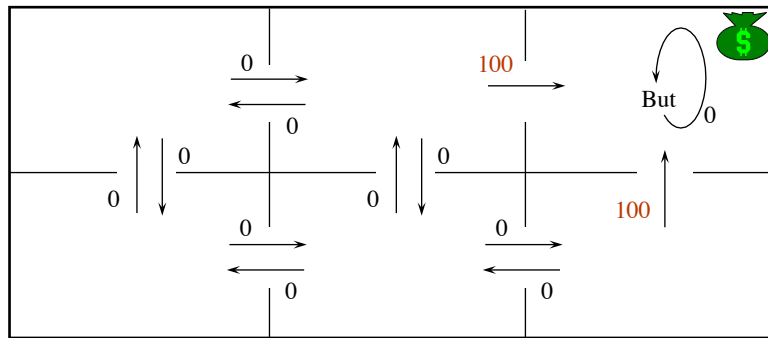
Notion d'utilité



$$\bar{V}^{\pi}(s) = E \{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots | s_t = s, \pi \}$$

Modélisation: états, actions et récompenses

$r(s,a)$ récompense immédiate (inconnue au départ)



➤ Dernière étape qui assure la récompense (jeux, monde des blocs, etc.)

➤ Tâche: apprendre la meilleure stratégie qui maximise le gain

Critères de gains

➤ Horizon fini

$$\sum_{t=0}^k r_t = r_0 + r_1 + \dots + r_k$$

➤ Horizon infini avec intérêt

$$\sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \dots$$

➤ En moyenne

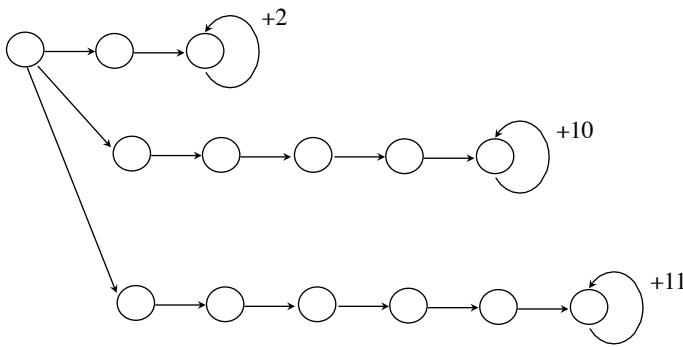
$$\lim_{k \rightarrow \infty} \frac{1}{k} \sum_{t=0}^k r_t$$

Comparaison des comportements

19

$k=4, \gamma=0.9$

Quelle est la meilleure stratégie



$$\sum_{t=0}^k r_t \quad \sum_{t=0}^{\infty} \gamma^t r_t \quad \lim_{k \rightarrow \infty} \frac{1}{k} \sum_{t=0}^k r_t$$

6 16.0 2

0 **59.0** 10

0 58.4 **11**

COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

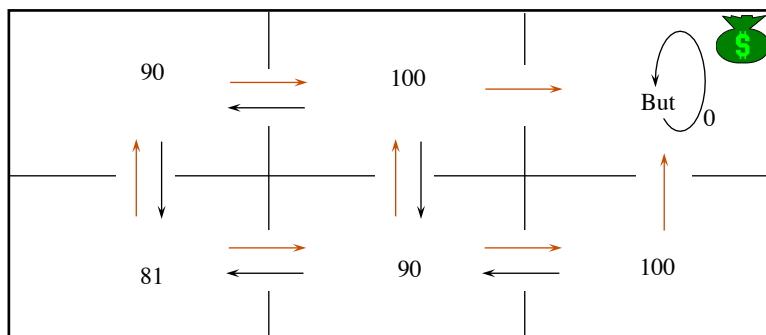
IFI 2011

Récompense Cumulée

20

► On définit la récompense cumulée $V^{\pi}(s_t) = \sum_{t=0}^{\infty} \gamma^t r_t$

► Le problème: trouver $\pi^* = \operatorname{argmax}_{\pi} (V^{\pi}(s))$



$V^*(s) = V^{\pi^*}(s)$ récompense cumulée optimale

COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

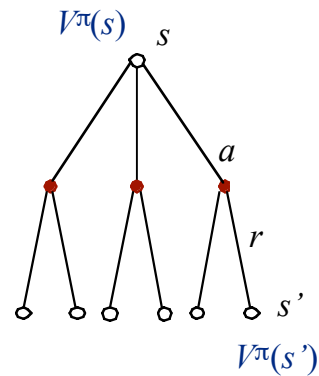
IFI 2011

2- Fonctions d'utilité : $V^\pi(s)$ et $Q^\pi(s,a)$

$$\begin{aligned} V^\pi(s) &= E_\pi \left\{ R_t \mid s_t = s \right\} \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}_{ss'}^a \left[\mathcal{R}_{ss'}^a + \gamma V^\pi(s') \right] \end{aligned}$$

$$\begin{aligned} Q^\pi(s,a) &= E_\pi \left\{ R_t \mid s_t = s, a_t = a \right\} \\ &= \sum_{s'} \mathcal{P}_{ss'}^a \left[\mathcal{R}_{ss'}^a + \gamma V^\pi(s') \right] \end{aligned}$$

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



2. Utilisation : avec la fonction d'utilité $V^*(s)$

- Une **politique** est une application $\pi : S \rightarrow A$

- **Valeur optimale d'un état :**

$$V^*(s) = \max_{\pi} V_{\pi}(s) = \max_{\pi} E_{\pi} \left(\sum_{t=0}^{\infty} \gamma^t r^t \right)$$

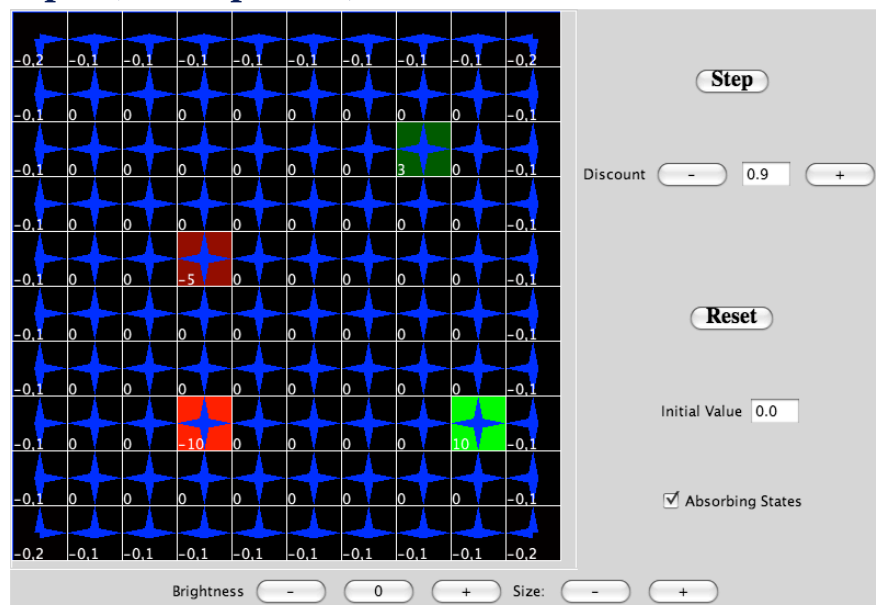
- **La fonction de valeur optimale V^* est unique**

$$V^*(s) = \max_{a \in \mathcal{Z}} \sum_{s'} \mathcal{P}_{ss'}^a \left[\mathcal{R}_{ss'}^a + \gamma V^*(s') \right]$$

- **Une politique stationnaire optimale existe :**

$$\pi^*(s) = a^* = \operatorname{ArgMax}_{a \in \mathcal{Z}} \sum_{s'} \mathcal{P}_{ss'}^a \left[\mathcal{R}_{ss'}^a + \gamma V^*(s') \right]$$

Exemple (récompenses)



COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

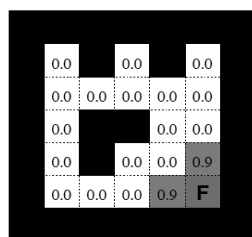
IFI 2011

3-1. Algorithme d'itération de valeur

```

initialize  $V(s)$  arbitrarily
loop until policy good enough
  loop for  $s \in \mathcal{S}$ 
    loop for  $a \in \mathcal{A}$ 
       $Q(s, a) := R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V(s')$ 
       $V(s) := \max_a Q(s, a)$ 
    end loop
  end loop
end loop

```



Initialize v arbitrarily, e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$

```

Repeat
   $\Delta \leftarrow 0$ 
  For each  $s \in \mathcal{S}$ :
     $v \leftarrow V(s)$ 
     $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$ 
     $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
  until  $\Delta < \theta$  (a small positive number)

```

Output a deterministic policy, π , such that:
 $\pi(s) = \arg \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$

COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

Algorithme Value Itération (Cas général)

```

initialize  $V(s)$  arbitrarily

loop until policy good enough

    loop for  $s \in \mathcal{S}$ 

        loop for  $a \in \mathcal{A}$ 

            
$$Q(s, a) := R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V(s')$$


            
$$V(s) := \max_a Q(s, a)$$


        end loop

    end loop

end loop

```

Algorithme Value Itération (Cas particulier)

```

initialize  $V(s)$  arbitrarily

loop until policy good enough

    loop for  $s \in \mathcal{S}$ 

        loop for  $a \in \mathcal{A}$ 

            
$$Q(s, a) := R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V(s')$$


            
$$V(s) := \max_a Q(s, a)$$


        end loop

    end loop

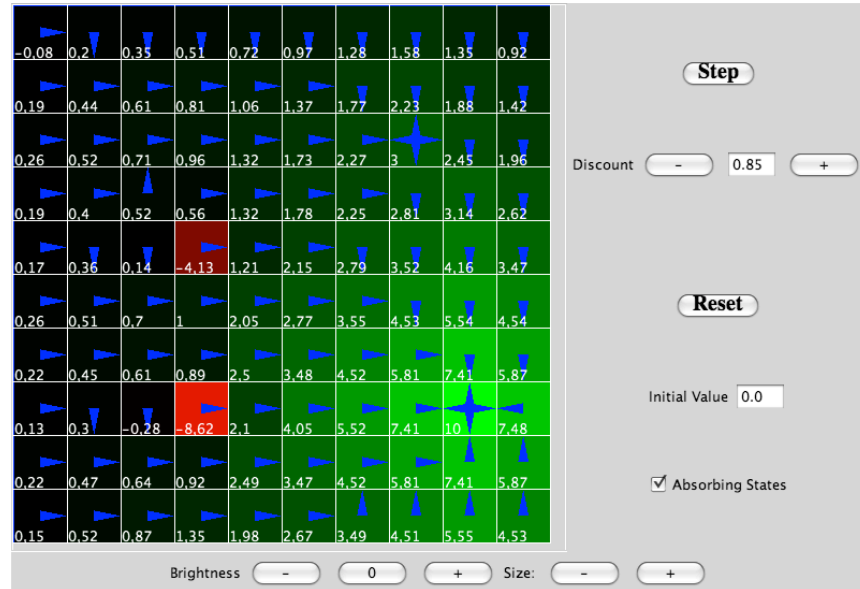
end loop

```

<http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node19.html>

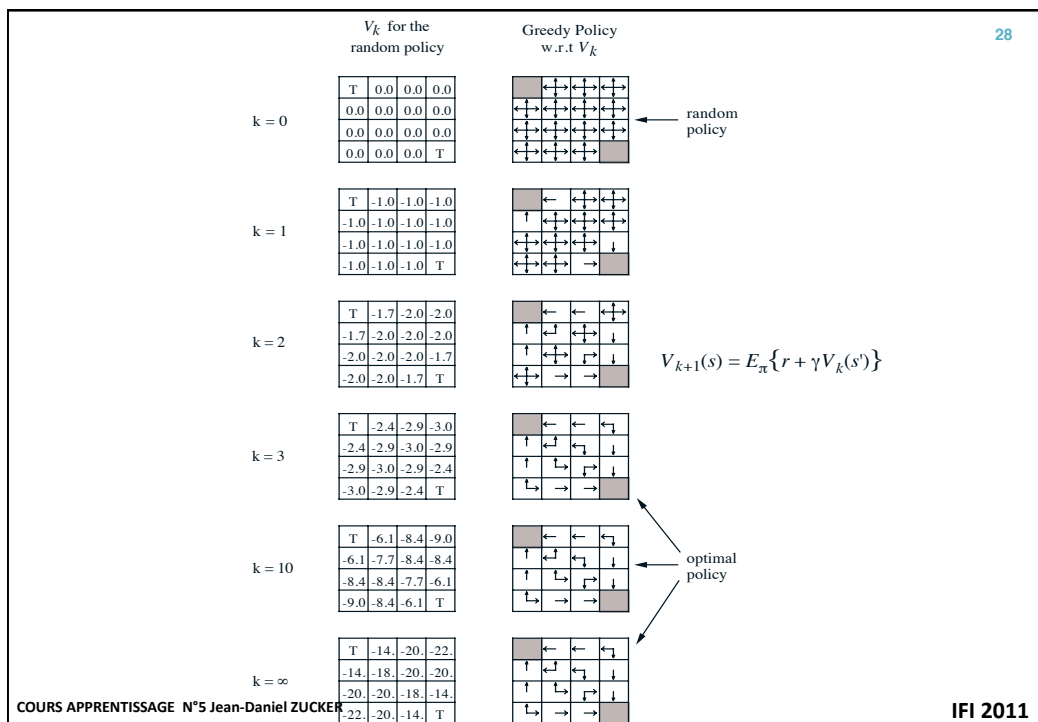
Exemple (Valeurs)

27



COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011



COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

Deux types d'approche pour apprendre π

A) Apprendre V^π la fonction d'utilité liée aux états (TD-learning)

Dans l'état S_i choisir l'action a qui maximise l'utilité $V(S_{i+1})$ supposée de l'état S_{i+1} obtenue après avoir fait l'action a .

Requiert un modèle assez précis de l'environnement pour connaître les états où mènent les actions (exemple: Jeux de dame+Minimax)

B) Apprendre Q^π la fonction d'utilité liée aux actions (Q-learning)

Choisir l'action a qui maximise $Q(S_i, a)$:
l'utilité supposée de l'action a dans l'état S_i

Requiert un modèle limité de l'environnement: on n'a besoin que de mesurer la valeur d'une action et non l'état résultant de l'action (pas de look-ahead) (exemple: attraper une plume, blitz)

Trouver un algorithme: problèmes

BUT: Trouver une politique π : $S \rightarrow A$,
qui à tout état s_t associe l'action a_t qui optimise un **critère de gain**.

- **"Temporal Credit Assignment":**
quelles sont les actions qui doivent être créditées ?
- **Exploration/exploitation:** quel compromis avoir ?
- **Etats partiellement observables:**
si les capteurs ne donnent pas toutes les infos ?
- **Apprentissage à long terme:**
ré-utiliser des connaissances apprises pour d'autres tâches ?

Performances:

vitesse de convergence, regret

Quelle fonction apprendre ?

- **La politique optimale π^* ?**
 - pas d'exemples de la forme (s,a)
- **La récompense cumulée V^* ?**
 - L'agent choisira alors s_1 plutôt que s_2 car $V^*(s_1) > V^*(s_2)$ et comme il faut choisir une action.

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} (r(s,a) + \gamma V^*(\delta(s,a)))$$

- Intéressant ssi $r(s,a)$ et $\delta(s,a)$ sont totalement connues

La fonction Q ci-dessous offre une réponse

- On définit $Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))$

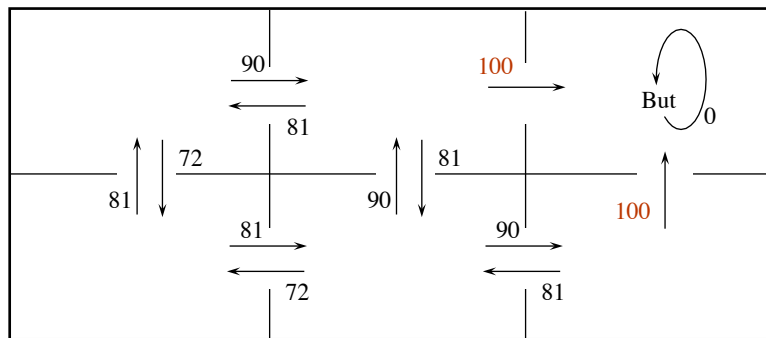
- **Point clef: l'agent pourra prendre les décisions optimales sans connaissances des fonctions $r(s,a)$ et $\delta(s,a)$**

La "beauté" de la fonction Q

33

- La fonction Q est définie comme étant LA fonction qui résume en UN nombre toute l'info nécessaire sur le gain cumulé d'une action a , prise dans l'état s .

$Q(s,a)$



COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

III. Un algorithme pour apprendre $Q(s,a)$ (cas déterministe)

34

$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))$$

- On a : $V^*(s) = \max_{a'} Q(s,a')$

- Définition réursive:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(\delta(s,a), a')$$

Pour chaque couple s, a initialiser la table $Q(s,a)$ à zéro.

Pour l'état courant s

Répéter

Choisir une action a et l'exécuter (**exploration vs. exploitation**)

Réception d'une récompense immédiate r

Observer le nouvel état s'

$$\text{MAJ de } \hat{Q}(s,a) \leftarrow r(s,a) + \gamma \max_{a'} \hat{Q}(\delta(s,a), a')$$

$s \leftarrow s'$

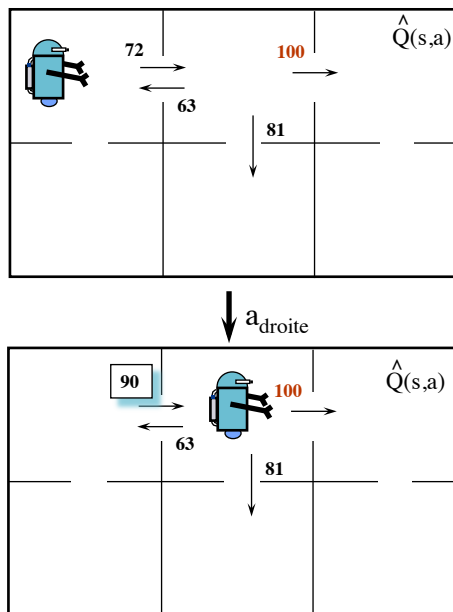
COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

Exemple illustratif...

On Prend $\gamma = 0.9$

$$\begin{aligned}\hat{Q}(s,a) &\leftarrow r + \gamma \max_{a'} \hat{Q}(\delta(s,a), a') \\ &\leftarrow 0 + 0.9 \max\{63, 81, 100\} \\ &\leftarrow 90\end{aligned}$$



COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

Convergence

► Théorème: convergence de l'algo Q-learning déterministe

- Soit $Q_n(s,a)$ l'hypothèse de $Q(s,a)$ après la $n^{\text{ième}}$ mise à jour.
- Si chaque couple (état, action) est visité un nombre infiniment souvent, alors $Q_n(s,a)$ converge vers $Q(s,a)$ quand n tend vers l'infini.

► Démonstration

$$\begin{aligned}\Delta_n &= \max_{s,a} |\hat{Q}_n(s,a) - Q(s,a)| \\ |\hat{Q}_{n+1}(s,a) - Q(s,a)| &= |(r + \gamma \max_{a'} \hat{Q}_n(s',a')) - (r + \gamma \max_{a'} Q(s',a'))| \\ |\hat{Q}_{n+1}(s,a) - Q(s,a)| &= \gamma \left| \max_{a'} \hat{Q}_n(s',a') - \max_{a'} Q(s',a') \right| \\ |\hat{Q}_{n+1}(s,a) - Q(s,a)| &\leq \gamma \max_{a'} |\hat{Q}_n(s',a') - Q(s',a')| \\ |\hat{Q}_{n+1}(s,a) - Q(s,a)| &\leq \gamma \max_{s',a'} |\hat{Q}_n(s'',a') - Q(s'',a')| \\ |\hat{Q}_{n+1}(s,a) - Q(s,a)| &\leq \gamma \Delta_n\end{aligned}$$

COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

Critères de gains espérés: cas non déterministe

➤ Horizon fini

$$E\left(\sum_{t=0}^k r_t\right) = r_0 + r_1 + \dots + r_k$$

➤ Horizon infini avec intérêt

$$E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) = E(r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \dots)$$

➤ En moyenne

$$\lim_{k \rightarrow \infty} E\left(\frac{1}{k} \sum_{t=0}^k r_t\right)$$

Récompenses et actions non déterministes

➤ e.g. au BackGammon: récompense dépend des dés

Pour chaque couple s, a initialisé la table $\hat{Q}(s, a)$ à zéro.

Pour l'état courant s

Répéter

Choisir une action a et l'exécuter

Réception d'une récompense immédiate r

Observer le nouvel état s'

$$\text{MAJ de } \hat{Q}_n(s, a) \leftarrow \hat{Q}_{n-1}(s, a) + \underbrace{\alpha_n \left[r(s, a) + \gamma \max_{a'} \hat{Q}_{n-1}(\delta(s, a), a') - \hat{Q}_{n-1}(s, a) \right]}_{\text{erreur temporelle (TD error)}}$$

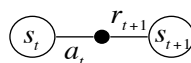
$s \leftarrow s'$

➤ **Théorème de convergence (Watkins & Dayan 92)**

$$\alpha_n = \frac{1}{1 + \text{visites}_n(s, a)}$$

1-Step Tabular Q-Learning

On each state transition:



Mise à jour:

$$\underbrace{Q(s_t, a_t)}_{\text{a table entry}} \leftarrow Q(s_t, a_t) + \alpha \underbrace{\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]}_{\text{TD Error}}$$

$\lim_{t \rightarrow \infty} Q(s, a) \rightarrow Q^*(s, a)$ Optimal behavior found without a model of the environment!
 $\lim_{t \rightarrow \infty} \pi_t \rightarrow \pi^*$ (Watkins, 1989)
 Assumes finite MDP

Une approche

- Apprendre une fonction Q:

$$Q : S \times A \rightarrow \text{valeur}$$

- L'utiliser pour choisir la meilleure action

$$\pi(s) = \operatorname{argmax}_{a \in A} Q(s, a)$$

Choix d'une politique (et donc d'une action)

- **Exploration versus exploitation:**

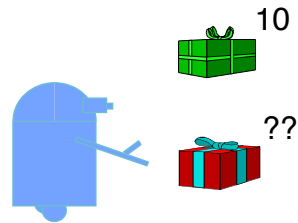
- **Si vous avez confiance en vous:**

- ▢ Dans epsilon % des cas choisissez

$$\pi(s) = \operatorname{argmax}_{a \in A} Q(s, a)$$

- **Sinon,**

- ▢ explorez



Temporal Difference Learning $TD(\lambda)$



Idée: ne pas mettre à jour que les successeurs ou prédécesseurs immédiats.

$$Q^{(1)}(s_t, a_t) = r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)$$



$$\begin{aligned} Q^{(2)}(s_t, a_t) &= r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2}, a) \\ \dots \\ Q^{(n)}(s_t, a_t) &= r_t + \gamma r_{t+1} + \dots + \gamma^n \max_a \hat{Q}(s_{t+n}, a) \end{aligned}$$



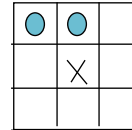
$$\begin{aligned} Q^{(\lambda)}(s_t, a_t) &= (1 - \lambda) \left[Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \dots \right] \\ Q^{(\lambda)}(s_t, a_t) &= r_t + \gamma \left[(1 - \lambda) \max_a \hat{Q}(s_t, a) + \lambda Q^{(\lambda)}(s_{t+1}, a_{t+1}) \right] \end{aligned}$$

TD /TP: le jeux de Tic-Tac-Toe

43

Soit

➤ Un damier à 9 cases



Donner

➤ Construire un programme qui apprenne à gagner

Approche

➤ a) un apprentissage par renforcement basé sur l'action

IV.1. Exemple illustratif du tic-tac-toe

44

Renforcement par différence temporelle (TD-learning [temporal difference])

On associe à chacune des 512 positions une valeur initiale égale à $V(s_{t=0})=0.5$ (sauf pour les positions perdantes 0. et gagnantes 1.).

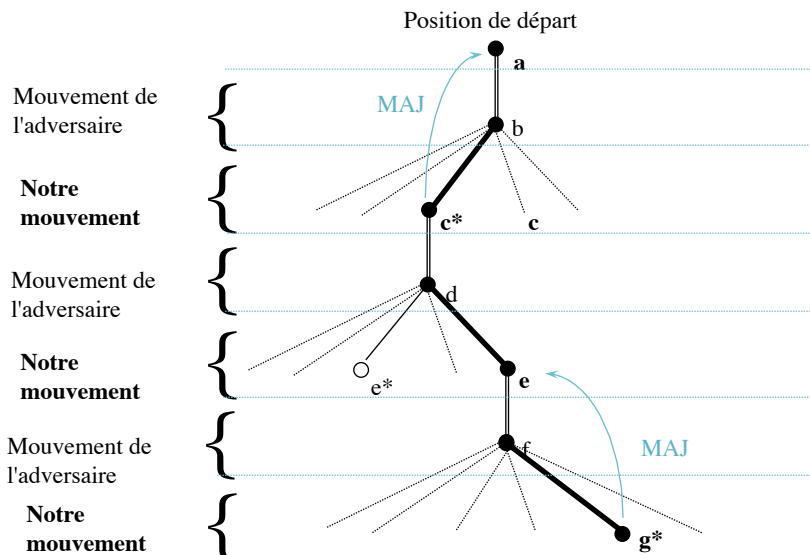
On joue ensuite contre un opposant en **choisissant** l'action qui mène à la position de plus haute récompense (pour laquelle $V(s)$ est maximale) ou parfois on choisit aléatoirement (ϵ) un autre coup, dit *exploratoire*. (ϵ -greedy)

Après chaque coup non exploratoires on met à jour les valeurs:

$$V(s_{t+1}) := V(s_t) + \alpha [(\gamma=1) \times \max(V(\delta(s_t)) - V(s_t))] \quad (\alpha \text{ fonct. décroissante})$$

==> converge vers une décision optimale dans chaque état

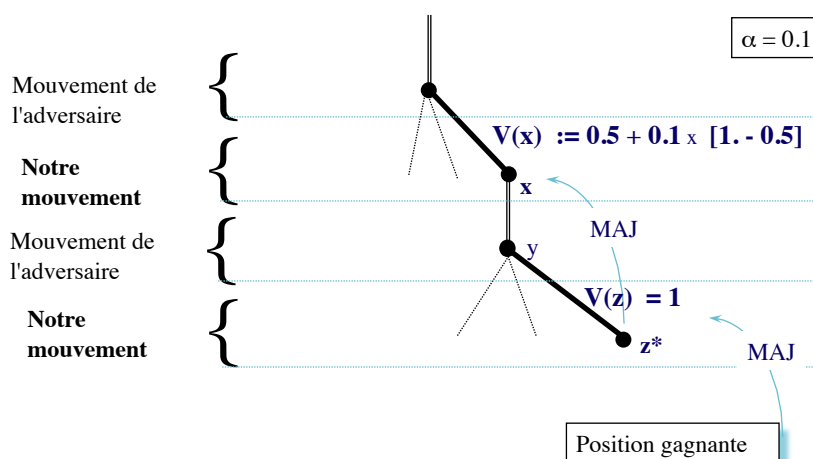
IV.2. Exemple illustratif du tic-tac-toe (suite)



COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

IV.3. Exemple illustratif du tic-tac-toe (suite)



- L'état x passe d'une valeur de 0.5 à 0.55
- Si l'adversaire joue toujours y dans la position x sa valeur ne cessera d'augmenter [m'me si x est, dans l'absolu, une position perdante]

COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

1. Action Selection (Exploration)

Ok, you've learned a value function,
How do you pick Actions?

$$Q \approx Q^*$$

Greedy Action Selection:

Always select the action that looks best:

$$\pi(s) = \arg \max_a Q(s, a)$$

ϵ -Greedy Action Selection:

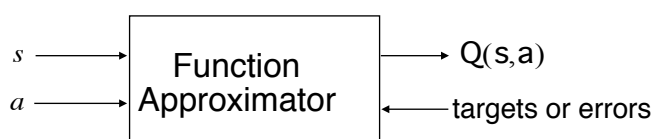
Be greedy most of the time

Occasionally take a random action

Exploitation
vs
Exploration!

Surprisingly, this is the state of the art.

Quand les tables ne suffisent pas: Function Approximation



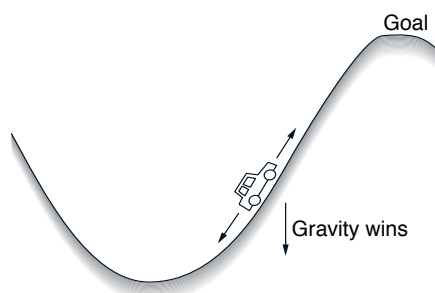
Could be:

- table
- ⇒ • Backprop Neural Network
- ⇒ • Radial-Basis-Function Network
- Tile Coding (CMAC)
- Nearest Neighbor, Memory Based
- Decision Tree

gradient-
descent
methods

The Mountain Car Problem

Moore, 1990



Minimum-Time-to-Goal Problem

SITUATIONS: car's position and velocity

ACTIONS: three thrusts: forward, reverse, none

REWARDS: always -1 until car reaches the goal

No Discounting

RL Applied to the Mountain Car

1. TD Method = Sarsa Backup

2. Action Selection = Greedy

with initial optimistic values

$$Q_0(s, a) = 0$$

3. Function approximator = CMAC

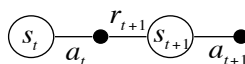
4. Eligibility traces = Replacing

Video Demo

(Sutton, 1995)

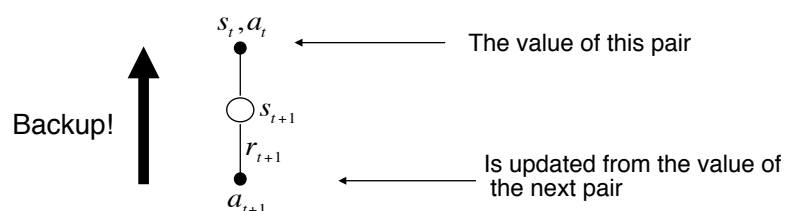
Sarsa Backup

On transition:

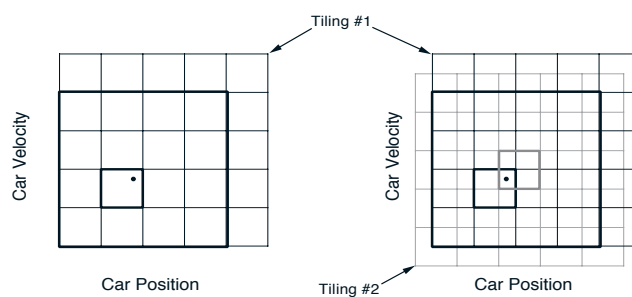


Update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \underbrace{[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]}_{\text{TD Error}}$$



Tile Coding applied to Mountain Car a.k.a. CMACs (Albus, 1980)



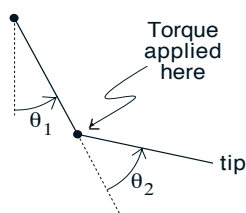
Shape/Size of tiles \Rightarrow Generalization **5x5 grid**

Tilings \Rightarrow Resolution of final approximation **(10)**

The Acrobot Problem

Goal: Raise tip above line

e.g., Dejong & Spong, 1994
Sutton, 1995



Minimum-Time-to-Goal:

4 state variables:
2 joint angles
2 angular velocities

CMAC of 48 layers

RL same as Mountain Car

Reward = -1 per time step

Complex TD Backups (*)

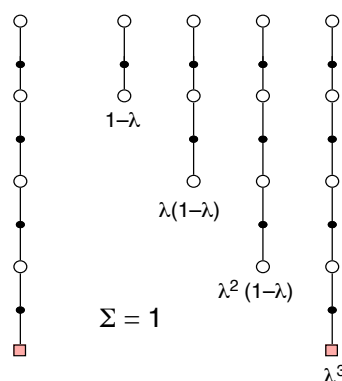
e.g. TD (λ)

Incrementally computes a weighted mixture of these backups as states are visited

$\lambda = 0$ → Simple TD
 $\lambda = 1$ → Simple Monte Carlo

trial

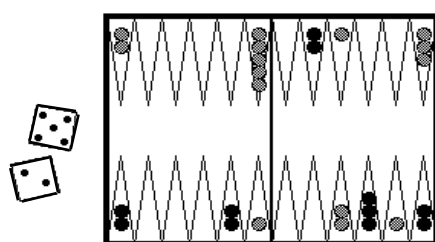
primitive TD backups



World-Class Applications of RL

- **TD-Gammon and Jellyfish** Tesauro, Dahl
World's best backgammon player
- **Elevator Control** Crites & Barto
World's best down-peak elevator controller
- **Inventory Management** Van Roy, Bertsekas, Lee & Tsitsiklis
10-15% improvement over industry standard methods
- **Dynamic Channel Assignment** Singh & Bertsekas, Nie & Haykin
World's best assigner of radio channels to mobile telephone calls

Backgammon



Tesauro, 1992,1995

SITUATIONS: configurations of
the playing board (about 10^{20})

ACTIONS: 20 moves

REWARDS: win: +1

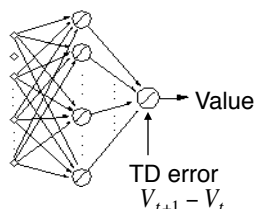
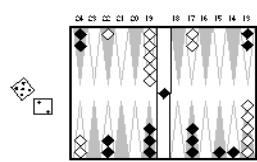
lose: -1

else: 0

Pure delayed reward

Tesauro, 1992–1995

TD-Gammon



Action selection
by 2–3 ply search

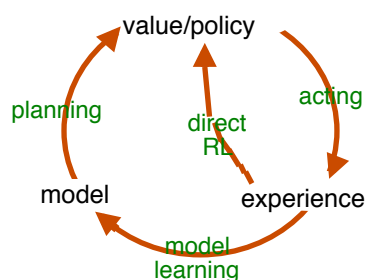
Start with a random network

Play millions of games against self

Learn a value function from this simulated experience

This produces arguably the best player in the world

Accretive Computation "Solves" Planning Dilemmas



Processes only loosely
coupled
Can proceed in parallel,
asynchronously
Quality of solution
accumulates in value &
model memories

Reactivity/Deliberation dilemma

"solved" simply by not opposing search and memory

Intractability of planning

"solved" by anytime improvement of solution

Dyna Algorithm

1. $s \leftarrow$ current state
2. Choose an action, a , and take it
3. Receive next state, s' , and reward, r
4. Apply RL backup to s, a, s', r
e.g., Q-learning update
5. Update Model(s, a) with s', r
6. Repeat k times:
 - select a previously seen state-action pair s, a
 - $s', r \leftarrow$ Model(s, a)
 - Apply RL backup to s, a, s', r
7. Go to 1

Actions \rightarrow Behaviors

MDPs seems too too flat, low-level

Need to learn/plan/design agents at a higher level

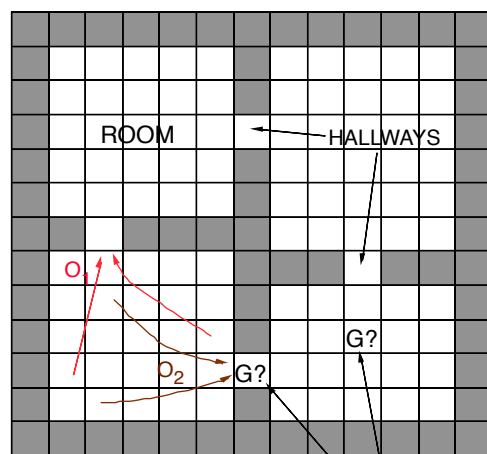
at the level of **behaviors**

rather than just low-level **actions**

e.g., open-the-door rather than twitch-muscle-17
walk-to-work rather than 1-step-forward

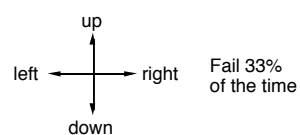
Behaviors are based on entire **policies**,
directed toward **subgoals**
enable abstraction, hierarchy, modularity, transfer...

Rooms Example



Goal states are given
a terminal value of 1

4 rooms
4 hallways
4 unreliable
primitive actions



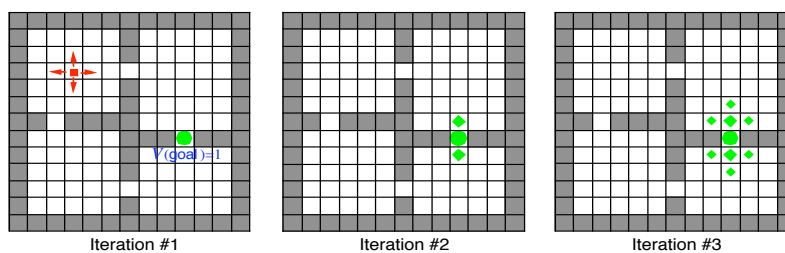
8 multi-step options
(to each room's 2 hallways)

Given goal location,
quickly plan shortest route

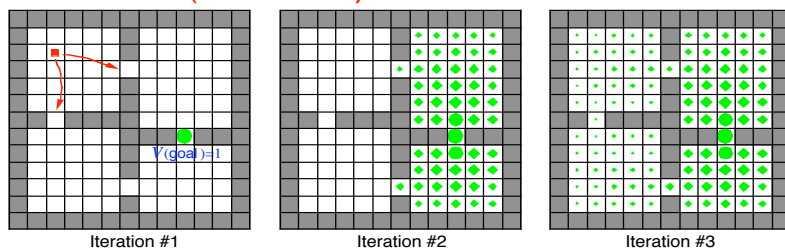
All rewards zero
 $\gamma = .9$

Planning by Value Iteration

with primitive actions (cell-to-cell)

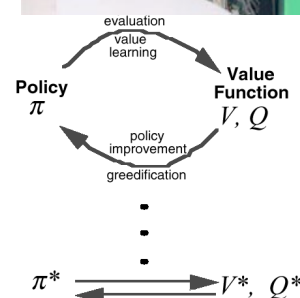
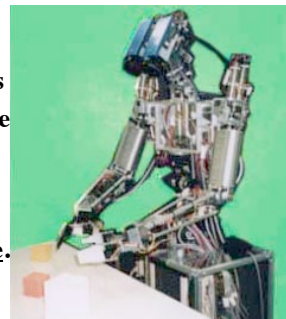


with behaviors (room-to-room)



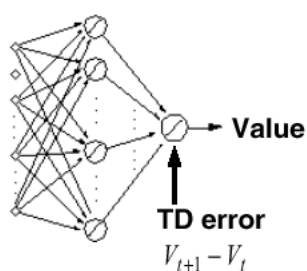
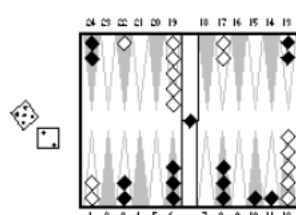
Résumé

- L'apprentissage par renforcement concerne les problèmes d'apprentissage par des agents autonomes de Tâches où le but est de maximiser les récompenses reçues.
- Le Q-learning est une forme d'apprentissage par renforcement qui a des bonnes propriétés de convergence.
- Le Q-learning fait partie d'une famille plus large d'algorithmes: Temporal Difference Learning
- Lien entre l'Apprentissage par renforcement et les processus décisionnels markovien (PDM ou MDP) et la programmation dynamique.
- Renouveau actuel de l'apprentissage par renforcement...



COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

Exemple: TD-Gammon



Action selection
by 2–3 ply search

Démarre avec un réseau aléatoire
Joue des millions de parties contre soi-même
Apprend une fonction d'évaluation

Donnerait le meilleur joueur de BackGammon du monde !

COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

Apprentissage par renforcement

Samuel, A. (1967). Some studies in machine learning using the game of checkers. IIó recent progress. IBM Journal on Research and Development, (pp. 601-617).

Sutton, R. (1998). Introduction to Reinforcement Learning.

<http://envy.cs.umass.edu/People/sutton/sutton.html>

3-2. PD : Comment améliorer une politique

Relation d'ordre sur les politiques :

Soient π et π' deux politiques déterministes, tq $s \in \mathcal{E}$:

$$Q^{\pi'}(s, \pi'(s)) \geq V^{\pi}(s) \quad (1)$$

Alors la politique π' est au moins aussi bonne que π :

$$V^{\pi'}(s) \geq V^{\pi}(s)$$

➡ Si l'on trouve une modification π' de la politique π vérifiant l'inégalité (1), alors on obtient une meilleure politique

3-3. PD : Amélioration de politique Cont.

Il suffit de faire cela pour tous les états pour obtenir une nouvelle politique π' qui est gloutonne par rapport à V^π :

$$\begin{aligned}\pi'(s) &= \arg \max_a Q^\pi(s, a) \\ &= \arg \max_a \sum_s P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')]\end{aligned}$$

➡ Alors $V^{\pi'} \geq V^\pi$

3-3. PD : Amélioration de politique (Cont.)

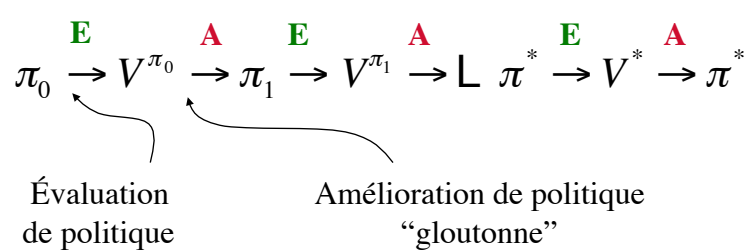
What if $V^{\pi'} = V^{\pi}$?

$$\text{i.e., for all } s \in S, \quad V^{\pi'}(s) = \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^{\pi}(s')] ?$$

But this is the Bellman Optimality Equation.

So $V^{\pi'} = V^*$ and both π and π' are optimal policies.

3-3. PD : Itération de politique



3-3. Algorithme d'itération de politique

Initialisation arbitraire de π

Faire

calcul de la fonction de valeur avec π

$$V_{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_{\pi}(s')$$

Amélioration de la politique à chaque état

$$\pi'(s) \leftarrow \max_a \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_{\pi}(s') \right)$$

$\pi' := \pi$

jusqu'à ce qu'aucune amélioration ne soit possible

3-3. PD : Policy Iteration

1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

Repeat

$\Delta \leftarrow 0$

For each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} [\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

3. Policy Improvement

$policy_stable \leftarrow true$

For each $s \in \mathcal{S}$:

$b \leftarrow \pi(s)$

$\pi(s) \leftarrow \arg \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$

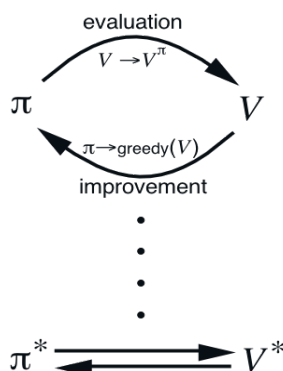
If $b \neq \pi(s)$, then $policy_stable \leftarrow false$

If $policy_stable$, then stop; else go to 2

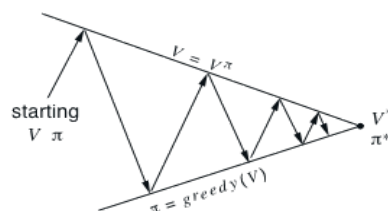
3-3. PD : Itération généralisée de politique

Generalized Policy Iteration (GPI):

Toute interaction d'étape d'évaluation de politique et d'étape d'amélioration de politique indépendamment de leur granularité :



Métaphore géométrique pour La convergence de GPI :



4. Environnement inconnu : Différences temporelles

Soit la méthode d'estimation par moyennage :

La moyenne des premiers k renforcements est (en ignorant la dépendance sur a):

$$Q_k = \frac{r_1 + r_2 + \dots + r_k}{k}$$

Peut-on faire le même calcul incrémentalement ?

Oui :

$$Q_{k+1} = Q_k + \frac{1}{k+1} [r_{k+1} - Q_k]$$

Règle classique d'amélioration :

$$\text{NouvelleEstimation} = \text{AncienneEstimation} + \text{Pas}[\text{Cible} - \text{AncienneEstimation}]$$

4-1. TD learning : évaluation par méthode des différences temporelles

Évaluation de politique :

pour une politique donnée π , calculer la fonction d'utilité V^π

Simple every - visit Monte Carlo method :

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)]$$

\uparrow
cible: le vrai gain sur une durée t

The simplest TD method, TD(0) :

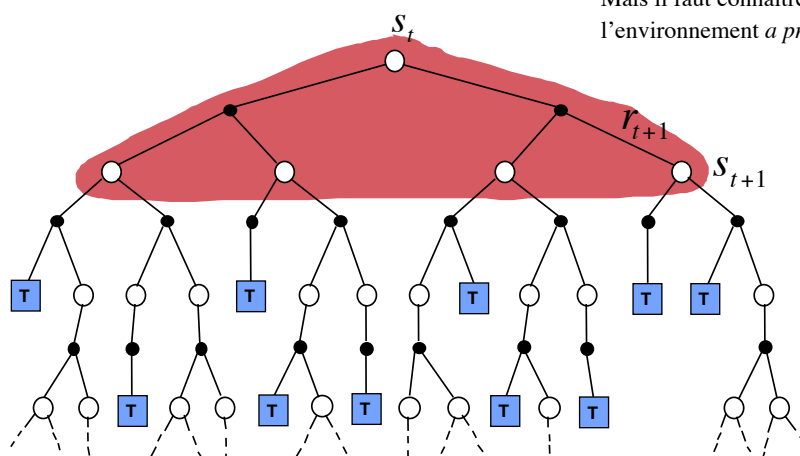
$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

$\underbrace{\hspace{10em}}$
cible: une estimation du gain

4-1. TD learning : cf. Dynamic Programming

$$V(s_t) \leftarrow E_\pi \{r_{t+1} + \gamma V(s_{t+1})\}$$

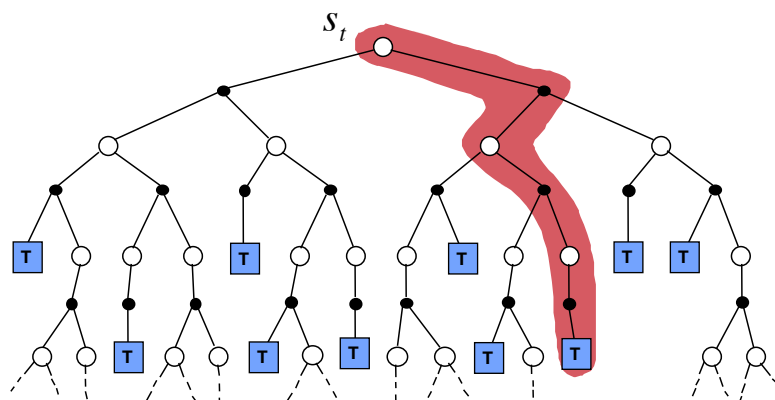
On calcule l'espérance.
Mais il faut connaître
l'environnement *a priori*.



4-1. TD learning : Simple Monte Carlo

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)]$$

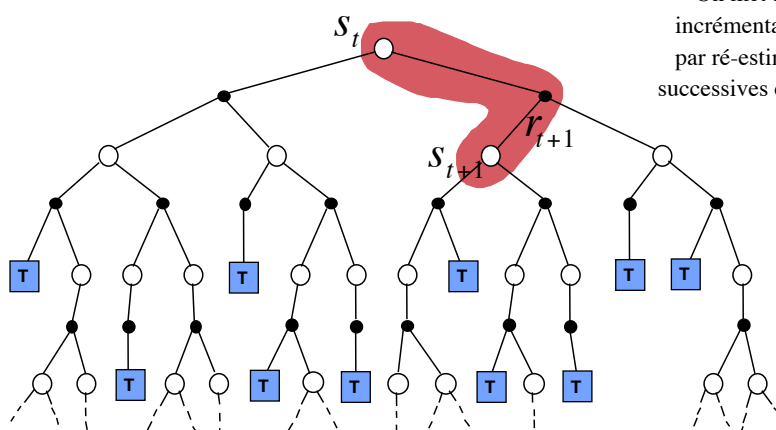
where R_t is the actual return following state s_t .



4-1. TD learning : Simplest TD Method

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

On met à jour
incrémentalement
par ré-estimations
successives et locales



4-1. TD learning : algo d'évaluation par différences temporelles

Initialisation :

$\pi \leftarrow$ politique à évaluer

$V \leftarrow$ une fonction arbitraire d'évaluation

Répéter (pour chaque pas de l'épisode) :

$a \leftarrow$ action préconisée par π pour s

Faire a ; recevoir r ; voir état suivant s'

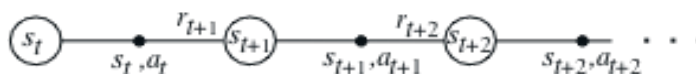
$V(s) \leftarrow V(s) + \alpha [r + \gamma V(s') - V(s)]$

$s \leftarrow s'$

jusqu'à s terminal

4-1. TD learning : Learning An Action-Value Function $Q(s,a)$

Estimate Q^π for the current behavior policy π .



After every transition from a nonterminal state s_t , do this :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1}) = 0$.

4-2. TD learning : Q-Learning

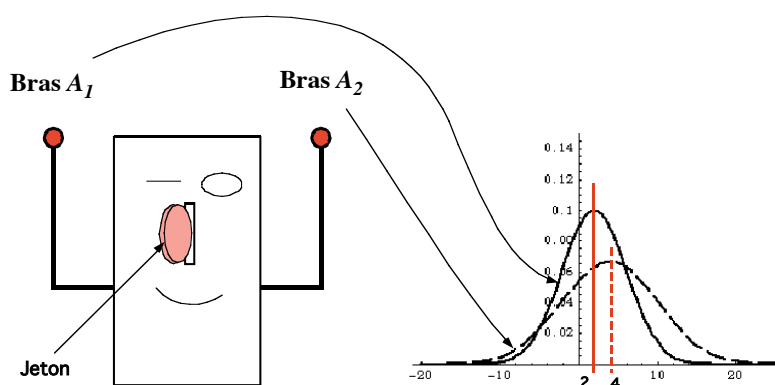
One - step Q - learning :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

```

Initialize  $Q(s, a)$  arbitrarily
Repeat (for each episode):
  Initialize  $s$ 
  Repeat (for each step of episode):
    Choose  $a$  from  $s$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
    Take action  $a$ , observe  $r, s'$ 
     $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
     $s \leftarrow s'$ 
  until  $s$  is terminal
  
```

4-2. Rappel : Le dilemme exploitation vs. exploration



Quelle politique pour maximiser le gain avec 1000 tirages ?

4-2- Sélection d'action ϵ -gloutonne

- Sélection d'action gloutonne :

$$a_t = a_t^* = \arg \max_a Q_t(a)$$

- ϵ -gloutonne :

$$a_t = \begin{cases} a_t^* & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

... La manière la plus simple de pondérer l'exploration et l'exploitation

4-2- Sélection d'action Softmax

- Softmax action selection methods grade action probs. by estimated values.
- The most common softmax uses a Gibbs, or Boltzmann, distribution:

Choose action a on play t with probability

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^n e^{Q_t(b)/\tau}},$$

where τ is the

“computational temperature”

4-3. L'apprentissage Q (Q -learning)

- **Idée** [Watkins,89] : Estimer les valeurs Q “en-ligne”, en trouvant à la fois la politique et la fonction d'évaluation d'action :

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') \right)$$

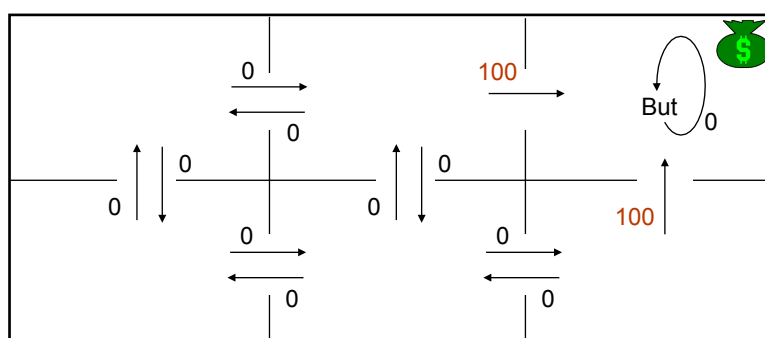
MAJ à chaque fois que l'action a est prise dans s .

- **Théorème** : Si chaque action est exécutée un nombre infini de fois dans chaque état, les valeurs Q calculées convergent vers Q^* , conduisant à une politique optimale.

4-3. Exemple

(1/4)

$r(s,a)$ récompense immédiate



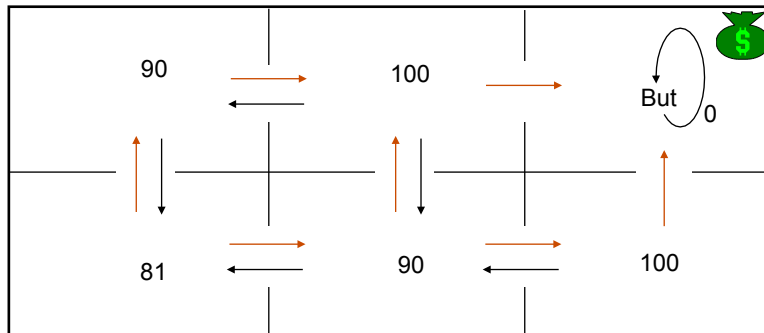
- **Rq**: La dernière étape assure la récompense (jeux, monde des blocs, etc.)
- **Tâche**: apprendre la meilleure stratégie

4-3. Exemple

(2/4)

87

- On définit la récompense cumulée $V^\pi(s_t) = \sum_{t=0}^{\infty} \gamma^t r_t$
- Le problème: trouver $\pi^* = \operatorname{argmax}_{\pi} (V^\pi(s))$



$V^*(s) = V^{\pi^*}(s)$ récompense cumulée optimale

COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

IFI 2011

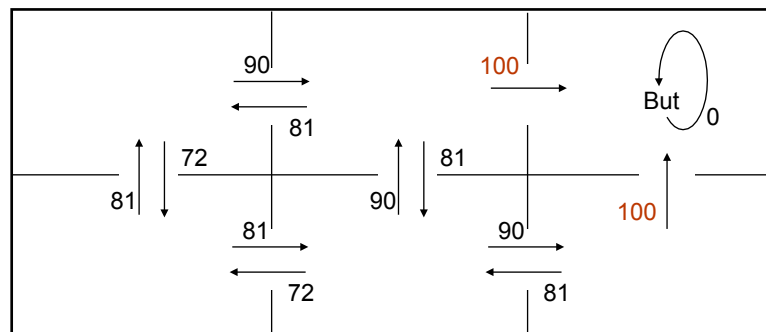
4-3. Exemple

(3/4)

88

- La fonction Q est défini comme étant LA fonction qui résume en UN nombre toute l'info nécessaire sur le gain cumulé d'une action a , prise dans l'état s .

$Q(s,a)$



COURS APPRENTISSAGE N°5 Jean-Daniel ZUCKER

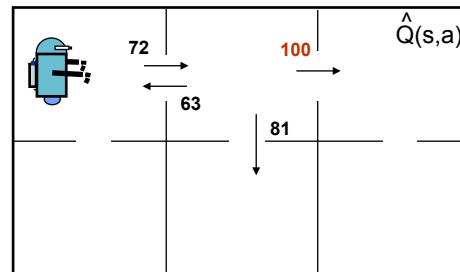
IFI 2011

4-3. Exemple

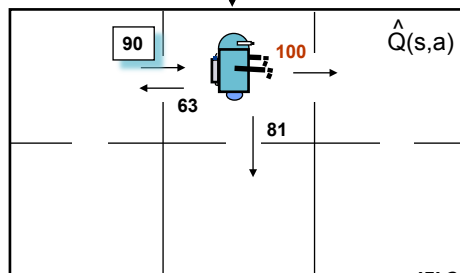
(4/4)

On Prend $\alpha = 1$.

$$\begin{aligned}\hat{Q}(s,a) &\leftarrow r + \gamma \max_{a'} \hat{Q}(\delta(s,a), a') \\ &\leftarrow 0 + 0.9 \max \{63, 81, 100\} \\ &\leftarrow 90\end{aligned}$$



↓ a_{droite}



5. Apprentissage avec généralisation

- Si l'espace S (ou $S \times A$) est trop important pour l'utilisation d'une table mémorisant les prédictions
- Deux options :
 - Utilisation d'une **technique de généralisation** dans l'espace S ou l'espace $S \times A$ (e.g. réseau de neurones, ...)
 - Utilisation d'une **technique de regroupement** d'états en classes d'équivalence (même prédiction et même action générée).

5. Généralisation : Approximation de la fonction $V(s)$

Comme avant : Évaluation de politique :

pour une politique donnée π , calculer la fonction d'utilité V^π

Mais avant, les fonctions d'utilité étaient stockées dans des tables.

Maintenant, l'estimation de la fonction d'utilité au temps t , V_t , dépend d'un vecteur de paramètres θ_t , et seul ce vecteur de paramètres est mis à jour.

e.g., θ_t pourrait être le vecteur de poids de connexions d'un réseau de neurones.

5. Généralisation : Backups as Training Examples


e.g., the TD(0) backup :

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

As a training example:

$$\{\text{description of } s_t, r_{t+1} + \gamma V(s_{t+1})\}$$


input


target output

5. Généralisation : n'importe quelle méthode inductive ?

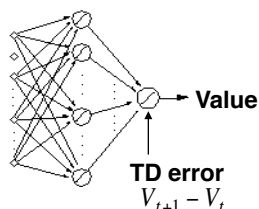
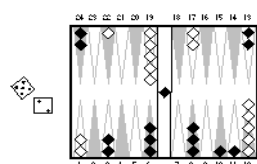
- **En principe, oui :**
 - ▢ Réseaux de neurones artificiels
 - ▢ Arbres de décision
 - ▢ Méthodes de régression multivariées
 - ▢ etc.
- **Mais l'App. par R. a des exigences particulières :**
 - ▢ Apprendre tout en agissant
 - ▢ S'adapter à des mondes non stationnaires

6. Some Notable RL Applications

- **TD-Gammon: Tesauro**
 - world's best backgammon program
- **Elevator Control: Crites & Barto**
 - high performance down-peak elevator controller
- **Inventory Management: Van Roy, Bertsekas, Lee & Tsitsiklis**
 - 10–15% improvement over industry standard methods
- **Dynamic Channel Assignment: Singh & Bertsekas, Nie & Haykin**
 - high performance assignment of radio channels to mobile telephone calls

6. TD-Gammon

Tesauro, 1992–1995



Action selection
by 2–3 ply search

Start with a random network

Play very many games against self

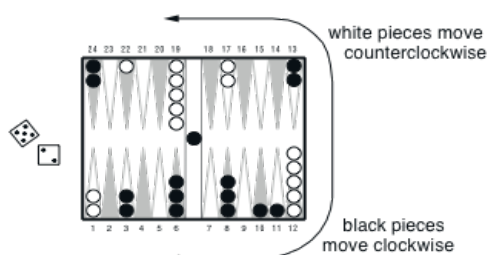
Learn a value function from this simulated experience

This produces arguably the best player in the world

6. Réalisations : TD Gammon

Tesauro 1992, 1994, 1995, ...

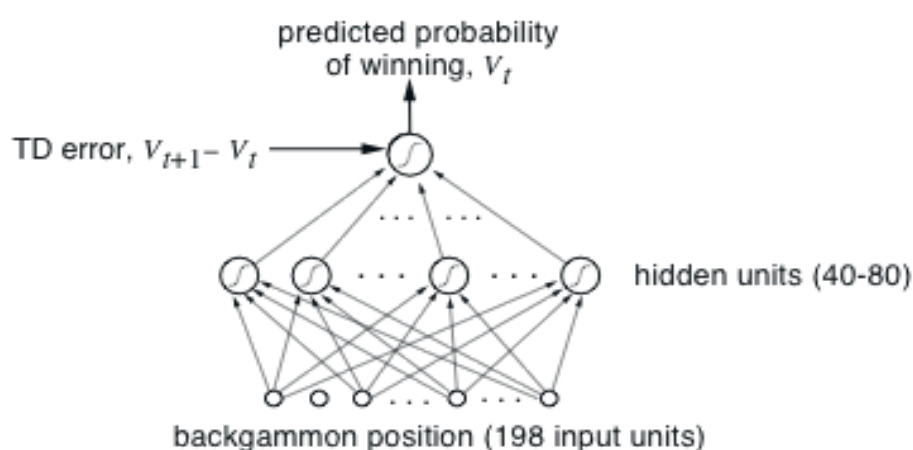
- White has just rolled a 5 and a 2 so can move one of his pieces 5 and one (possibly the same) 2 steps
- Objective is to advance all pieces to points 19-24
- Hitting
- Doubling
- 30 pieces, 24 locations implies enormous number of configurations
- Effective branching factor of 400



6. Réalisations : A Few Details

- Reward: 0 at all times except those in which the game is won, when it is 1
- Episodic (game = episode), undiscounted
- Gradient descent TD(1) with a multi-layer neural network
 - ▢ weights initialized to small random numbers
 - ▢ backpropagation of TD error
 - ▢ four input units for each point; unary encoding of number of white pieces, plus other features
- Use of afterstates
- Learning during self-play

6. Réalisations : Multi-layer Neural Network



6. Réalisations : Summary of TD-Gammon Results

Program	Hidden Units	Training Games	Opponents	Results
TD-Gam 0.0	40	300,000	other programs	tied for best
TD-Gam 1.0	80	300,000	Robertie, Magriel, . . .	−13 points / 51 games
TD-Gam 2.0	40	800,000	various Grandmasters	−7 points / 38 games
TD-Gam 2.1	80	1,500,000	Robertie	−1 point / 40 games
TD-Gam 3.0	80	1,500,000	Kazaros	+6 points / 20 games

7. Bilan : trois idées principales

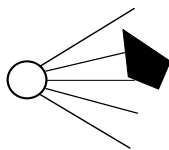
1. La passage par des **fonctions d'utilité**
2. La **rétro-propagation de ces valeurs** le long de trajectoires réelles ou simulées
3. **Itération généralisée de politique** : (i) calculer continuellement une estimation de la fonction d'utilité optimale et (ii) chercher une politique optimale grâce à cette estimation, qui, en retour, s'adapte en conséquence

7. Bilan : Frontier Dimensions

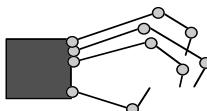
- **Prove convergence for bootstrapping control methods.**
- **Trajectory sampling**
- **Non-Markov case:**
 - ▢ **Partially Observable MDPs (POMDPs)**
 - Bayesian approach: belief states
 - construct state from sequence of observations
 - ▢ Try to do the best you can with non-Markov states
- **Modularity and hierarchies**
 - ▢ Learning and planning at several different levels
 - Theory of options

7. Bilan : More Frontier Dimensions

- **Using more structure**
 - ▢ **factored state spaces: dynamic Bayes nets**



- ▢ **factored action spaces**



7. Bilan : Still More Frontier Dimensions

- **Incorporating prior knowledge**

- advice and hints
- trainers and teachers
- shaping
- Lyapunov functions
- etc.

Sources documentaires

- **Ouvrages / articles**

- Sutton & Barto (98) : *Reinforcement Learning : an introduction*. MIT Press, 1998.
- Kaelbling L.P. (93) : *Learning in embedded systems*. MIT Press, 1993.
- Kaelbling, Littman & Moore (96) : *Reinforcement learning : A survey*. Journal of Artificial Intelligence Research, 4:237-285.

- **Sites web**

- <http://http://www-anw.cs.umass.edu/~rich/RL-FAQ.html>
(FAQ maintenue par Rich Sutton et point d'entrée pour de nombreux sites)