Apprentissage par renforcement
École en intelligence artificielle et sciences cognitives

Alexandre Blondin Massé

Département d'informatique
Université du Québec à Montréal

Équipe sciences des données et calcul haute performance
Institut de recherche d'Hydro-Québec

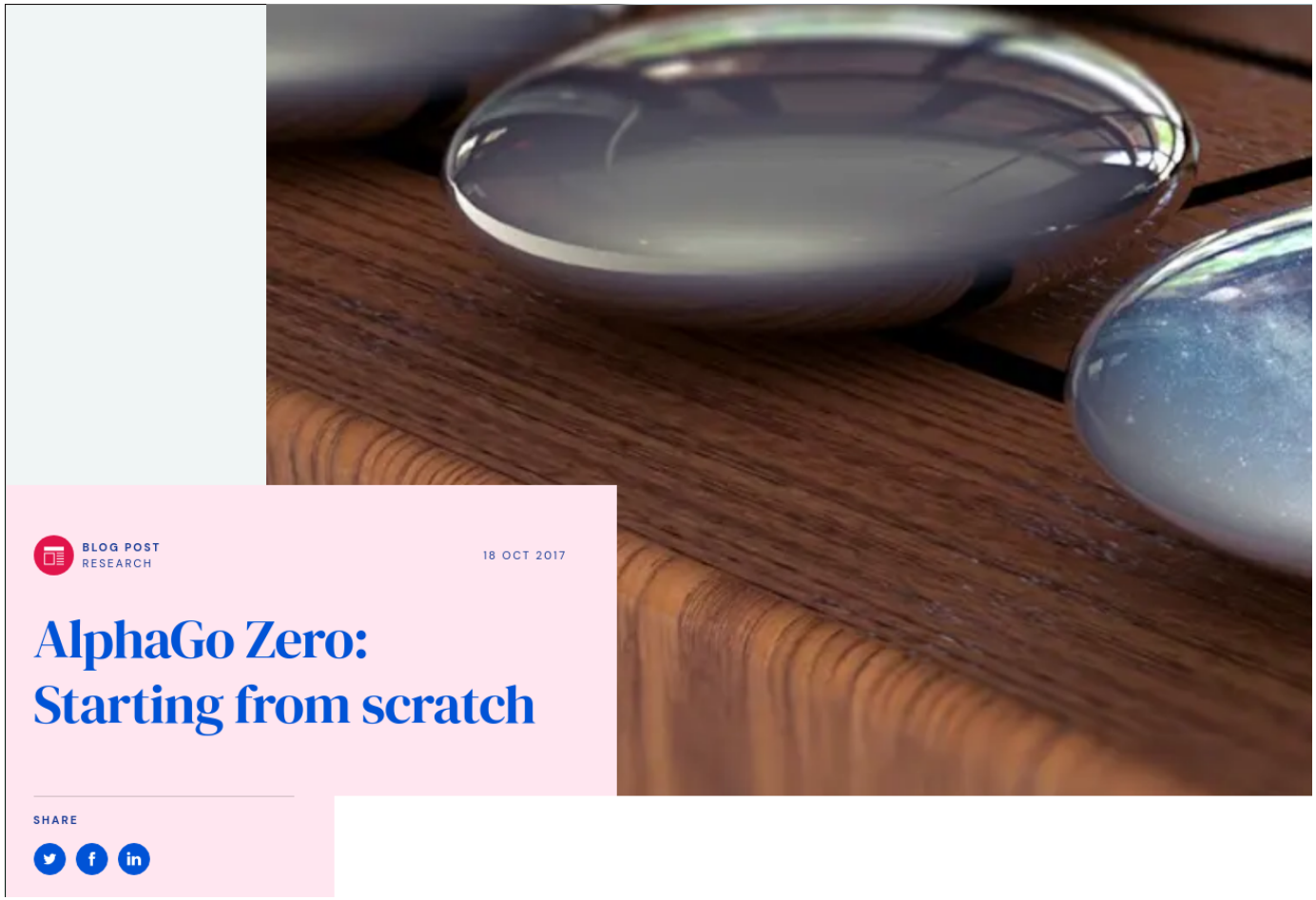
Hiver 2021

AlphaGo

Source: Youtube

AlphaGo contre Lee Sedol, 2016

AlphaGo Zero



Source: DeepMind

AlphaGo Zero, octobre 2017

AlphaZero

Source: DeepMind

AlphaZero généralise AlphaGo, décembre 2017

Comment ça marche? (1/2)

Jouer contre soi (*self-play*)

- Pas besoin de **parties** jouées
- Pas besoin d'**expertise** (données **annotées**)

Réseau de neurones profond

- Plateau de jeu ressemble à une **image**
- Réseau **résiduel** (*ResNet*) de 20 ou 40 **blocs**

Parallélisation

- *Génération des parties*: 5,000 TPU[1] de 1re génération
- *Entraînement du réseau*: 64 TPU de 2e génération

[1] Tensor Processing Unit

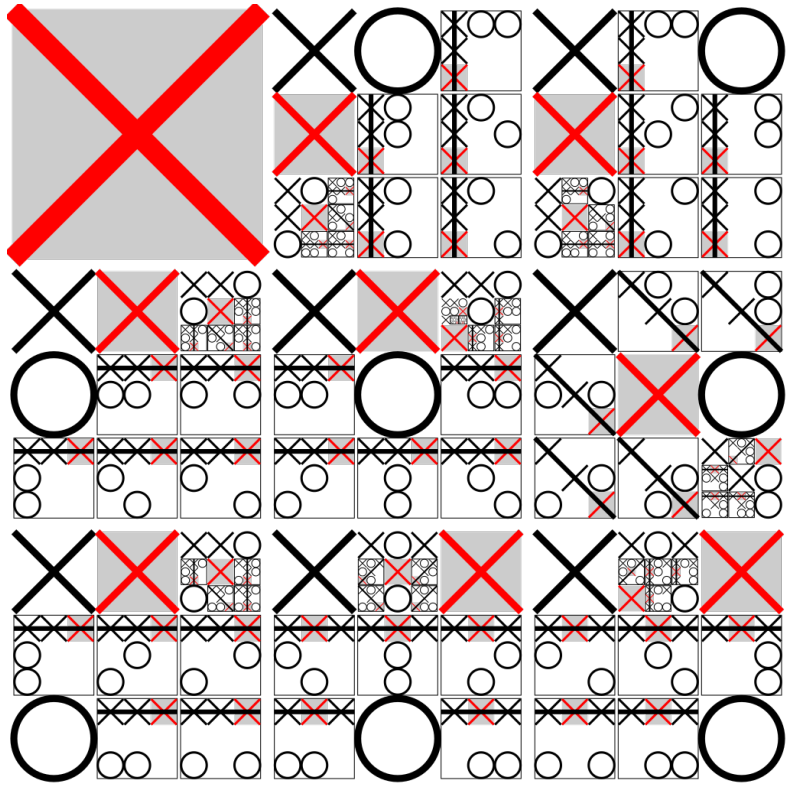
Comment ça marche? (2/2)

1: **fonction** AlphaZero()
2: Initialiser $f_\theta : \mathcal{S} \to \Pi \times \mathbb{R}$
3: **tant que** *vrai* **faire**
4: JouerPartie()
5: MettreAJour(θ)
6:
7: **fonction** JouerPartie()
8: Soit S l'état initial
9: **tant que** la partie n'est pas finie **faire**
10: $\pi \leftarrow MCST(S, f_\theta)$
11: Enregistrer (S, π)
12: Mettre à jour S à partir de π
13:
14: **fonction** MettreAJour(θ)()
15: Soit $z \in \{-1, 0, +1\}$ le résultat de JouerPartie() pour (S, π)
16: Entraîner f_θ sur (S, π, z) par descente du gradient

Table des matières

Jeux et intelligence artificielle

Tic-tac-toe: stratégie parfaite (1/2)



(Source: Wikipedia)

- Si `X` commence dans un coin
- Décrite par Newell et Simon en **1972**:

Tic-tac-toe: stratégie parfaite (2/2)

1. **Gain**: si on peut gagner directement
2. **Bloc**: si l'adversaire menace de gagner directement
3. **Fourchette**: si on peut créer deux menaces de gagner simultanées
4. **Bloquer une fourchette**: s'il n'y en a qu'une possible, jouer là, s'il y en a plusieurs, jouer sur une case commune aux fourchettes, sinon, on joue un coup qui menace un gain immédiat (donc l'adversaire doit l'empêcher), de sorte qu'en défendant, l'adversaire ne crée pas de fourchette
5. **Centre**: jouer au centre, si la case est libre.
6. **Coin opposé**: jouer dans le coin opposé à l'adversaire, si la case est libre
7. **Coin**: s'il y a un coin de libre
8. **Côté**: s'il y a un côté de libre

Jeux de plateau

Tic-tac-toe

Newell et Simon, 1972

Puissance 4

VICTOR, 1989

Échecs

Deep Blue, 1996-1997

Go/shogi/échecs

AlphaGo, AlphaGo Zero, AlphaZero, 2017-2018

Prochaine étape?

« [La victoire d'AlphaGo] marque la fin d'une époque... les jeux de plateau sont plus ou moins réglés, et il est temps de passer à autre chose. »

— Murray Campbell (équipe de Deep Blue)

Apprentissage automatique

Trois types d'apprentissage automatique

Non supervisé

- Données *non étiquetées*
- Aucune *rétroaction*
- On recherche des informations *cachées*

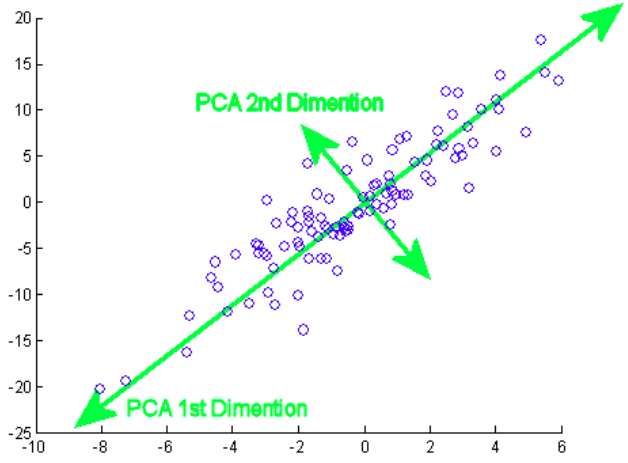
Supervisé

- Données *étiquetées*
- *Rétroaction directe*
- On souhaite *prédire*

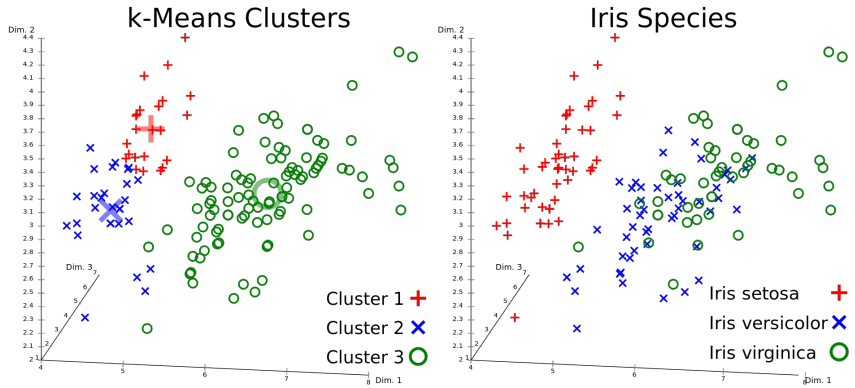
Par renforcement

- *Agent* évoluant dans un *environnement*
- Mécanisme de *récompenses*
- On souhaite apprendre une série *d'actions*

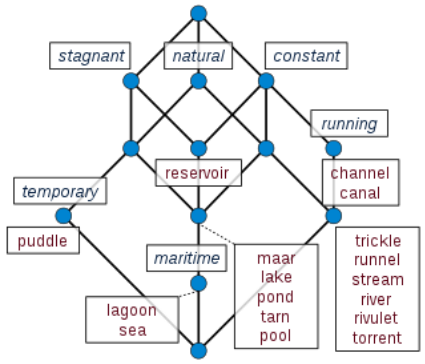
Apprentissage non supervisé



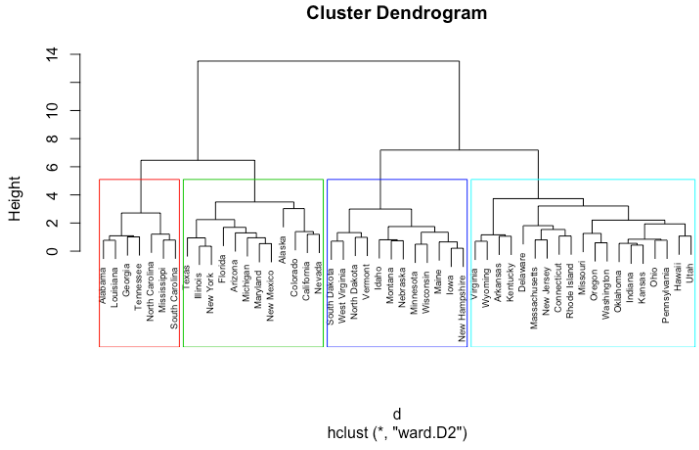
(Source: Machine Learning Course)
Analyse en composantes principales



(Source: Wikipedia)
Partitionnement en *k*-moyennes

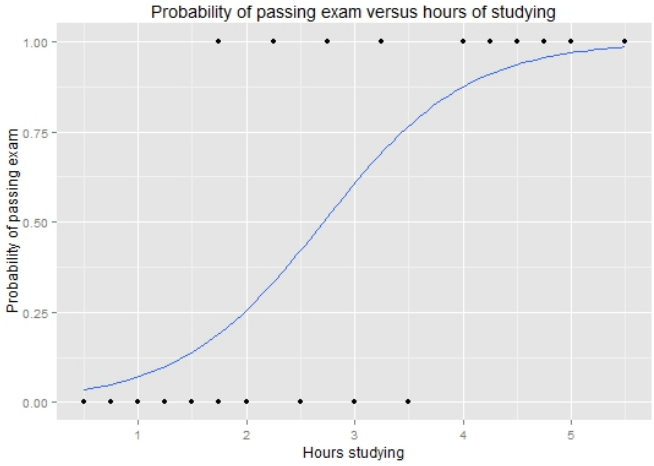


(Source: Wikipedia)
Analyse de concepts formels

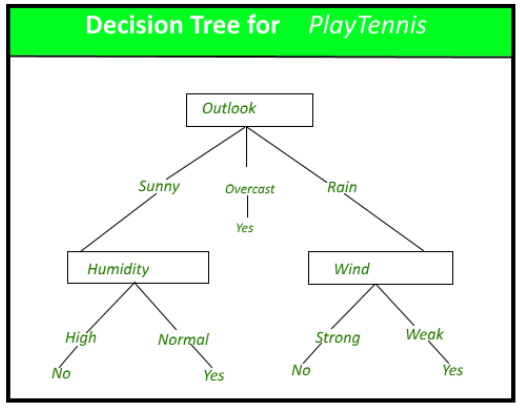


(Source: UC Business Analytics)
Partitionnement hiérarchique

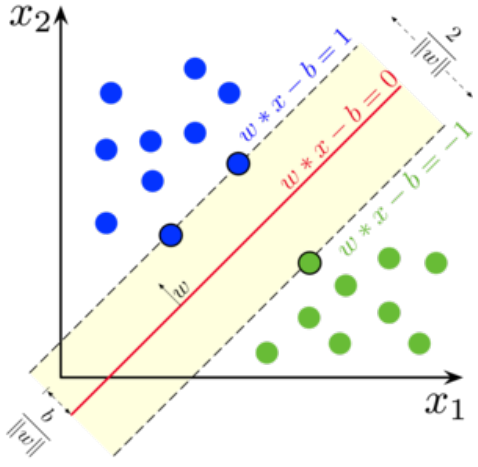
Apprentissage supervisé



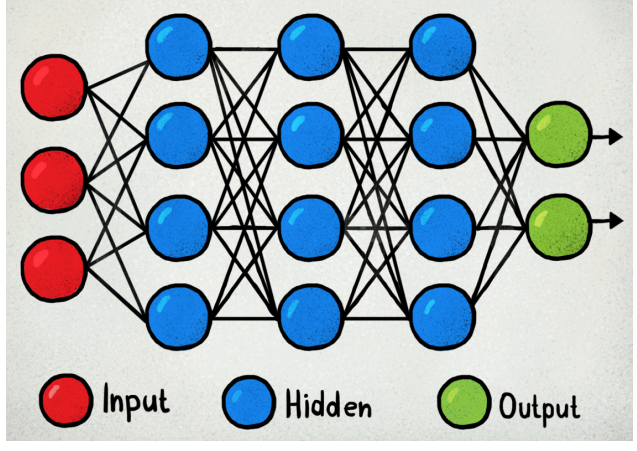
(Source: Wikipedia)
Régression



(Source: GeeksforGeeks)
Arbre de décision

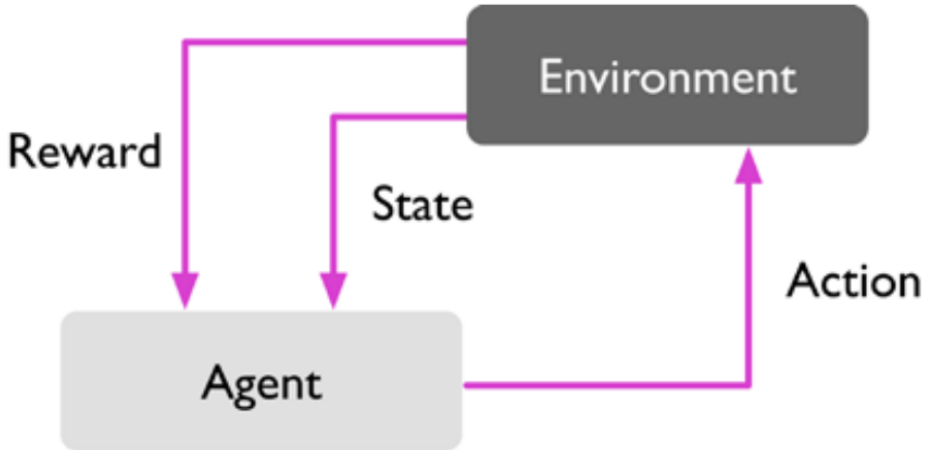


(Source: Wikipedia)
Machine à vecteurs de support



(Source: Medium.com)
Réseau de neurones artificiels

Apprentissage par renforcement



(Source: Python Machine Learning)

- L'agent effectue des **actions**
- L'environnement communique l'**état** résultant et...
- ...retourne une **récompense** (immédiate) à l'agent
- On souhaite **optimiser** le total des récompenses

Références principales

- *Introduction to reinforcement learning*, R.S. Sutton, A.G. Barto et al., MIT Press Cambridge, 2018, disponible en ligne
- *Python Machine Learning, 3rd edition*, S. Raschka, Packt Publishing, 2019, site officiel
- *Deep Learning*, par I. Goodfellow, Y. Bengio and A. Courville, MIT Press, 2016, disponible en ligne
- *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*, par D. Silver et al., 2017, disponible en ligne
- *Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model*, par J. Schrittwieser et al., ArXiV, 2019, disponible en ligne

Apprentissage par renforcement

Généralités

- Apprentissage par **essais** et **erreurs**
- Similaire à ce que les **animaux** et les **humains** font
- L'agent **découvre** les actions plus intéressantes
- Mécanisme de **récompense** immédiates et futures
- L'agent doit équilibrer l'**exploitation** et l'**exploration**

Différence avec autres types d'apprentissage

- Pas **supervisé**: pas de données étiquetées, plus interactif, mais souvent combinés
- Pas **non supervisé**: ne cherche pas de structure cachée, mais maximise la récompense

Politique (*policy*)

- **Comportement** de l'agent à tout moment
- Fonction qui indique, pour chaque **état perçu**, quelle(s) **action(s)** devrait (pourraient) être effectuée(s):

$$\pi : \mathcal{S} \rightarrow \mathcal{A}$$

- Peut être **déterministe** ou **stochastique**
- Peut être **précalculée** ou calculée en **temps réel**

Récompense (*reward*)

- En anglais, *reward signal*
- Rétroaction émise à **chaque étape**
- Généralement un **nombre réel**
- L'agent cherche à **maximiser** le total de ses récompenses
- **Pénalité**: souvent un nombre négatif
- Peut aussi être **nulle** (ni récompense ni pénalité)
- Mesure **locale**
- Utile pour **mettre à jour** la politique
- Peut être **déterministe** ou **stochastique**

Valuation d'un état

- En anglais, *value function* ou *V-function*
- Fonction qui indique, pour chaque **état**, la **récompense totale** que l'agent peut s'attendre à obtenir:

$$V : \mathcal{S} \to \mathbb{R}$$

- Évaluation **globale**, complémentaire aux **récompenses**
- En général beaucoup plus difficile à évaluer qu'une **récompense**
- Doit souvent être **estimée**

Qualité d'une action

- En anglais, *Q-function*
- Fonction qui indique, pour chaque paire **état-action**, la **récompense totale** que l'agent peut s'attendre à obtenir:

$$Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$

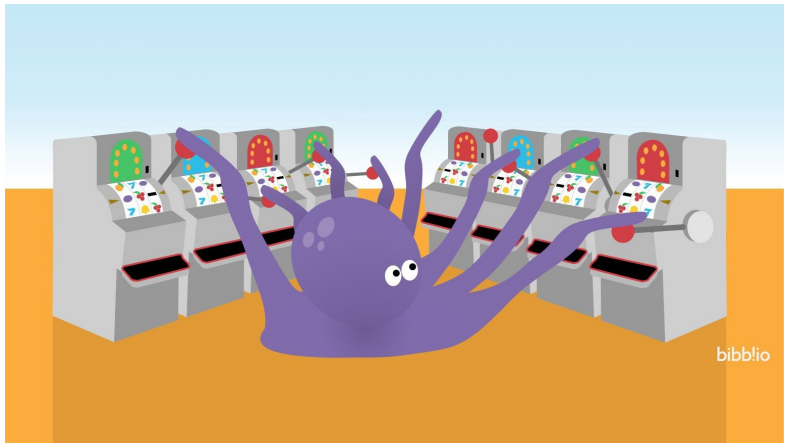
- Doit souvent être **estimée**

Note

- Les fonctions V et Q sont fortement interreliées
- Selon la situation, on s'intéresse à l'une ou l'autre

Quelques exemples

Exemple 1: problème du bandit manchot



Source: bibblio

- Choix parmi k **actions** différentes a_1, a_2, ..., a_k
- Lorsqu'on choisit l'action a_i, on obtient une **récompense** R_t
- On **répète** ce choix $t_{\max}$ fois
- On ne **connaît pas** la distribution de probabilité de R_t
- Mais on sait qu'elle est **stationnaire**
- On cherche à maximiser le **total** des récompenses
- On dénote par A_t l'action **choisie** à l'étape t
- Meilleure **stratégie**?

Estimation

- Pour toute action a, la **qualité** de a est définie par

$$q_*(a) = \mathbb{E}[R_t \mid A_t = a]$$

- Idéalement, on devrait choisir a telle que $q_*(a)$ est **optimale**
- Mais on ne connaît pas q_*
- On va donc chercher à **estimer** cette valeur
- Soit $Q_t(a)$ l'évaluation de $q_*(a)$ qu'on a au temps t
- Apprendre $q_*(a) \approx$ faire approcher $Q_t(a)$ de $q_*(a)$ quand $t \to t_{\max}$

Remarque

- R_t, A_t et Q_t sont des **variables aléatoires**
- Elles sont évidemment **dépendantes** l'une de l'autre
- Elles dépendent aussi de t

Calcul de la valeur d'une action

Estimation de Q_t

On prend

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} \chi(A_i = a) R_i}{\sum_{i=1}^{t-1} \chi(A_i = a)}$$

Autrement dit, on estime la moyenne Q_t par **échantillonnage**

Stratégie possible: ε-gloutonne (ε-*greedy*)

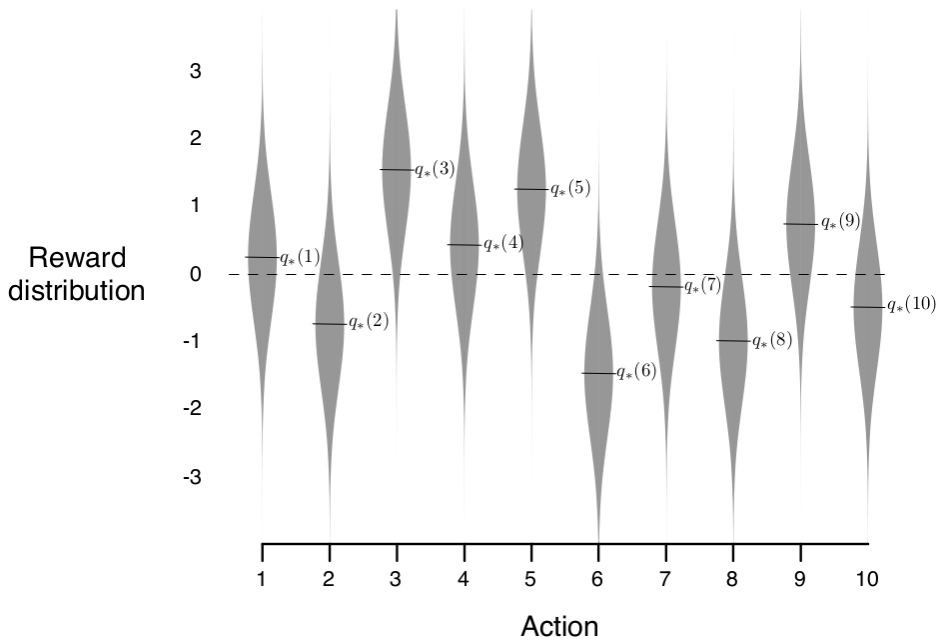
- Avec probabilité ε, on choisit une action au **hasard**
- Avec probabilité $1 - \varepsilon$, on choisit de façon **gloutonne**
- On peut **fixer** ε ou le faire **varier** dans le temps
- Permet d'équilibrer **exploitation** et **exploration**

Pseudocode

1: **fonction** BANDIT
2: $(q(a), n(a)) \leftarrow (0, 0)$ pour toute action a
3: **pour** $t \in \{1, 2, \ldots, T\}$ **faire**
4: $p \leftarrow \text{UNIFORME}(0, 1)$
5: **si** $p < \varepsilon$ **alors**
6: Choisir a au hasard
7: **sinon**
8: $a \leftarrow \text{argmax}_a q(a)$
9: Soit r la récompense retournée après avoir choisi a
10: $n(a) \leftarrow n(a) + 1$
11: $q(a) \leftarrow q(a) + (r - q(a))/n(a)$

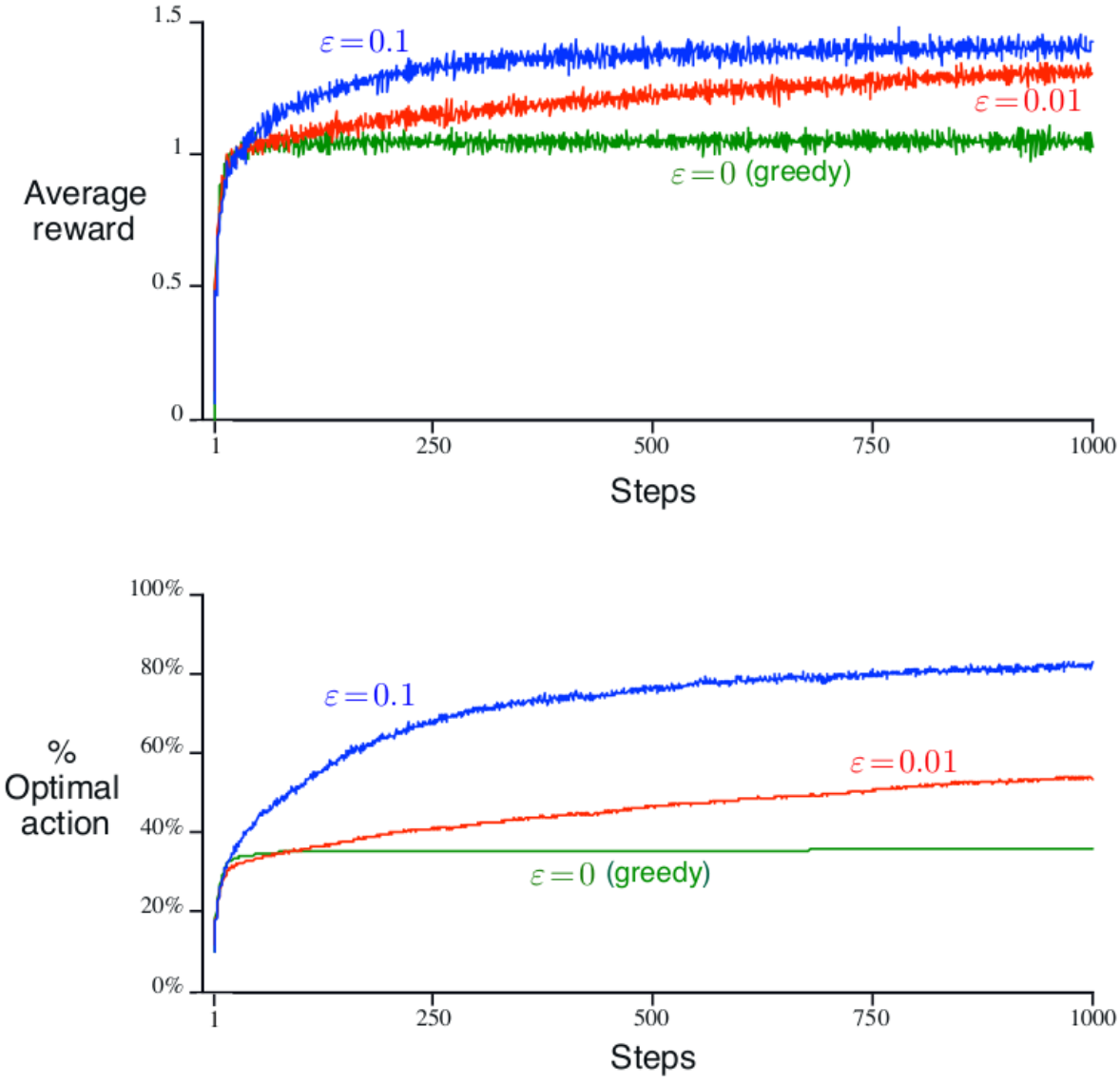
Jeu de données pour tests

- **Génération aléatoire** de $2\,000$ exemplaires
- En fixant $k = 10$ actions possibles
- En choisissant, pour $i = 1, 2, \ldots, k$, le paramètre μ_k selon $\mathcal{N}(0, 1)$
- Puis en posant $R_t \sim \mathcal{N}(\mu_k, 1)$



(Source: Sutton et Barto, chapitre 2)

Simulation



(Source: Sutton et Barto, chapitre 2)

Mise à jour de la valeur

1: **fonction** BANDIT
2: $(q(a), n(a)) \leftarrow (0, 0)$ pour toute action a
3: **pour** $t \in \{1, 2, \ldots, T\}$ **faire**
4: $p \leftarrow \text{UNIFORME}(0, 1)$
5: **si** $p < \varepsilon$ **alors**
6: Choisir a au hasard
7: **sinon**
8: $a \leftarrow \text{argmax}_a q(a)$
9: Soit r la récompense retournée après avoir choisi a
10: $n(a) \leftarrow n(a) + 1$
11: $q(a) \leftarrow q(a) + (r - q(a))/n(a)$

– On peut généraliser la **mise à jour**

Mise à jour de la valeur

1: **fonction** BANDIT
2: $(q(a), n(a)) \leftarrow (0, 0)$ pour toute action a
3: **pour** $t \in \{1, 2, \ldots, T\}$ **faire**
4: $p \leftarrow \text{UNIFORME}(0, 1)$
5: **si** $p < \varepsilon$ **alors**
6: Choisir a au hasard
7: **sinon**
8: $a \leftarrow \text{argmax}_a q(a)$
9: Soit r la récompense retournée après avoir choisi a
10: $n(a) \leftarrow n(a) + 1$
11: $q(a) \leftarrow q(a) + \alpha_n(r - q(a))$

- On peut généraliser la **mise à jour**
- En autant que α_n respecte certaines **contraintes**

Convergence

Théorème

L'algorithme converge avec probabilité 1 si

$$\sum_{n \geq 1} \alpha_n(a) = \infty \quad \text{et} \quad \sum_{n \geq 1} \alpha_n^2(a) < \infty$$

Cas non stationnaire

- Préférable **« d'oublier »** partiellement les récompenses passées
- On prend α **constant**
- Mais ne **converge pas**

Exemple 2: le robot recycleur

- Un robot qui se **déplace** dans une pièce
- De façon **autonome**
- **Unique tâche**: récupérer les canettes vides
- Il est alimenté par une **batterie rechargeable**
- On souhaite maximiser le nombre de canettes **récupérées**
- En minimisant le nombre de **recharges** manuelles et autonomes

États possibles

- `high`: la batterie est *chargée*
- `low`: la batterie est *faible*

Actions possibles

- `search`: le robot recherche une canette
- `wait`: le robot attend que quelqu'un lui donne une canette
- `recharge`: le robot recharge sa batterie de façon autonome

Paramètres

- α: probabilité de rester dans l'état `high`
- β: probabilité de rester dans l'état `low`
- $r_{\texttt{search}}$: la récompense espérée quand le robot cherche
- $r_{\texttt{wait}}$: la récompense espérée quand le robot attend
- $r_{\texttt{recharge}}$: récompense (ou pénalité) encourue lors d'une recharge manuelle

Contraintes

- $r_{\texttt{search}} > r_{\texttt{wait}} > 0$: plus intéressant de chercher que d'attendre
- $r_{\texttt{recharge}} < 0$: pénalisant de recharger la batterie manuellement

Conséquences

- $1 - \alpha$: probabilité de passer de `high` vers `low`
- $1 - \beta$: probabilité que la batterie se décharge

Graphe

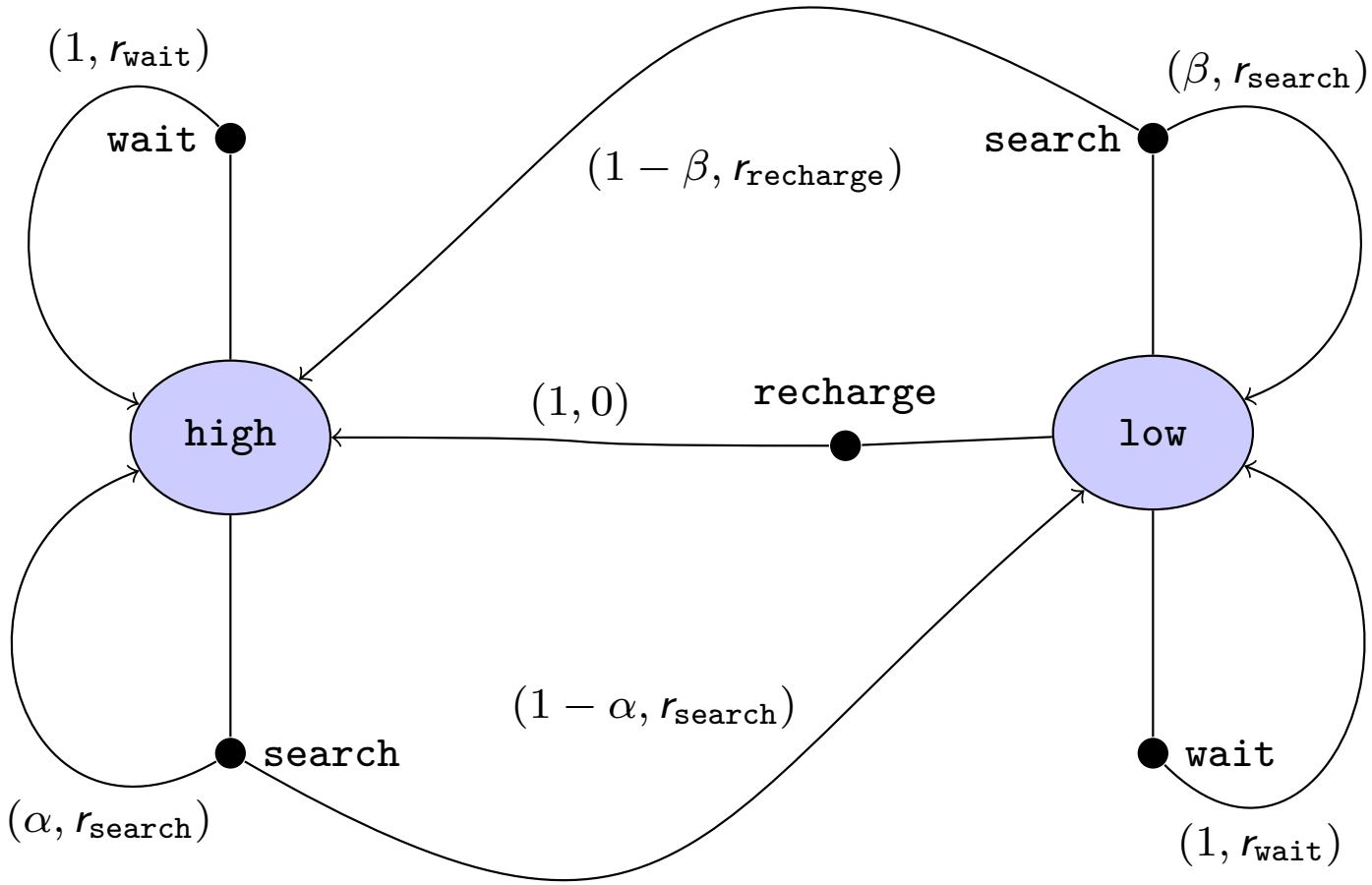


Image adaptée du livre de Sutton et Barto (chapitre 3)

Tableau de transition (complet)

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1-\alpha$	$r_{\texttt{search}}$
low	search	high	$1-\beta$	$r_{\texttt{recharge}}$
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	—
low	wait	high	0	—
low	wait	low	1	$r_{\texttt{wait}}$
high	recharge	high	0	—
high	recharge	low	0	—
low	recharge	high	1	0
low	recharge	low	0	—

Tableau de transition (compact)

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1 - \alpha$	$r_{\texttt{search}}$
low	search	high	$1 - \beta$	$r_{\texttt{recharge}}$
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0

Exemple 3: déplacement sur une grille

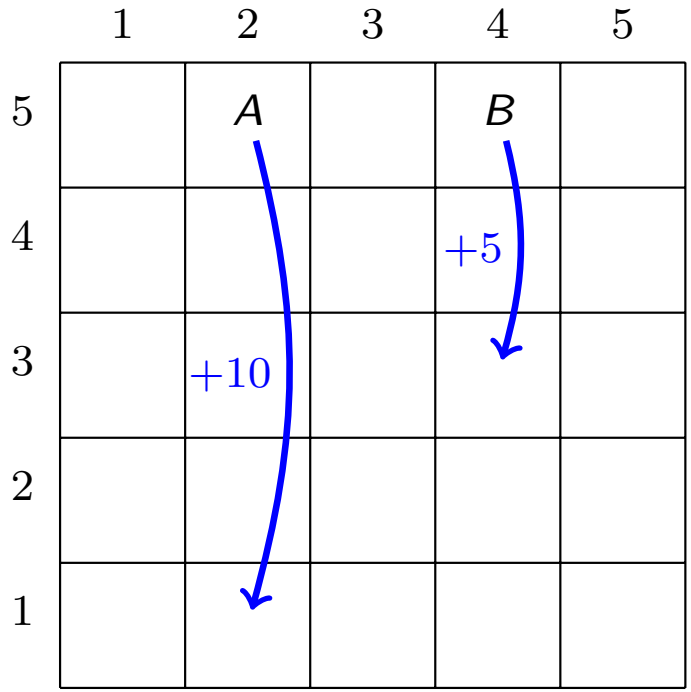


Image adaptée du livre
de Sutton et Barto
(chapitre 3)

États: chacune des 25 cases

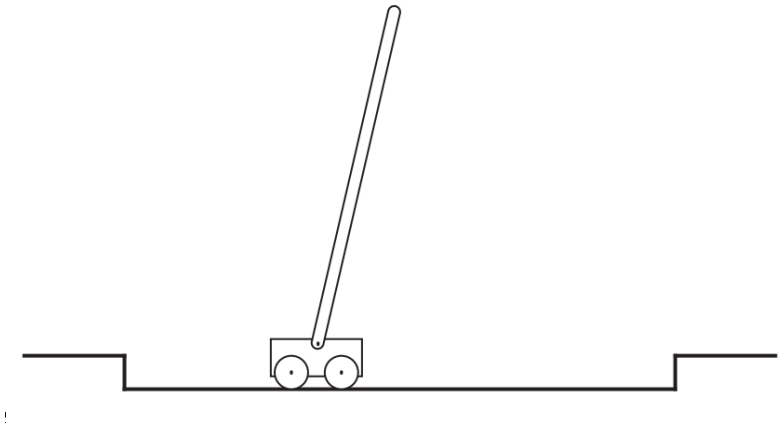
Actions: `est`, `nord`, `ouest` et `sud`

- *Cas général*: change selon l'action
- *Hors grille*: ne change pas d'état
- *État spécial A*: au bout de la flèche
- *État spécial B*: au bout de la flèche

Récompenses:

- *Hors grille*: -1
- *État spécial A*: $+10$
- *État spécial B*: $+5$
- *Autrement*: 0

Exemple 4: équilibre d'une tige (*pole balancing*)



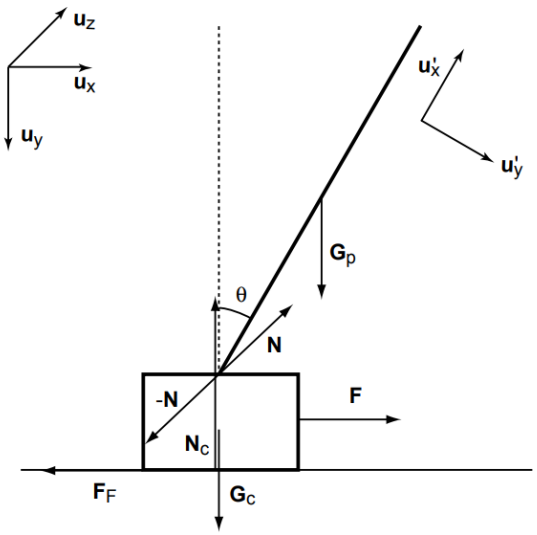
Source: Sutton et Barto

États (*sans friction*): position, vitesse et accélération angulaire de la tige $(\theta, \theta', \theta'')$
États (*avec friction*): position, vitesse et accélération angulaire de la tige, vitesse et accélération du chariot $(x', x'', \theta, \theta', \theta'')$
Actions: `droite`, `gauche`

Stratégies de récompenses
Possibilité 1 (épisodique)
- *Si la tige ne tombe pas*: $+1$
- *Si la tige tombe*: $+0$

Possibilité 2 (continue)
- *Si la tige ne tombe pas*: $+0$
- *Si la tige tombe*: -1

Mise à jour de l'état (sans friction)



g	:	constante gravitationnelle
ℓ	:	demi-longueur de la tige
m_p	:	masse de la tige
m_c	:	masse du chariot
F	:	force appliquée sur le chariot

Source: R.V. Florian

$$\theta''_{t+1} = \frac{g \sin\theta_t + \cos\theta_t \left(\frac{-F - m_p \ell \theta'^2_t \sin\theta_t}{m_c + m_p} \right)}{\ell \left(\frac{4}{3} - \frac{m_p \cos^2\theta_t}{m_c + m_p} \right)}$$

$$\theta'_{t+1} = \theta'_t + \Delta t \theta''_{t+1}$$

$$\theta_{t+1} = \theta_t + \Delta t \theta'_{t+1}$$

Exemple 5: jeux de plateau

- **États**: configurations de l'échiquier, de la grille ou du plateau de jeu
- **Actions**: « déplacement » ou « coup »
- **Récompenses**:
  - $+1$ quand on gagne
  - 0 quand on fait une nulle
  - -1 quand on perd
  - $-p$ quand on fait un coup illégal
- **Politique**: stratégie
- **V-fonction**: estimation de la « valeur » d'un état
- **Q-fonction**: estimation de la « qualité » d'un coup

Problème

- Explosion **combinatoire** de la taille de l'espace
- On doit entraîner « intelligemment »

En résumé

Espace d'états:

- *un seul état*: bandit manchot
- *fini petit*: robot recycleur, grille
- *fini très grand*: jeux de plateau
- *continu*: chariot-tige
- *discret infini* aussi possible

Actions:

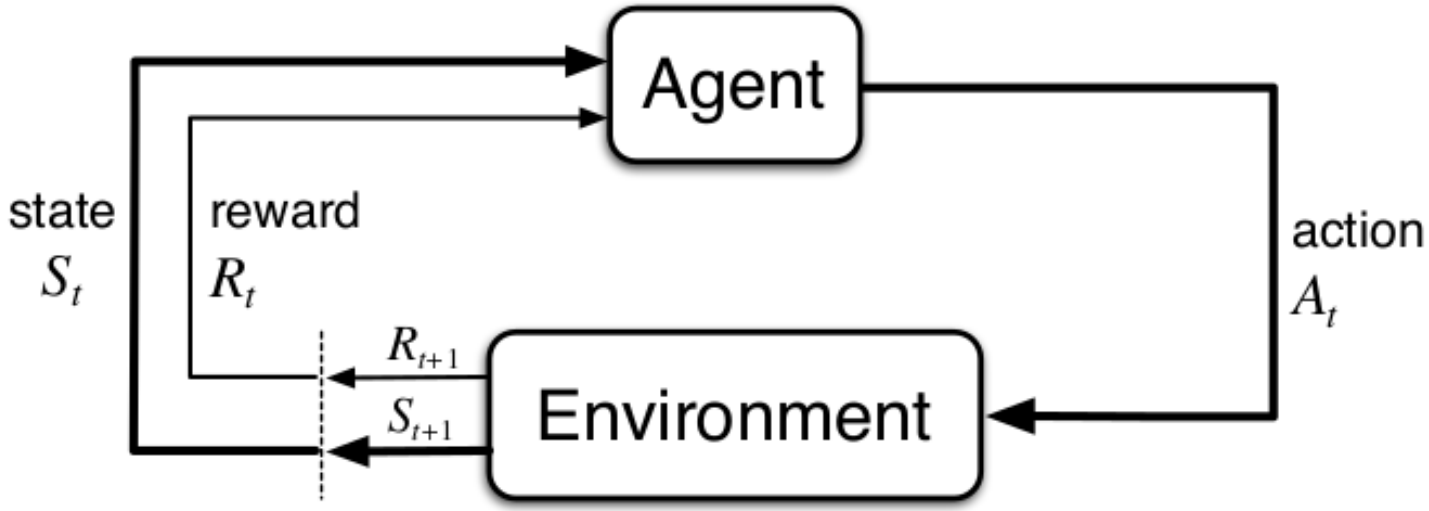
- Souvent un nombre **fini**, pas **trop grand**
- Quand continues, souvent **discrétisées**

Récompenses:

- **Déterministe**: grille, jeux de plateau, chariot-tige
- **Stochastique**: bandit manchot, robot recycleur

Processus de décisions markoviens

Interaction agent-environnement



(Source: Sutton et Barto, chapitre 3)

- t: **étape** ou **moment**
- $\mathcal{S}$: ensemble de tous les **états** possibles
- $\mathcal{R} \subseteq \mathbb{R}$: ensemble de toutes les **récompenses** possibles
- $\mathcal{A}$: ensemble de toutes les **actions** possibles
- On se concentre sur le cas **fini**: $|\mathcal{S}|, |\mathcal{R}|, |\mathcal{A}| < \infty$

Définition

Un **processus de décision markovien** fini est la donnée

- d'un ensemble fini d'**états** $\mathcal{S}$
- d'un ensemble fini d'**actions** $\mathcal{A}$
- d'un ensemble fini de **récompenses** $\mathcal{R}$
- d'une **suite d'états** $\{S_t\}_{t\geq 0}$
- d'une **suite d'actions** $\{A_t\}_{t\geq 0}$
- d'une **suite de récompenses** $\{R_t\}_{t\geq 1}$
- d'une fonction de **transition** (*dynamics*)

$$p : (\mathcal{S} \times \mathcal{R}) \times (\mathcal{S} \times \mathcal{A}) \to [0, 1]$$

définie par

$$p(s', r \mid s, a) = \mathbb{P}(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a)$$

- ayant la **propriété de Markov**

Propriété de Markov

Pour tout t, S_t et R_t ne **dépendent** que de S_{t-1} et A_{t-1}:

$$\begin{aligned} p(s', r \mid s, a) &= \mathbb{P}(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a) \\ &= \mathbb{P}(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a, \\ &\qquad S_{t-2} = s_{t-2}, A_{t-2} = a_{t-2}, \\ &\qquad \ldots, \\ &\qquad S_1 = s_1, A_1 = a_1) \end{aligned}$$

Ainsi,

- pour **calculer** la probabilité de se trouver à S_t et d'avoir une récompense R_t
- nous n'avons **pas besoin** de connaître les états S_{t-2}, S_{t-3}, ... ni les récompenses R_{t-2}, R_{t-3}, ...

Remarques et notation

– Pour tous $s \in \mathcal{S}$ et $a \in \mathcal{A}$, on a

$$\sum_{s',r} p(s', r \mid s, a) = 1$$

– On **étend** la fonction de dynamique p:

$$p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$$

en posant

$$p(s' \mid s, a) = \mathbb{P}(S_t = s' \mid S_{t-1} = s, A_{t-1} = a) = \sum_r p(s', r \mid s, a)$$

Récompense espérée (1/2)

- On introduit une fonction indiquant la **récompense espérée**:

$$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$

définie par

$$r(s, a) = \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a]$$

- Par **définition** de l'espérance mathématique, on a donc

$$r(s, a) = \sum_r r \sum_{s'} p(s', r \mid s, a)$$

Récompense espérée (2/2)

– La fonction r s'étend naturellement à

$$r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$$

en posant

$$r(s, a, s') = \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s']$$

– Par conséquent, on trouve

$$r(s, a, s') = \sum_r r \, \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}$$

Retour espéré

Rappel

Suite de **récompenses**: R_1, R_2, ..., $R_{t_{\max}}$

Retour (ou gain) espéré

Pour $t = 0, 1, \ldots, t_{\max} - 1$, on définit le **retour espéré** par

$$G_t = R_{t+1} + R_{t+2} + \ldots + R_{t_{\max}}$$

Si le processus est **continu** (épisode infini), c'est plutôt

$$G_t = R_{t+1} + R_{t+2} + \ldots$$

Interprétation

Récompense **totale** qu'on espère avoir à partir de maintenant

Processus continu

- Quand $t_{\text{max}} \to \infty$
- Le **retour espéré** $G_t = R_{t+1} + R_{t+2} + \ldots$ peut tendre vers ∞

Facteur d'actualisation

- On souhaite **éviter** cette complication
- On introduit un **paramètre** γ, avec $0 \leq \gamma \leq 1$
- Appelé **facteur d'actualisation** (en anglais, *discount*)
- **Analogie**: inflation
- Et on définit plutôt

$$\begin{aligned} G_t &= \sum_{k=0}^{t_{\text{max}}} \gamma^k R_{t+k+1} \\ &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \ldots \end{aligned}$$

Facteur d'actualisation

- $\gamma = 0$: l'agent est « myope », ne se préoccupe que des récompenses immédiates
- $\gamma \to 1$: vision à long terme
- Pour assurer la **convergence** de

$$G_t = \sum_{k=0}^{t_{\text{max}}} \gamma^k R_{t+k+1}$$

on impose que $t_{\text{max}} \neq \infty$ **ou** $\gamma < 1$

Notation uniforme

- Le paramètre $t_{\max}$ peut être **fini** ou **infini**
- Pour uniformiser, dans le cas fini, on pose $R_t = 0$ pour $t \geq t_{\max}$
- Et donc on peut poser, pour tout $t \geq 0$,

$$G_t = \sum_{t \geq 0} \gamma^k R_{t+k+1}$$

Expression récursive

- On remarque que

$$\begin{aligned} G_t &= \sum_{k\geq 0} \gamma^k R_{t+k+1} \\ &= R_{t+1} + \sum_{k\geq 1} \gamma^k R_{t+k+1} \\ &= R_{t+1} + \gamma \sum_{k\geq 0} \gamma^k R_{t+k+2} \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

- Il suffit donc de calculer le retour à partir de la **fin**

Politique

- Une **politique** est une fonction

$$\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$$

définie par

$$\pi(a \mid s) = \mathbb{P}(A_t = a \mid S_t = s)$$

pour tout t

- La politique π est dite **déterministe** si pour chaque $s \in S$, il existe $a \in A$ telle que

$$\pi(a \mid s) = 1 \quad \text{et} \quad \pi(a' \mid s) = 0, \text{pour tout } a' \neq a$$

- Sinon, elle est dite **stochastique**

Valuation d'un état

- Soit π une **politique**
- On définit la **valuation** (en anglais, *state-value function*) d'un état par

$$v_\pi : \mathcal{S} \to \mathbb{R}$$

en posant

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

pour tout t, où $\mathbb{E}_\pi$ est l'espérance mathématique en supposant que l'agent **applique** la politique π
- Ainsi, pour tout $s \in \mathcal{S}$,

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k \geq 0} \gamma^k R_{t+k+1} \,\middle|\, S_t = s \right]$$

Qualité d'une paire action-état

- Soit π une **politique**
- De la même façon, on définit la **qualité** (en anglais, *action-value function*) d'une paire action-état par

$$q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$

en posant

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

pour tout t

- On a donc, pour tout $s \in \mathcal{S}$,

$$q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k \geq 0} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a \right]$$

Équations de Bellman

- On aimerait trouver une expression **récursive** pour $v_\pi(s)$ en fonction des états **successeurs** de s
- Même chose pour $q_\pi(s, a)$
- On peut montrer que

$$v_\pi(s) = \sum_{r,a,s'} \pi(a \mid s)\ p(s', r \mid s, a)\ \big(r + \gamma v_\pi(s')\big)$$

- De façon similaire, on trouve

$$q_\pi(s, a) = \sum_{r,a,s'} p(s', r \mid s, a) \left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a') \right]$$

- Les équations ci-haut sont **linéaires**
  - une variable par état pour v_π
  - une variable par paire état-action pour q_π
- Et elles admettent chacune une solution **unique**

Relations entre v_π et q_π

- Il est possible de calculer v_π en fonction de q_π

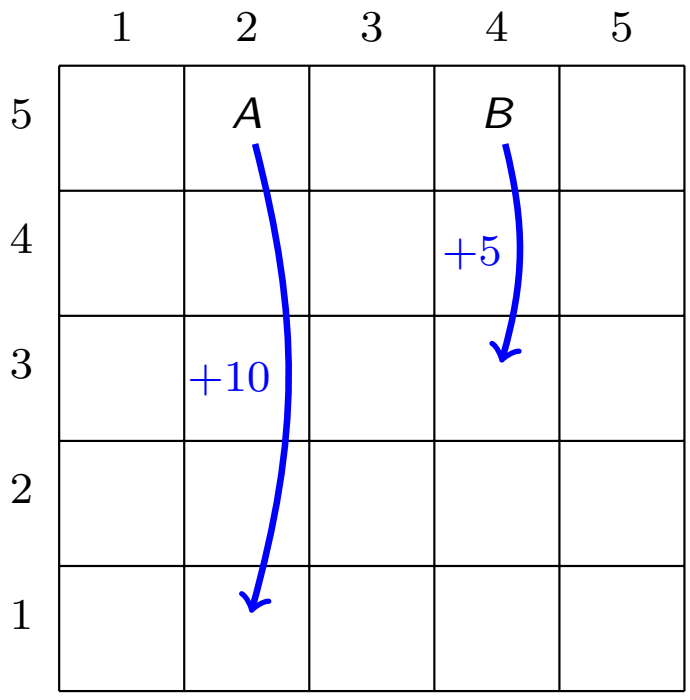
$$v_\pi(s) = \sum_a \pi(a \mid s)\ q_\pi(s, a)$$

- Et, inversement, q_π en fonction de v_π

$$q_\pi(s, a) = \sum_{r,s'} p(s', r \mid s, a)(r + \gamma v_\pi(s'))$$

Retour sur la grille

Environnement



Fonction de valuation v_π

	1	2	3	4	5
5	3.3	8.8	4.4	5.3	1.5
4	1.5	3.0	2.3	1.9	0.5
3	0.1	0.7	0.7	0.4	−0.4
2	−1.0	−0.4	−0.4	−0.6	−1.2
1	−1.9	−1.3	−1.2	−1.4	−2.0

Avec $\gamma = 0.9$ et quatre actions **équiprobables**

Valeurs tronquées à la position des **dixièmes**

(Images adaptées du livre de Sutton et Barto, chapitre 3)

Politique optimale

- Considérons un processus de décision **markovien**
- Avec la même notation ($\mathcal{S}$, $\mathcal{A}$, p, etc.)
- Soit Π l'ensemble de **toutes** les politiques pour ce processus
- On définit une relation **d'ordre partiel** sur Π par

$$\pi \geq \pi'$$

si et seulement si

$$v_\pi(s) \geq v_{\pi'}(s), \quad \text{pour tout } s \in \mathcal{S}$$

- Toute politique π qui est maximale pour la relation $\geq$ est dite **optimale**
- On écrit alors

$$v_*(s) = \max_{\pi \in \Pi} v_\pi(s) \quad \text{et} \quad q_*(s,a) = \max_{\pi \in \Pi} q_\pi(s,a)$$

Équations d'optimalité de Bellman

- Soit π_* une politique **optimale**
- On peut montrer que

$$v_*(s) \quad = \quad \max_a \sum_{r,s'} p(s', r \mid s, a)[r + \gamma \; v_*(s')]$$

- De façon similaire,

$$q_*(s, a) = \sum_{r,s'} p(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]$$

- Les équations ne sont **pas linéaires**
- Car elles utilisent la fonction max
- En revanche, la solution est **unique** pour v_π et q_π
- On construit facilement une **politique optimale** à partir de là
  - pour **chaque état**
  - on vérifie quelles sont les **actions optimales**
  - on assigne une probabilité **nulle** à chaque action non optimale
  - on répartit le poids 1 entre les **actions optimales**

Retour sur le robot recycleur (1/2)

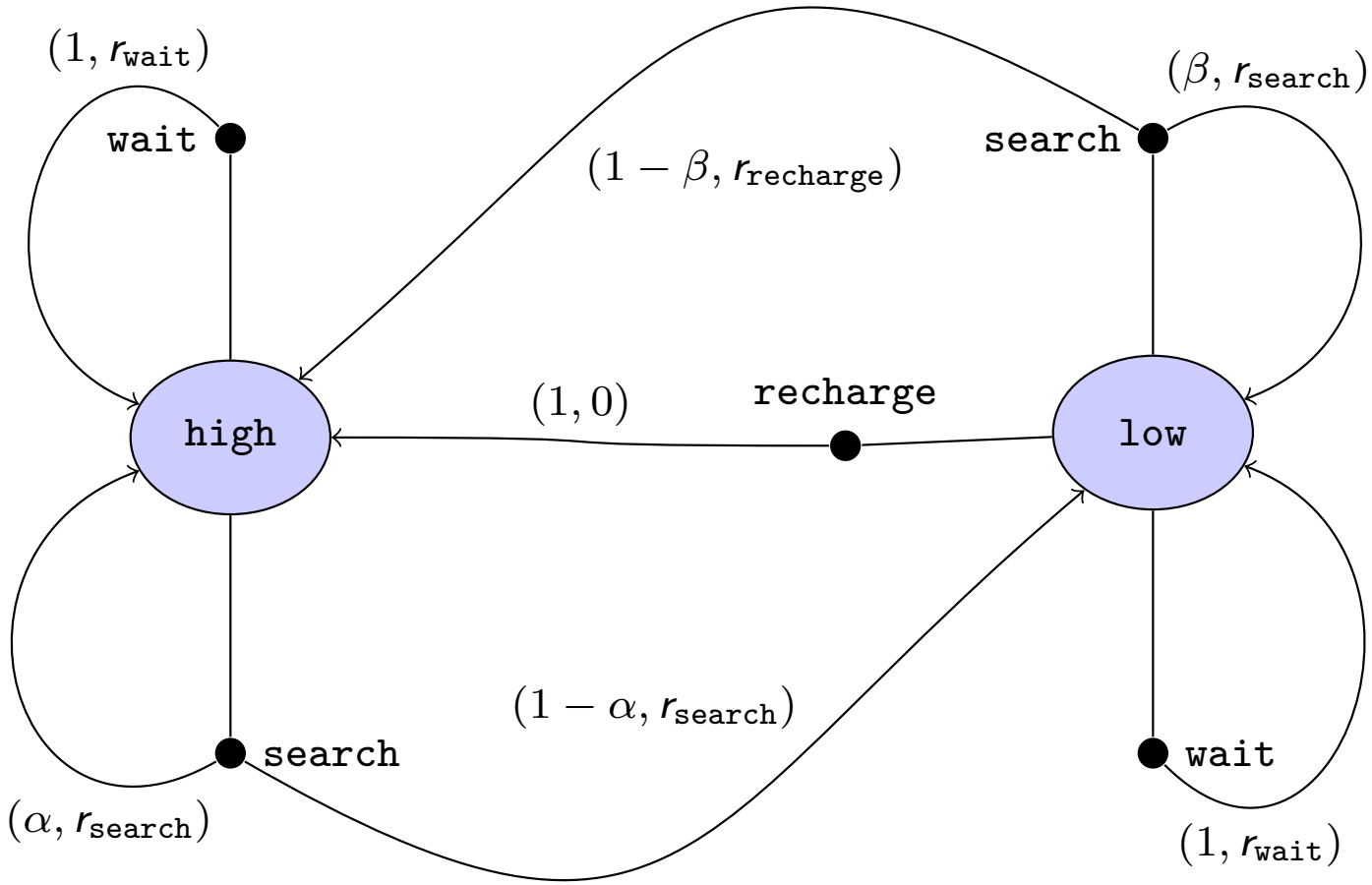


Image adaptée du livre de Sutton et Barto (chapitre 3)

Retour sur le robot recycleur (2/2)

– Les équations à résoudre sont

$$
\begin{aligned}
v_*(\texttt{high}) &= \max\{r_{\texttt{search}} + \gamma[\alpha v_*(\texttt{high}) + (1-\alpha) v_*(\texttt{low})], \\
&\qquad r_{\texttt{wait}} + \gamma v_*(\texttt{high})\} \\
v_*(\texttt{low}) &= \max\{\beta r_{\texttt{search}} + r_{\texttt{recharge}}(1-\beta) \\
&\qquad + \gamma[(1-\beta) v_*(\texttt{high}) + \beta v_*(\texttt{low})], \\
&\qquad r_{\texttt{wait}} + \gamma v_*(\texttt{low}), \\
&\qquad \gamma v_*(\texttt{high})\}
\end{aligned}
$$

– La solution est **unique** lorsque les paramètres sont fixés

En pratique

Coûteux

- On résoud **rarement** les équations d'optimalité de Bellman
- Trop coûteux en **temps de calcul**
- Et en **espace mémoire**

Méthodes plus avancées

- Simulation de **Monte-Carlo**
- Méthode des **différences temporelles**
- Q-**apprentissage**
- Recherche arborescente de **Monte-Carlo**
- Ignorer les états **peu probables**
- Approximation de **fonctions**
- Q-apprentissage **profond**

Méthodes tabulaires

Méthodes tabulaires (1/2)

Tabulaire

- On stocke $V(s)$ et $Q(s, a)$ dans des **tableaux**
- Si $\mathcal{S}$ et $\mathcal{S} \times \mathcal{A}$ sont trop **grands**?
- Problème de **mémoire**
- On peut **hacher** les valeurs, par exemple
- Mais aussi de **temps**: pour *remplir* les valeurs

Approximation de fonctions

- Fonctions **linéaires**
- Réseaux de **neurones**

Méthodes tabulaires (2/2)

1: **fonction** AlphaZero()
2: Initialiser $f_\theta : \mathcal{S} \to \Pi \times \mathbb{R}$
3: **tant que** *vrai* **faire**
4: JouerPartie()
5: MettreAJour(θ)
6:
7: **fonction** JouerPartie()
8: Soit S l'état initial
9: **tant que** la partie n'est pas finie **faire**
10: $\pi \leftarrow MCST(S, f_\theta)$
11: Enregistrer (S, π)
12: Mettre à jour S à partir de π
13:
14: **fonction** MettreAJour(θ)()
15: Soit $z \in \{-1, 0, +1\}$ le résultat de JouerPartie() pour (S, π)
16: Entraîner f_θ sur (S, π, z) par descente du gradient

Programmation dynamique (1/2)

Hypothèse

Dynamique ($p(s', r \mid s, a)$) **complètement connue**

Calcul itératif de la valuation v_π (*iterative policy evaluation*)

- Soit π une politique **fixée**
- On choisit v_0 de façon **arbitraire**
- Puis on calcule, v_{k+1} à partir de v_k:

$$v_{k+1}(s) = \sum_{r,a,s'} \pi(a \mid s)\ p(s', r \mid s, a)\ \left(r + \gamma v_k(s')\right)$$

- On a $v_k \to v_\pi$ quand $k \to +\infty$
- **Numériquement**: on itère jusqu'à ce que $|v_k(s) - v_{k+1}(s)| \leq \Delta$, pour un certain seuil Δ et pour tous $s \in \mathcal{S}$

Programmation dynamique (2/2)

Amélioration de la politique (*policy improvement*)

- Soit π **fixée**
- On calcule v_π
- Puis on définit π' par

$$\pi'(s) = \text{argmax}_a \sum_{s',r} p(s', r \mid s, a) \left(r + \gamma v_\pi(s')\right)$$

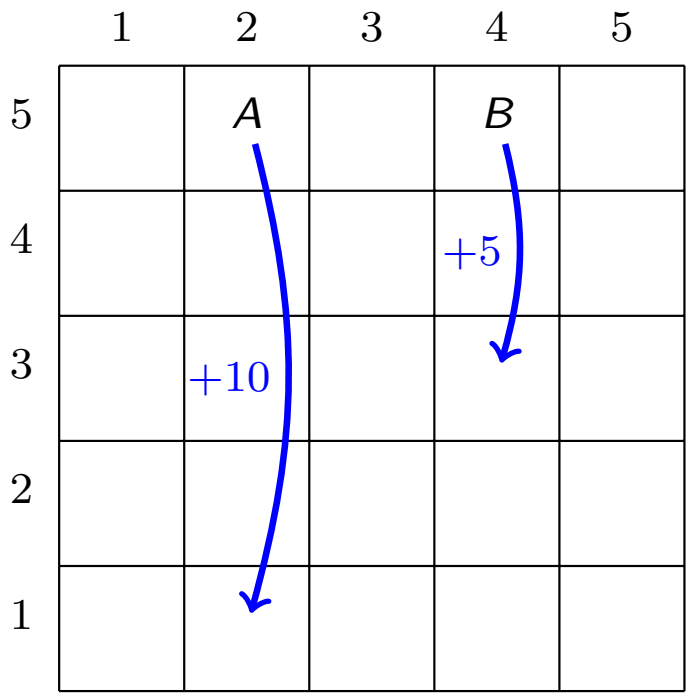
- La politique est **optimale** si $v_\pi = v_{\pi'}$

Calcul itératif de la politique (*policy iteration*)

- On peut répéter **calcul de valuation** et **amélioration de politique**
- Mais aussi ne pas **attendre** que $v_k \to v_\pi$
- Toute stratégie qui **alterne** ces deux tâches est appelée **calcul itératif généralisé de politique** (*generalized policy iteration*)

Retour sur la grille (1/2)

Environnement



Fonction de valuation v_π

	1	2	3	4	5
5	3.3	8.8	4.4	5.3	1.5
4	1.5	3.0	2.3	1.9	0.5
3	0.1	0.7	0.7	0.4	−0.4
2	−1.0	−0.4	−0.4	−0.6	−1.2
1	−1.9	−1.3	−1.2	−1.4	−2.0

Avec $\gamma = 0.9$ et quatre actions **équiprobables**

Valeurs tronquées à la position des **dixièmes**

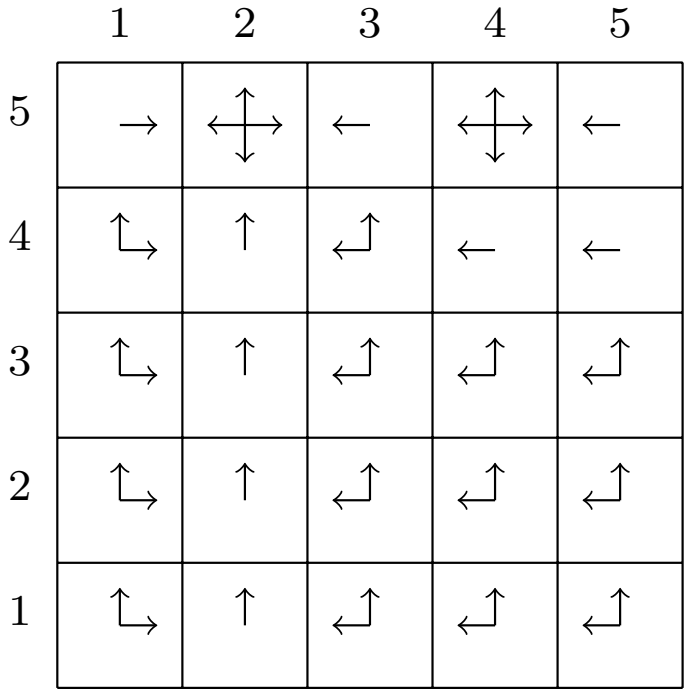
(Images adaptées du livre de Sutton et Barto, chapitre 3)

Retour sur la grille (2/2)

Valuation optimale v_*

	1	2	3	4	5
5	22.0	24.4	22.0	19.4	17.5
4	19.8	22.0	19.8	17.8	16.0
3	17.8	19.8	17.8	16.0	14.4
2	16.0	17.8	16.0	14.4	13.0
1	14.4	16.0	14.4	13.0	11.7

Politique optimale π_*



Valeurs tronquées à la position des **dixièmes**

(Images adaptées du livre de Sutton et Barto, chapitre 3)

Dynamique non connue?

Question

- Qu'arrive-t-il si $p(s', r \mid s, a)$ n'est **pas connue**?
- Ou **trop complexe**?
- Mais qu'on a une fonction **stochastique** de déplacement

$$(s', r) \leftarrow \textsc{Move}(s, a)?$$

Réponses possibles

- Simulation de **Monte-Carlo**
- Différence **temporelles** ($TD = \textit{temporal difference}$)
- Apprentissage **TD(λ)**
- Recherche **arborescente** de Monte-Carlo

Simulations de Monte-Carlo (1/2)

Idée générale

- **Monte-Carlo**: utilisation répétée du hasard
- Dans l'espoir qu'on **converge** vers la « vraie valeur »

Un peu plus de détails

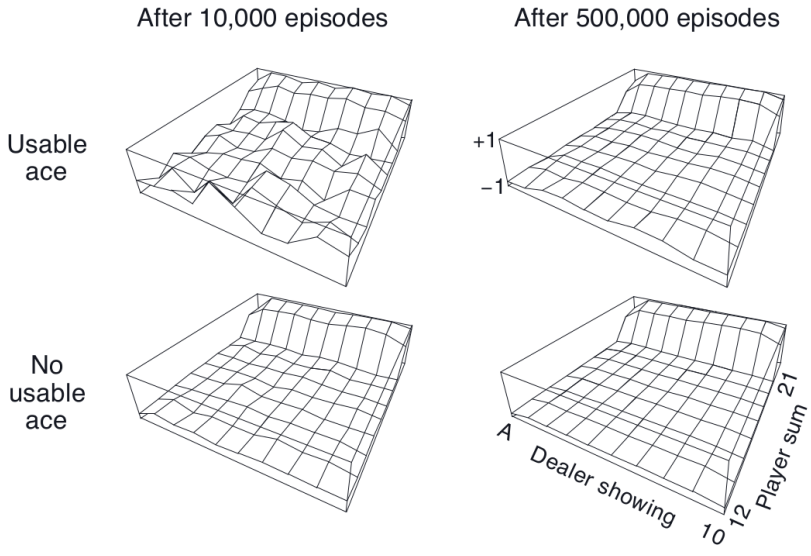
- Soit π une politique **fixée**
- On génère un **épisode**:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \ldots, S_{T-1}, A_{T-1}, R_T$$

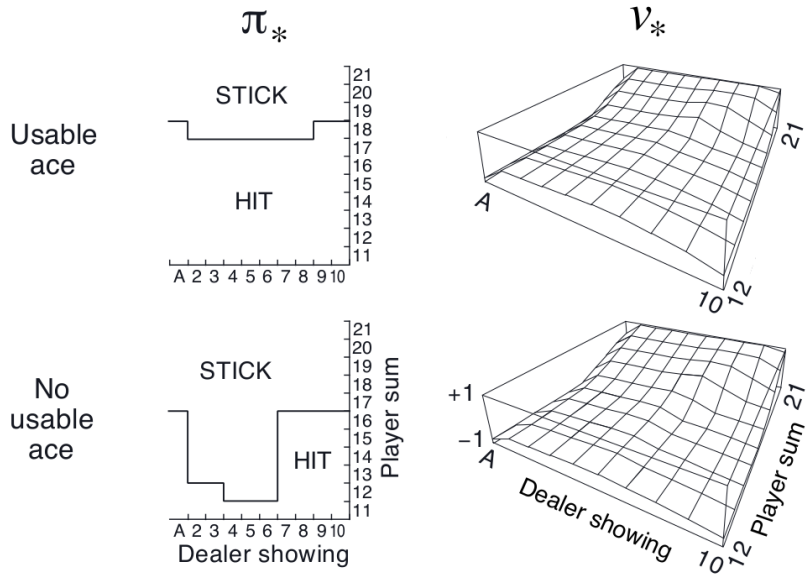
- Comme on va jusqu'à la fin, on a une estimation du **retour**
- Puis on **recule**: pour $t = T-1, T-2, \ldots, 0$,
  - on met à jour $G_t \leftarrow \gamma G_{t+1} + R_{t+1}$
  - on met à jour $V(S_t) \leftarrow \textsc{Moyenne}(V(S_t), G_t)$

Simulations de Monte-Carlo (2/2)

Application au jeu de **Blackjack**:



(a) Politique fixée 20-21



(b) Politique optimale

Source: Sutton et Barto

Différences temporelles (1/3)

Idée générale

- Mise à jour **au fur et à mesure**
- **Plutôt qu'à la fin** (Monte-Carlo)

Exemple: TD(0)

- Soit π une politique **fixée**
- On **démarre** à un état S
- On **sélectionne** une action A par rapport à π
- On **récupère** R et S' en appliquant A
- On **met à jour** $V(S_t) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
- Puis on **répète**

Différences temporelles (2/3)

1: **fonction** SARSA
2: **pour** chaque épisode **faire**
3: Initialiser S
4: Dériver une politique π de Q
5: Choisir A à partir de S et π
6: **pour** chaque étape de l'épisode **faire**
7: $(R, S') \leftarrow \text{MOVE}(S, A)$
8: Dériver une politique π de Q
9: Choisir A' à partir de S' et π
10: $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
11: $(S, A) \leftarrow (S', A')$

- α: taux d'apprentissage
- ε: seuil

Différences temporelles (3/3)

1: **fonction** Q-APPRENTISSAGE
2: **pour** chaque épisode **faire**
3: Initialiser S
4: **pour** chaque étape de l'épisode **faire**
5: Dériver une politique π de Q
6: $(R, S') \leftarrow \text{MOVE}(S, A)$
7: $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
8: $S \leftarrow S'$

Remarque

- SARSA: *on-policy*
- Q-apprentissage: *off-policy*

Variantes

Politiques

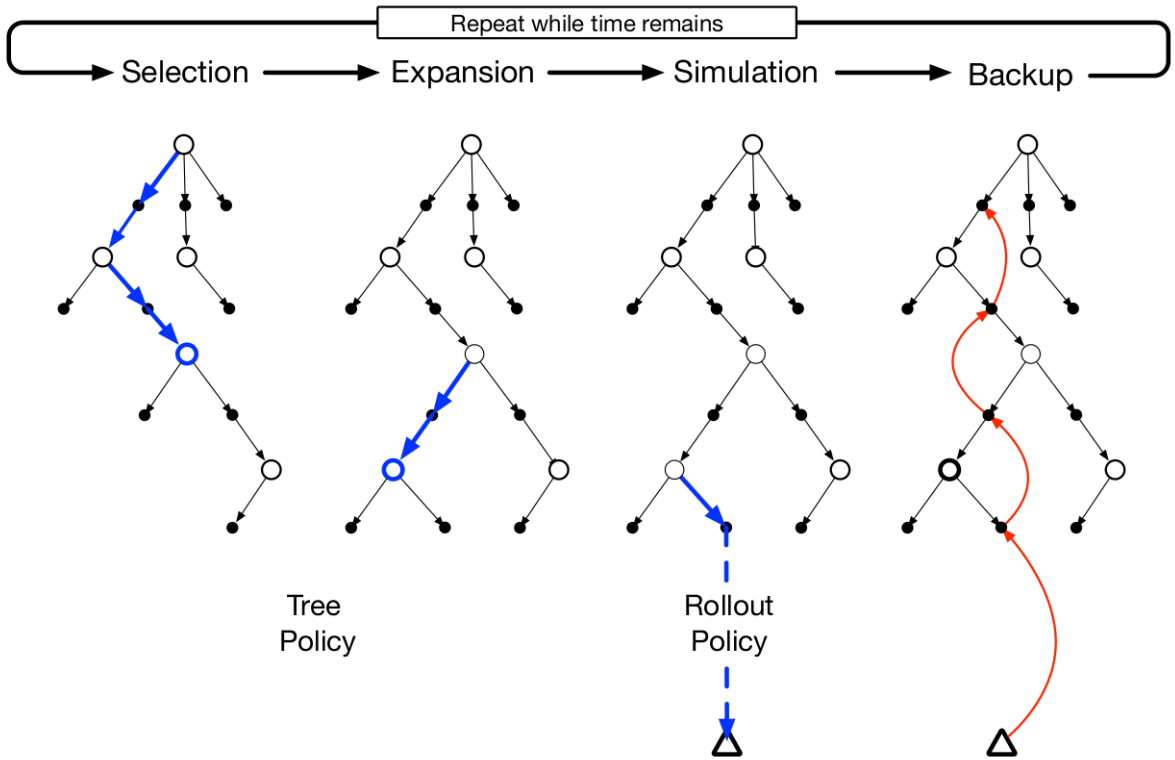
- ε-**flexibles**
- **Active** (*on-policy*) ou **non-active** (*off-policy*)
- **Exemple**: séparer politiques cible et comportementale

Apprentissage

- Biais du maximum
- Double apprentissage
- n-étapes

Recherche arborescente de Monte-Carlo

- En anglais, *MCST = Monte-Carlo search tree*
- **Deux politiques**: arborescence et déroulement
- On lance **plusieurs simulations** de Monte-Carlo
- Puis on passe au **noeud suivant** et on recommence



Et ensuite?

Conclusion

Si $\mathcal{S} \times \mathcal{A}$ est trop grand?

- Approximation de fonctions
- Comment **représenter** cette fonction?
- Fonctions **linéaires**? Réseaux de **neurones**?

AlphaStar (janvier 2019)

- *Off-policy*
- Auto-imitation
- Distillation de politique
- Rejeu d'expérience

MuZero (novembre 2019, décembre 2020)

- Généralisation de Alpha Zero
- Les règles sont **apprises**