Apprentissage par renforcement

Cours 10

Nicolas Baskiotis
nicolas.baskiotis@sorbonne-universite.fr

Master 1 DAC

équipe MLIA, Institut des Systèmes Intelligents et Robotique (ISIR))
Sorbonne Université

S2 (2021-2022)

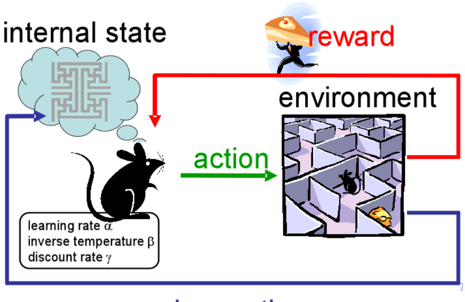
Plan

Principe

Lexique

- Agent, Environnement
- Etat (*state*) : ce que perçoit l'agent
- Action : une interaction de l'agent avec l'environnement
- Récompense (*reward*) : une quantité perçue après chaque action
- Politique (*policy*) : une fonction de sélection de l'action selon l'état

Objectif : trouver une politique qui permet de maximiser l'ensemble des récompenses reçues

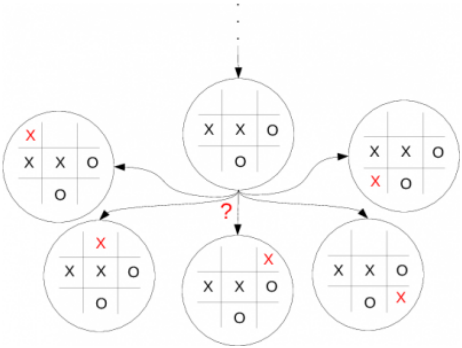


Objectif : adaptation du système à son environnement

Reproduction artificielle du comportement du “conditionnement”

Comment :

- enseigner un comportement à l’aide de récompenses ?
- réagir à une situation donnée ?
- agir de manière à maximiser les récompenses ?



Problématiques

- Comment représenter un système par des états, des actions et de récompenses ?
- Apprendre à évaluer une action en fonction de récompenses futures
- Explorer quand peu d'informations disponibles
- Exploiter pour rester dans des sénarios *viables*

On peut également chercher :

- à modéliser ou non l'environnement (model-free, model-based),
- apprendre online (en interagissant directement avec l'environnement)
- ou offline à partir de sénarios déjà joués.

Différence par rapport à l'apprentissage supervisé

- les exemples ne sont plus i.i.d. !!
- les exemples dépendent des actions précédentes, de la politique en cours

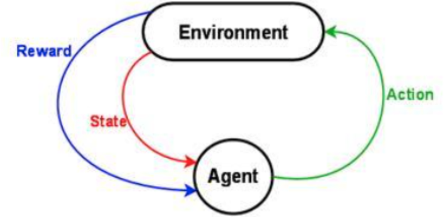
Plan

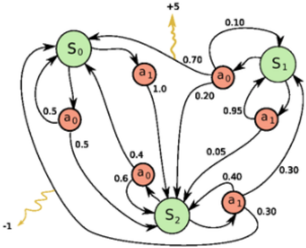
Markov Decision Process (MDP)

Définition du modèle

- $\mathcal{S}$: un espace d'états
- $\mathcal{A}$: un espace d'actions
- $T : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$: fonction de transition
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$: récompense

Hypothèse markovienne : la récompense et la fonction de transition ne dépendent que de l'état (et action) en cours, pas de l'historique.

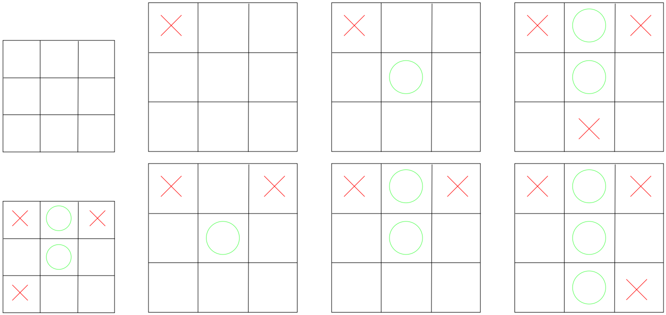




MDP : exemple

Jeu du Tic-Tac-Toe

- un état : une configuration de la grille du Jeu

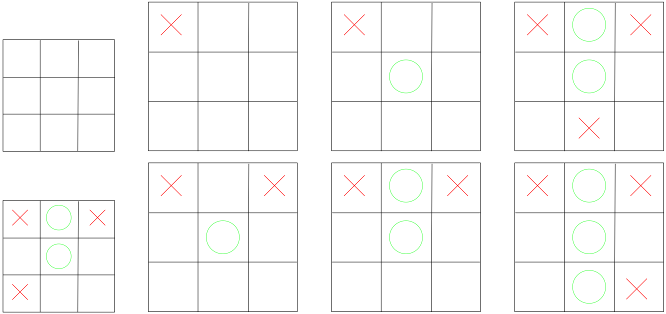


États discrets $\Rightarrow$ indexation de chaque état par un entier dans $\mathbb{N}$.
Nombre d'états (borne max): 3^9

MDP : exemple

Jeu du Tic-Tac-Toe

- un état : une configuration de la grille du Jeu
- une action $=$ jouer une case de la grille : 9 actions (et pas 18, on réfléchit par rapport à un joueur donné).

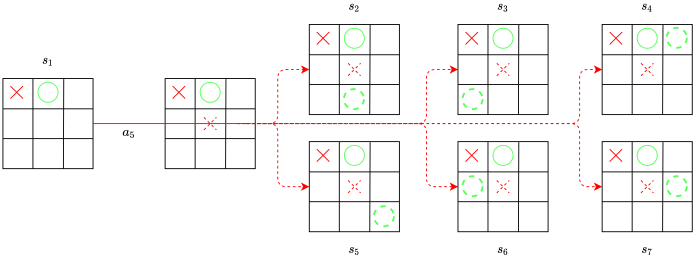


États discrets $\Rightarrow$ indexation de chaque état par un entier dans $\mathbb{N}$.
Nombre d'états (borne max): 3^9

MDP : exemple

Jeu du Tic-Tac-Toe

- un état : une configuration de la grille du Jeu
- une action $=$ jouer une case de la grille : 9 actions (et pas 18, on réfléchit par rapport à un joueur donné).
- fonction de transition : T ici non déterministe, l'état d'arrivée dépend de l'action du 2ème joueur

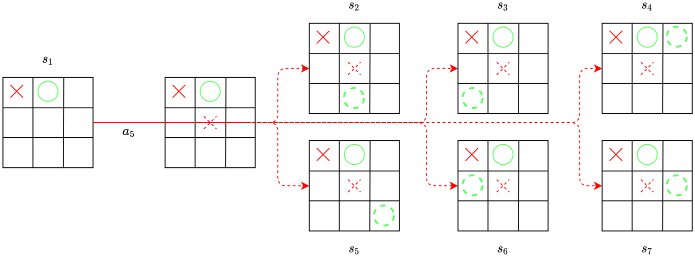


$$T(s_1, a_5) = \{s_2 : 0.1,\ s_3 : 0.1,\ s_4 : 0.1,\ s_5 : 0.4,\ s_6 : 0.2,\ s_7 : 0.1\}$$

MDP : exemple

Jeu du Tic-Tac-Toe

- un état : une configuration de la grille du Jeu
- une action $=$ jouer une case de la grille : 9 actions (et pas 18, on réfléchit par rapport à un joueur donné).
- fonction de transition : T ici non déterministe, l'état d'arrivée dépend de l'action du 2ème joueur
- une récompense : r qui représente la valeur de l'état d'arrivée (et uniquement de l'état d'arrivée, pas du futur !)



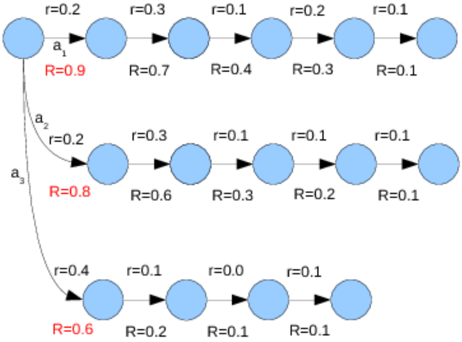
$$T(s_1, a_5) = \{s_2 : 0.1,\ s_3 : 0.1,\ s_4 : 0.1,\ s_5 : 0.4,\ s_6 : 0.2,\ s_7 : 0.1\}$$

Problèmatique

Comment choisir une action ?

- regarder la récompense liée à chaque action
- Mais aussi les récompenses futurs !

$\Rightarrow$ **fonction de valeur** d'états (ou d'état/action) : indication sur le long terme des récompenses attendues ($\neq$ récompenses immédiates)



Formalisation

Définitions

- Une politique associe à tout état s une action $\pi(s)$: $\pi : \mathcal{S} \rightarrow \mathcal{A}$ (ou dans le cas probabiliste dans $\Pi(\mathcal{A})$)
- Un scénario est une séquence d'état obtenu en suivant une politique : $[(s_1, a_1, r_1), (s_2, a_2, r_2), \ldots, (s_n, a_n, r_n)], \quad (s_i, a_i, r_i) \in \mathcal{S} \times \mathcal{A} \times \mathbb{R}$
- La récompense globale d'un scénario à partir d'un instant t_0 est $R_{t_0} = \sum_{t=0}^{T-t_0} \gamma^t r_{t_0+t}$, avec $0 < \gamma \leq 1$
- γ est une constante qui permet de prendre plus ou moins en compte les récompenses à long terme.

Fonctions *valeur*

- Les fonctions *valeur* d'une politique permettent de refléter la récompense à moyen terme $\rightarrow$ agrégation des récompenses
- Une fonction de valeur d'états : $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$
  $V^\pi(s) = \mathbb{E}_\pi(R_{t_0} | s_{t_0} = s) = \mathbb{E}_\pi \left[\sum_{t=t_0}^{T} \gamma^{t-t_0} r(s_t, \pi(s_t), s_{t+1}) \middle| s_{t_0} = s \right]$
- Une fonction de valeur d'actions : $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
  $Q^\pi(s, a) = \mathbb{E}_\pi(R_{t_0} | s_{t_0} = s, a_{t_0} = a)$

Plan

Equation de Bellman

$$V^\pi(s) = \mathbb{E}_\pi(R_{t_0}|s_{t_0} = s) = \mathbb{E}_\pi\left[\sum_{t=t_0}^{T} \gamma^{t-t_0} r(s_t, \pi(s_t), s_{t+1})|s_{t_0} = s\right]$$

Développement au prochain coup

- π et MDP déterministe :

$$V^\pi(s) = r(s, \pi(s), s') + \gamma V^\pi(s')$$

- MDP non déterministe :

$$V^\pi(s) = \sum_{s'} p(s'|s, \pi(s)) \left[r(s, \pi(s), s') + \gamma V^\pi(s')\right]$$

- Politique non déterministe :

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

- même relation pour $Q(s, a)$

Propriétés des fonctions valeurs

Politique et fonction valeur optimale :

- On dit que $\pi \geq \pi' \iff \forall s \in \mathcal{S},\ V^{\pi}(s) \geq V^{\pi'}(s)$
- Il existe une politique optimale, noté π^*, telle que $\forall \pi, \forall s \in \mathcal{S}, V^{\pi^*}(s) \geq V^{\pi}(s)$
- $V^*(s) = max_{\pi} V^{\pi}(s) = max_a \ \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V^*(s')\right]$
- $Q^*(s,a) = max_{\pi} Q^{\pi}(s,a) = \mathbb{E}(r(s,a,s_{t+1}) + \gamma V^*(s_{t+1}))$
  $= \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma max_{a'} \ Q^*(s',a')\right]$
- Si Q^* est connue, la politique optimale est gloutonne : $\pi^*(s) = argmax_a Q^*(s,a)$

Estimation de V^π

Opérateur de Bellman

- Nécessite la connaissance complète du MDP : transitions et récompenses.
- La résolution se fait en évaluant itérativement V_t par programmation dynamique en utilisant l'opérateur de Bellman T^π

$$V_{t+1}(s) = (T^\pi V_t)(s) = \sum_{s'} p(s'|s,\pi(s))\left[r(s,\pi(s),s') + \gamma V_t(s')\right]$$

- Opérateur contractant $\rightarrow$ existence d'un point fixe et convergence $V^\pi = lim_{n\rightarrow\infty}(T^\pi)^n(V_0)$.
- Permet d'estimer V^π par itération successive $\Rightarrow$ d'estimer une politique

Algorithme Policy et Value iteration

Policy iteration

A partir de π_0 initialisée aléatoirement, on alterne évaluation de V^π et amélioration de la politique π

- V^{π_t} est évaluer par l'opérateur de Bellman
- La politique π_{t+1} est déterminée par (politique greedy)

$$\pi_{t+1}(s) = argmax_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V^{\pi_t}(s')\right]$$

Value Iteration

A partir d'un V_0 aléatoire, sans passer par une politique

- On boucle : $V_{t+1}(s) = max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V_t(s')\right]$
- On en dérive la politique optimale :
  $\forall s\ \pi'(s) \leftarrow argmax_a \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')]$ (politique greedy)

Programmation dynamique - Résumé

Résolution exacte

- Nécessite la connaissance exacte du MDP (transitions, fonction de récompense)
- Opérateur de Bellman : $(T^\pi V)(s) = \sum_{s'} p(s'|s,\pi(s))\,[r(s,\pi(s),s') + \gamma V(s')]$
- Value iteration : $V^{t+1}(s) = max\ \sum_{s'} p(s'|s,a)\,[r(s,a,s') + \gamma V^t(s')]$
- Policy iteration : $V_\pi^{t+1} = T^\pi V_\pi^t$,
  $\pi^{t+1}(s) =$ greedy$(\pi^t)(s) = argmax_a \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V_\pi^t(s')]$

Schéma général

Répéter :

- Evaluation de la politique : estimation de V_π par itération de T^π.
- Amélioration (improvement) de la politique par décision greedy

Choisir l'action qui optimise V :

$\Rightarrow$ garantie l'amélioration de la politique.

$\Rightarrow$ garantie de convergence vers la politique optimale.

Plan

Renforcement Learning : Résumé

- Lorsque le MDP est connu : DP et planification
- Lorsque le MDP n'est pas connu : Reinforcement Learning
  - Comment estimer une politique, calculer V_π ?
  - Comment dériver des meilleures politiques, optimiser la valeur de V_π (contrôle) ?

- Différents contextes :
  - interaction directe ou non avec l'environnement
  - possibilité d'échantillonner l'espace d'actions
  - disponibilité d'un simulateur, ...

- Une constante : exploration toujours nécessaire à la convergence vers une politique optimale.
- Souvent on préfère optimiser $Q(s, a)$ plutôt que $V(s)$ (car le MDP n'est pas connu)

Algorithme de Monte-Carlo

Principe :

- estimation de $Q_\pi(s,a)$ par moyenne empirique de la récompense sur un grand nombre d'échantillons
- Estimateur non biaisé si les échantillons sont indépendants.
- Model free : aucun besoin du modèle ou d'approximation.
- Apprend à partir d'épisodes joués selon la politique

$\Rightarrow$ limite : les épisodes doivent se terminer.

Moyenne incrémentale

Pour $x_1, \ldots, x_k$ échantillons :

- $\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$
- Dans le cas non stationnaire, la moyenne d'un flux peut être approximée par : $\mu_k = \mu_{k-1} + \alpha(x_k - \mu_{k-1}) = (1-\alpha)\mu_{k-1} + \alpha x_k$

Algorithme de Monte-Carlo

Evaluation de politique : estimation de $Q_\pi(s,a)$

On procède par itération en partant de la fin :

- Répéter
  1. Générer un épisode $(s_1, a_1, r_1) \dots (s_T, a_T, r_T)$ avec la politique π
  2. Fixer $R_T = r_T$
  3. Pour $t = T - 1$ à $t = 1$:
     - $R_t = \gamma R_{t+1} + r_t$
     - $Q(s_t, a) = Q(s_t, a) + \alpha(R_t - Q(s_t, a))$

Le même algorithme peut être utilisé pour $V_\pi(s)$

Différentes variantes

- First visit : un état n'est mis à jour qu'une seule fois par scénario, la première fois qu'il est rencontré.
- Every visit : tel que ci-dessus.
- Nécessite dans tous les cas la fin du scénario pour propager la récompense.

Temporal-Difference Learning - SARSA

Principe : combiner Monte-Carlo et DP

- Apprentissage à chaque étape plutôt qu'à la fin d'un scénario
- Mise-à-jour "à la" DP
- Estimation sur grands nombres d'échantillons "à la" Monte-Carlo
- Peut apprendre avant la fin du scénario, avec des séquences incomplètes ou sans fin.
- Apprentissage "on-policy"

Algorithme SARSA

- Pour chaque épisode $\{(s_1, a_1, r_1) \dots (s_T, a_T, r_T)\}$:
- Pour $t = 1$ à $T - 1$:
  - $Q(s_t, a) = Q(s_t, a) + \alpha[r_t + \gamma Q(s_{t+1}, a) - Q(s_t, a)]$

Le même algorithme peut estimer $V_\pi(s)$.

Généralisation à n pas de lookahead et moyennage des récompenses : $TD(\lambda)$

Q-Learning : TD en off-policy

Principe : similaire à SARSA

Mais off-policy : on apprend la Q-valeur optimale (sous condition d'exploration suffisante)

Algorithme Q-Learning

- Pour chaque épisode $\{(s_1, a_1, r_1) \dots (s_T, a_T, r_T)\}$:
- Pour $t = 1$ à $T - 1$:
  - $Q(s_t, a) = Q(s_t, a) + \alpha[r_t + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a)]$

Le même algorithme peut estimer $V_\pi(s)$.

Optimisation de politique

Toujours explorer

- On est capable d'évaluer une politique π
- Mais comment l'améliorer ?

$\Rightarrow$ sans exploration, pas d'amélioration possible !

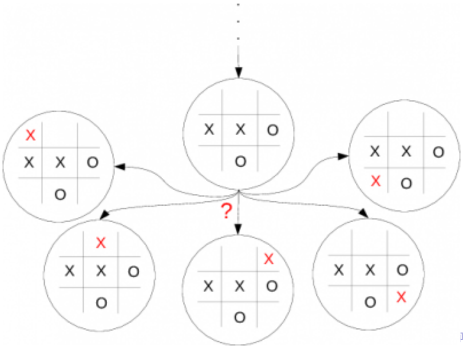
Policy iteration généralisée

- rendre non déterministe notre politique π greedy estimée (ϵ-greedy, UCB, softmax,...).
- Trois étapes à répéter :
  1. Jouer un scénario selon la politique π avec une dose d'exploration
  2. Evaluer la politique : mettre à jour V et Q
  3. Améliorer la politique : mettre à jour de manière greedy π.

Dilemme Exploration/Exploitation

A chaque état :

- Décision de l'action sur l'*estimation* de la fonction de valeurs
- A-t-on confiance ou non en cette estimation ?
  - oui $\rightarrow$ exploitation
  - non $\rightarrow$ exploration des autres actions possibles
- L'exploration peut-être dangereuse
- A-t-on assez explorer ? (récompenses relatives ...)



Dilemme Exploration/Exploitation

A chaque état :

- Décision de l'action sur l'*estimation* de la fonction de valeurs
- A-t-on confiance ou non en cette estimation ?
  - oui $\rightarrow$ exploitation
  - non $\rightarrow$ exploration des autres actions possibles
- L'exploration peut-être dangereuse
- A-t-on assez explorer ? (récompenses relatives . . .)

Algorithmes

- `greedy` (glouton),
- ϵ-`greedy` : glouton avec une probabilité de $1 - \epsilon$ au hasard sinon
- La famille UCB, à un temps T : $argmax_a \ \mu_a + \sqrt{\frac{2Klog(T)}{T_a}}$, et T_a nombre de fois où a a été choisie ($T = \sum_a T_a$).

Algorithme de contrôle

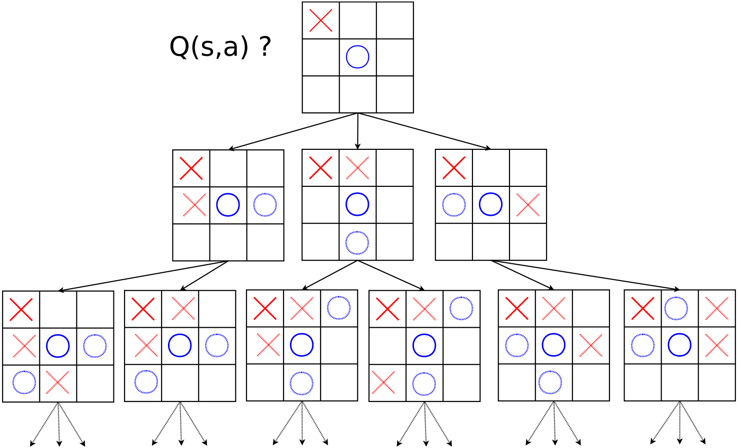
Répéter pour chaque épisode

1. Jouer un scénario selon la politique π avec une dose d'exploration
2. Evaluer la politique : mettre à jour V et Q :
   - Monte-carlo : $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_t - Q(s_t, a_t))$
   - Sarsa : $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
   - Q-learning : $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \; max_{a'} \; Q(s', a') - Q(s_t, a_t))$
3. Améliorer la politique : mettre à jour de manière greedy π.

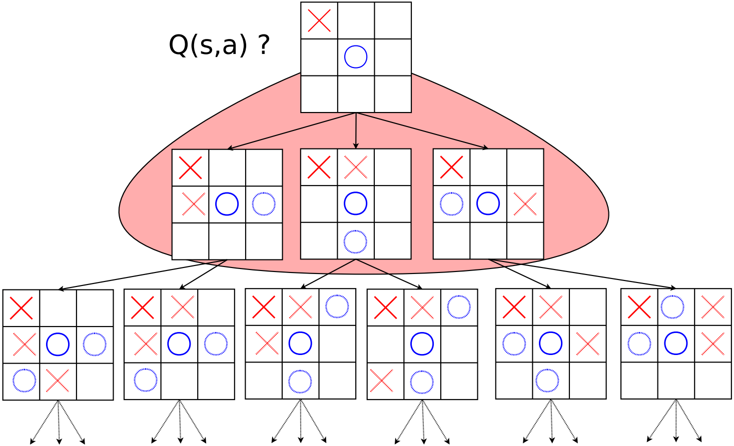
Résumé

Deux questions fondamentales

- Comment évaluer une politique : estimation de $V(s),\ Q(s,a)$
- Comment améliorer une politique : échantillonner l'espace (Monte-Carlo) ou politique dérivée de greedy

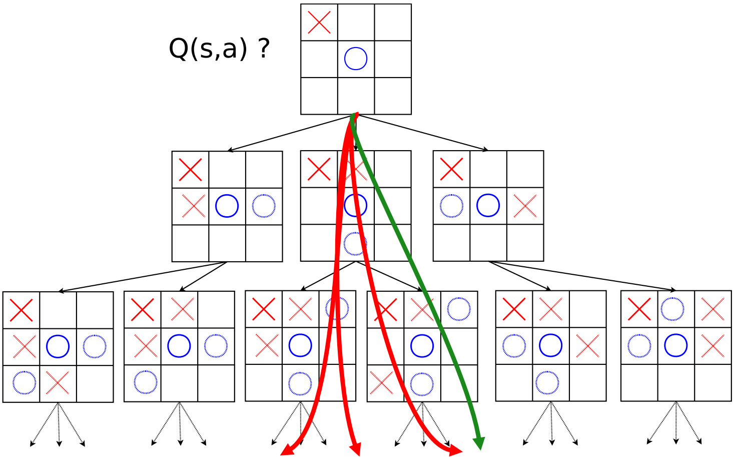


Résumé - DP



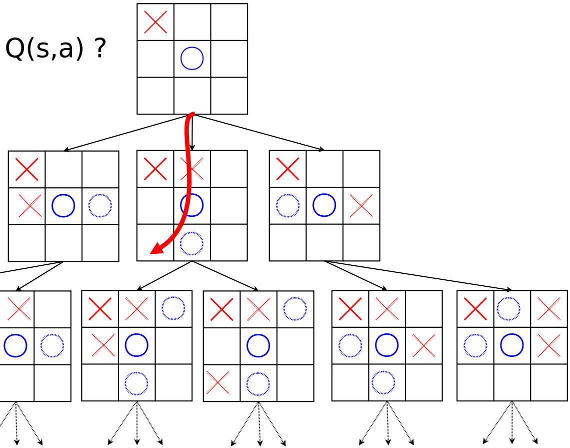
Calcul itéré sur les prochains états : $Q(s_t, a) = \mathbb{E}_\pi[r_t + \gamma V(s_{t+1}))]$

Résumé - MC



Echantillonnage dans l'espace des états et estimation par moyenne du retour du scénario complet : $Q(s_t, a) = Q(s_t, a) + \alpha(R_t - Q(s_t, a))$

Résumé - TD



Estimation à partir de la connaissance actuelle :

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

Passage à l'échelle

Limites : Complexité spatiale en $|\mathcal{S}| \times |\mathcal{A}|$

- Espace d'états devient vite gigantesque (10^{20} pour le backgammon, 10^{170} pour le go)
- Et états continus ? Et actions continues ? (exemple : contrôle de drone)

$\Rightarrow$ problème rapidement intractable

Solution : problème de régression

- On cherche une approximation de $V,\ Q$
- On fixe une famille de fonctions paramétrées par $\mathbf{w} \in \mathbb{R}^d$
  - $\hat{v}(s, \mathbf{w}) \sim V_\pi(s)$
  - $\hat{q}(s, a, \mathbf{w}') \sim Q_\pi(s, a)$
  - $\mathbf{w}$, $\mathbf{w}'$ sont optimisés en utilisant Monte-Carlo ou TD-learning
- Permet de généraliser sur des états non vus
- Utilisation d'algorithmes d'apprentissages on-line (exemple par exemple)
- Possibilité d'approximer l'état ou le couple (état,action).

En pratique

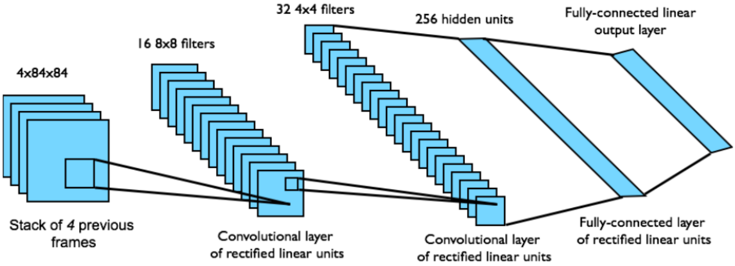
Formalisation

- Trouver $\mathbf{w}^* = argmin_{\mathbf{w}} \; \mathbb{E}_\pi[(Q_\pi(s,a) - \hat{q}(s,a,\mathbf{w}))^2]$

$\Rightarrow$ Descente de gradient stochastique :
$\mathbf{w} = \mathbf{w} + \alpha(r + \gamma\hat{q}(s',a',\mathbf{w}) - \hat{q}(s,a,\mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(s,a,\mathbf{w})$

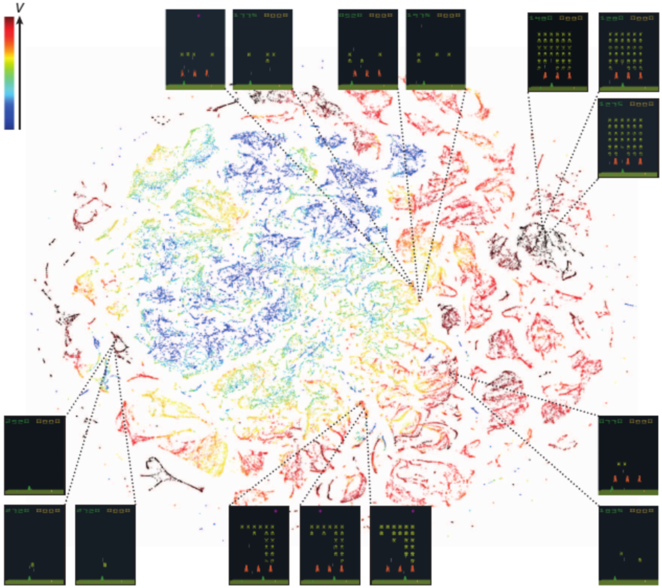
- Représentation d'un état : fonction de projection dans $\mathbb{R}^d$, $\mathbf{x}(s) = (x_1(s), \ldots, x_d(s))$
- Dans le cas linéaire : $\Delta\mathbf{w} = \alpha(r + \gamma\hat{q}(s',a',\mathbf{w}) - \hat{q}(s,a,\mathbf{w}))\mathbf{x}(s)$
- Version batch possible dans le cas d'apprentissage sur des scénarios existants

Deep RL en quelques mots slides de M. Herrmann, (U. Edinburgh) et Google DeepMind)



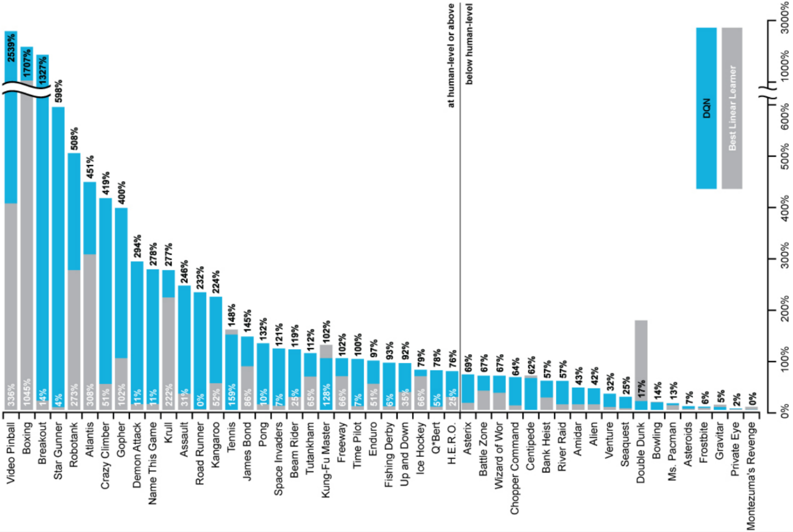
Network architecture and hyperparameters fixed across all games *[Mnih et al.]*

Deep RL en quelques mots slides de M. Herrmann, (U. Edinburgh) et Google DeepMind)



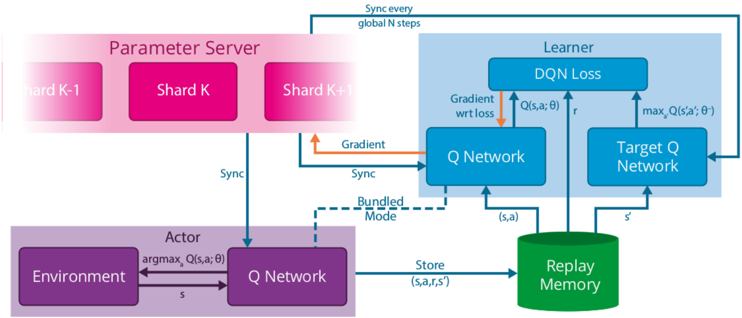
Deep RL en quelques mots

slides de M. Herrmann, (U. Edinburgh) et Google DeepMind)



Deep RL en quelques mots slides de M. Herrmann, (U. Edinburgh) et Google DeepMind)

Gorila (Google Reinforcement Learning Architecture)



- Parallel acting: generate new interactions
- Distributed replay memory: save interactions
- Parallel learning: compute gradients from replayed interactions
- Distributed neural network: update network from gradients