Apprentissage par renforcement II

Cours 10

Nicolas Baskiotis
nicolas.baskiotis@lip6.fr

Master 1 DAC

équipe MLIA, Laboratoire d'Informatique de Paris 6 (LIP6)
Université Pierre et Marie Curie (UPMC)

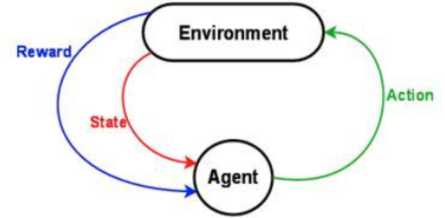
S2 (2016-2017)

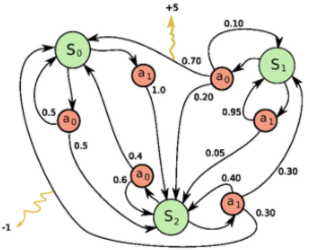
Markov Decision Process (MDP)

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
- $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$: récompense

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





Formalisation

Définitions

- Une politique $\pi : \mathcal{S} \rightarrow \mathcal{A}$ (ou dans le cas probabiliste dans $\Pi(\mathcal{A})$)
- Une fonction de valeur d'états : $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$
- Une fonction de valeur d'actions : $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
- Une séquence (un scénario) :
  $[(s_1, a_1, r_1), (s_2, a_2, r_2), \ldots, (s_n, a_n, r_n)], \quad (s_i, a_i, r_i) \in \mathcal{S} \times \mathcal{A} \times \mathbb{R}$

Fonctions valeur

- Reflète la récompense à moyen terme $\rightarrow$ aggrégation des récompenses
- Sur une séquence : $R_{t_0} = \sum_{t=t_0}^{t_T} \gamma^{t-t_0} r_t$
- $V^\pi(s) = \mathbb{E}_\pi(R_{t_0} | s_{t_0} = s) = \mathbb{E}_\pi \left[\sum_{t=t_0}^{t_T} \gamma^{t-t_0} r(s_t, \pi(s_t)) \middle| s_{t_0} = s \right]$
- $Q^\pi(s, a) = \mathbb{E}_\pi(R_{t_0} | s_{t_0} = s, a_{t_0} = a)$

Propriétés fondamentales

Fonctions optimales

- $V_*(s) = max_\pi \; V_\pi(s)$
- $Q_*(s,a) = max_\pi \; Q_\pi(s,a)$
- π_* : politique optimale telle que $\pi_* \geq \pi, \forall \pi$
  i.e $\forall s, \forall \pi V_(\pi^*)(s) = V^*(s) \geq V_\pi(s)$

Conséquences

- Pour tout MDP, il existe au moins une politique optimale.
- Si Q_* est connue, une politique greedy est optimale :
  $\pi_*(s) = argmax_a \; Q_*(s,a)$.

Equation de Bellman

- $V_*(s) = max_a \; Q_*(s,a) = max_a \; r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_*(s')$
- $Q_*(s,a) = r(s,a) + \sum_{s'} p(s'|s,a) V_*(s') = r(s,a) + \sum_{s'} p(s'|s,a) max_{a'} q_*(s',a')$

Programmation dynamique - Résumé

Résolution exacte

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

Schéma général

Répéter :

- Evaluation de la politique : estimation de V_π par itération de T^π.
- Amélioration (improvement) de la politique par décision greedy
  $\pi^{t+1} = \text{greedy}(\pi^t)$

Choisir l'action qui optimise V :

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

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

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.

Algorithme de Monte-Carlo

Principe :

- estimation de $V_\pi(s)$ 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 : $\mu_k = \mu_{k-1} + \alpha(x_k - \mu_{k-1})$

Algorithme de Monte-Carlo

Evaluation de politique : estimation de $V_\pi(s)$

- Répéter
  1. Générer un épisode $(s_1, a_1, r_1) \dots (s_T, a_T, r_T)$ avec la politique π
  2. Fixer $R = 0$
  3. Pour $t = T - 1$ à $t = 1$:
     - $R = \gamma R + r_t$
     - $V(s_t) = V(s_t) + \alpha(R - V(s_t))$

Le même algorithme peut être utilisé pour $Q_\pi(s, a)$

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

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.

Evaluation de politique

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

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

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

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.

Algorithme de contrôle

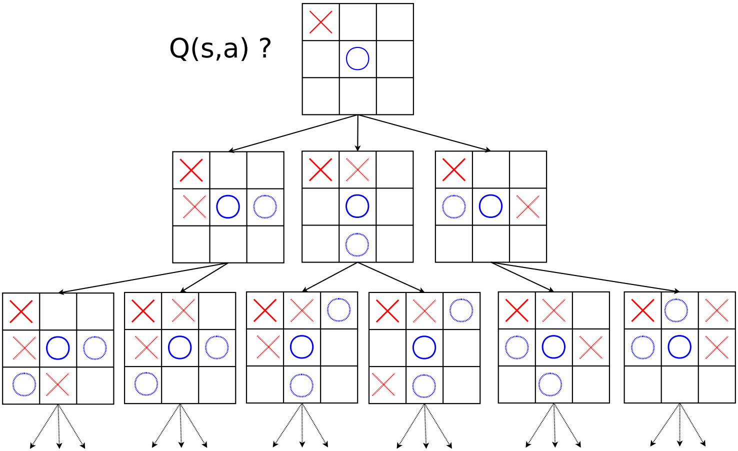
Répéter pour chaque épisode

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 :
     $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma\ max_{a'}\ \sum_{s'} p(s'|s_t, a_t)Q(s', a') - Q(s_t, a_t))$
3. Améliorer la politique : mettre à jour de manière greedy π.

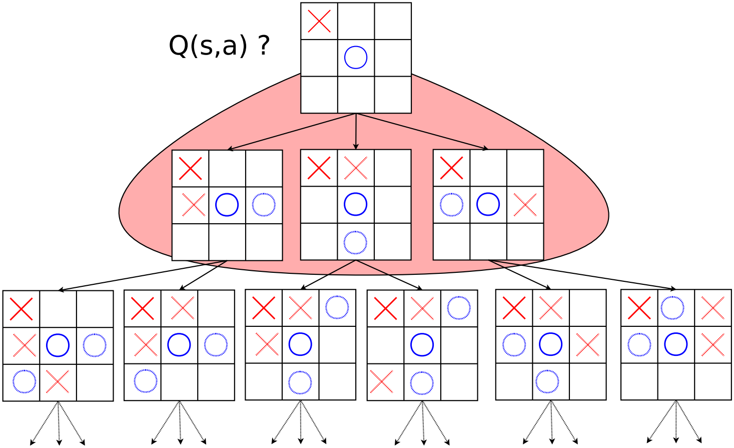
Résumé

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

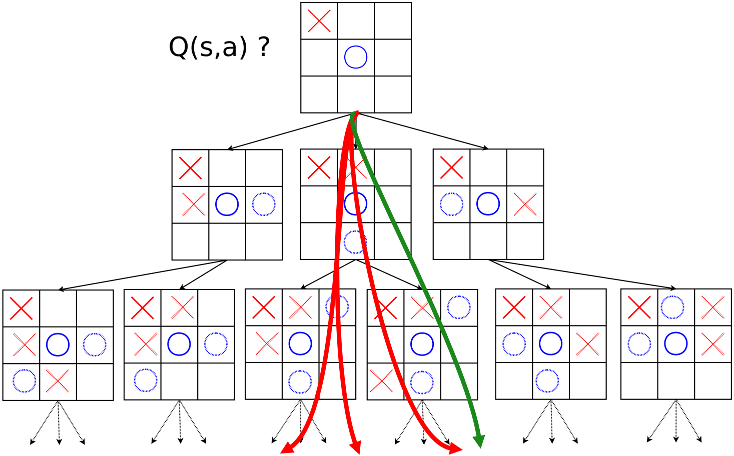


Résumé - DP



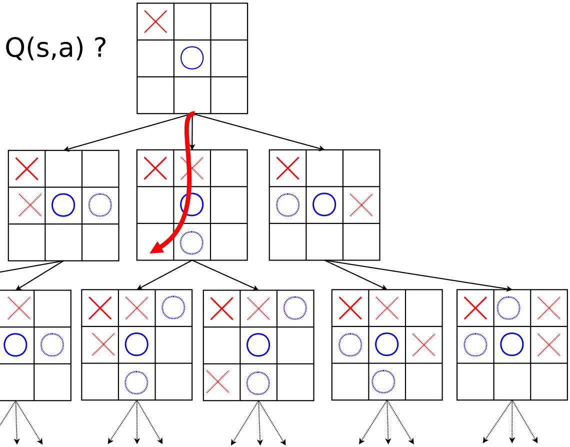
Calcul itéré sur les prochains états : $V(s_t) = \mathbb{E}_\pi[r_{t+1} + \gamma V(s_{t+1})]$

Résumé - MC



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

Résumé - TD



Estimation à partir de la connaissance actuelle :
$V(s_t) = V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$

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

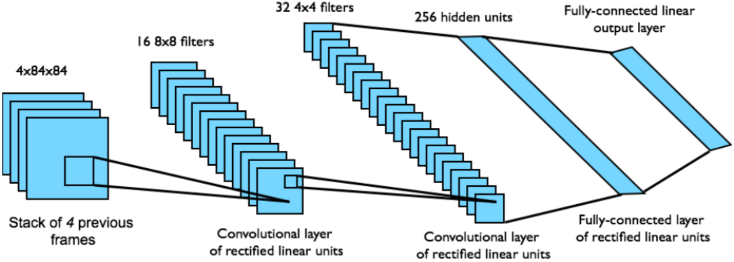
Formalisation

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

$\Rightarrow$ Descente de gradient stochastique : $\Delta\mathbf{w} = \alpha(V_\pi(s) - \hat{v}(s, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(s, \mathbf{w})$

- Représentation d'un état : fonction de projection dans $\mathbb{R}^d$, $\mathbf{x}(s) = (x_1(s), \ldots, x_d(s))$
- Famille de fonctions : linéaire, RBF, ...
- Dans le cas linéaire : $\Delta\mathbf{w} = \alpha(V_\pi(s) - \hat{v}(s, \mathbf{w}))\mathbf{x}(s)$
- Version batch possible dans le cas d'apprentissage sur des scénarios existants

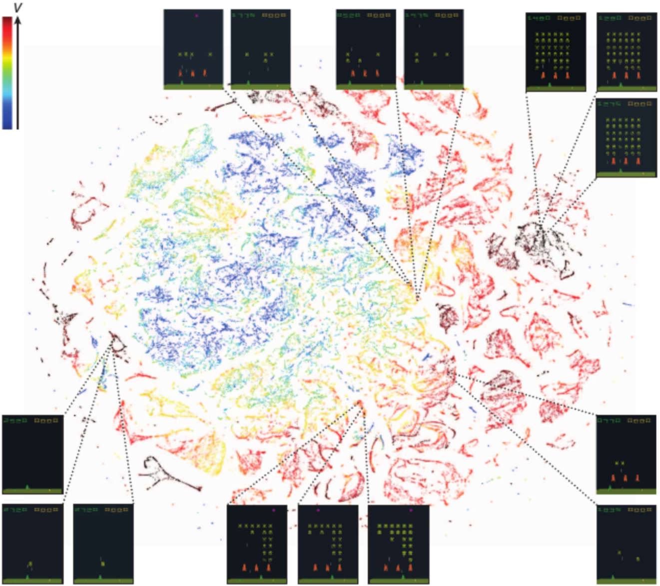
Deep RL en quelques mots slides de M. Herrmann, (U. Edinburgh) et Google DeepMind)



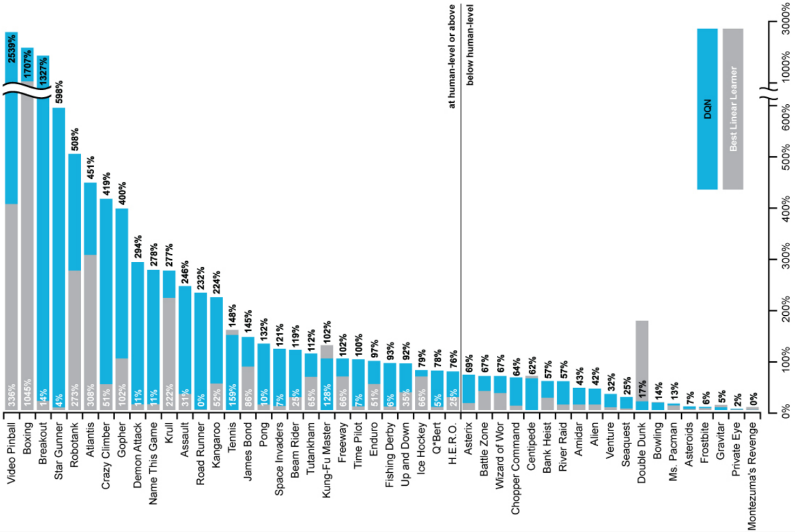
Network architecture and hyperparameters fixed across all games *[Mnih et al.]*

Deep RL en quelques mots

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



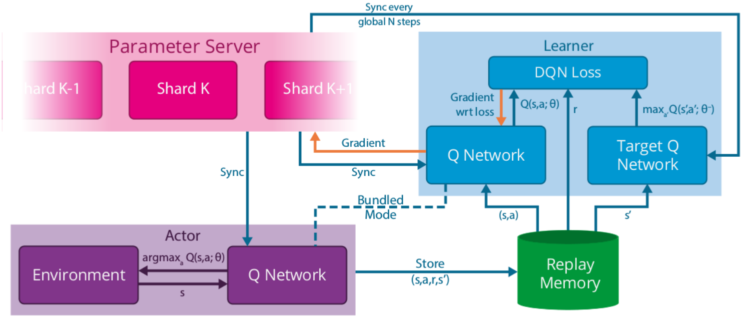
Deep RL en quelques mots slides de M. Herrmann, (U. Edinburgh) et Google DeepMind)



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