

Apprentissage par renforcement

Cours 8

Nicolas Baskiotis

`nicolas.baskiotis@lip6.fr`

Master 1 DAC

équipe MLIA, Laboratoire d'Informatique de Paris 6 (LIP6)
Sorbonne Université

S2 (2018-2019)

Plan

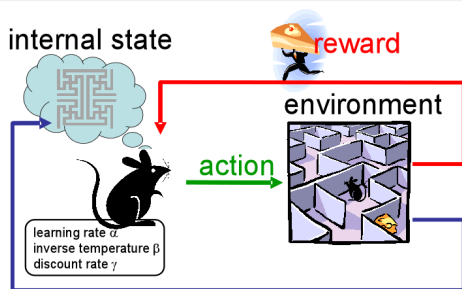
- 1 Introduction
- 2 Formalisation et outils
- 3 Reinforcement Learning

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

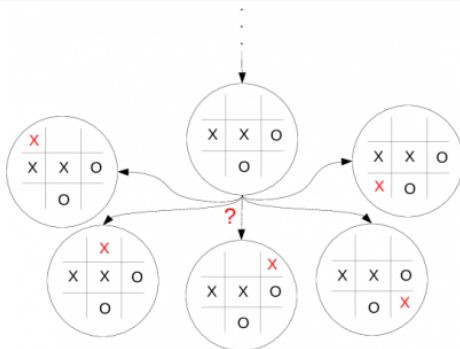


Objectif : adaptation du système à son environnement

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 ?



Objectifs

- Représenter le système comme un ensemble d'états, d'actions et de récompenses
- Apprendre à évaluer un état/une action en fonction des récompenses
- Trouver la meilleure séquence d'actions
- Explorer quand peu d'informations disponibles
- Exploiter pour rester dans des scénarios *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 scénarios déjà joués.

Différence par rapport à l'apprentissage supervisé : les exemples ne sont plus i.i.d. !!

Plan

1 Introduction

2 Formalisation et outils

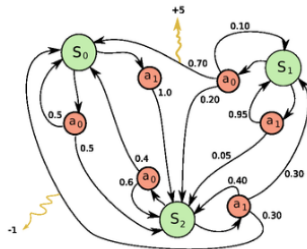
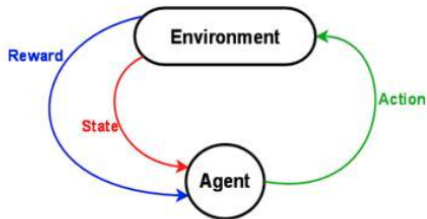
3 Reinforcement Learning

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) :
 $[(s_1, a_1, r_1), (s_2, a_2, r_2), \dots, (s_n, a_n, r_n)], \quad (s_i, a_i, r_i) \in \mathcal{S} \times \mathcal{A} \times \mathbb{R}$
- Sur une séquence de t_0 à T , la récompense globale est :
 $R_{t_0} = \sum_{t=t_0}^{T-t_0} \gamma^t r_{t_0+t}$, avec $0 < \gamma \leq 1$

Fonctions valeur

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

Equation de Bellman

Quelques relations

- π et MDP déterministe : $V^\pi(s) = r(s, \pi(s)) + \gamma V^\pi(s')$
- MDP non déterministe : $V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) V^\pi(s')$
- Politique non déterministe :
$$V^\pi(s) = \sum_a \pi(s, a) [r(s, a) + \gamma \sum_{s'} p(s'|s, a) V^\pi(s')]$$
- même relation pour $Q(s, a)$

Comparaison de politiques

- $\pi \geq \pi' \iff \forall s \in \mathcal{S} V^\pi(s) \geq V^{\pi'}(s)$
- Il existe une politique optimale, noté π^*
- $V^*(s) = \max_{\pi} V^\pi(s) = \max_a r(s, a) + \gamma \sum_{s'} p(s'|s, a) V^*(s')$
- $Q^*(s, a) = \max_{\pi} Q^\pi(s, a) = \mathbb{E}(r_{t+1} + \gamma V^*(s_{t+1} | s_t = s, a_t = a))$
 $= r(s, a) + \gamma \max_a \sum_{s'} p(s'|s, a) Q^*(s', a')$
- Si Q^* connue, la politique optimale est gloutonne :
$$\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$$

Résolution exacte : Programmation dynamique

Opérateur de Bellman

- T^π , opérateur de mise-à-jour de V sur un pas :
$$V_{t+1}(s) = (T^\pi V_t)(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) V_t(s')$$
- Opérateur contractant \rightarrow existence d'un point fixe
- Permet d'estimer V^π par itération successive \Rightarrow d'estimer une politique

Algorithme Policy Iteration

On boucle jusqu'à convergence:

- On évalue V^{π_t}
- On améliore $\pi_{t+1}(s) = \operatorname{argmax}_a [r(s, a) + \gamma \sum_{s'} p(s'|s, a) V^{\pi_t}(s')]$

Algorithme Value Iteration

- On alterne évaluation de V et amélioration de la politique
- On boucle : $V_{t+1}(s) = \max_a r(s, a) + \gamma \sum_{s'} p(s'|s, a) V_t(s')$
- On en dérive la politique optimale :
$$\forall s \pi'(s) \leftarrow \operatorname{argmax}_a \sum_{s'} p(s'|s, a) [r(s, a) + \gamma V^\pi(s')] \text{ (politique greedy)}$$

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 : $V^{t+1}(s) = \max_a r(s, a) + \gamma \sum_{s'} p(s'|s, a) V^t(s')$
- Policy iteration : $V_\pi^{t+1} = T^\pi V_\pi^t$,
 $\pi^{t+1}(s) = \text{greedy}(\pi^t)(s) = \text{argmax}_a \sum_{s'} p(s'|s, a) [r(s, a) + \gamma V_\pi^t(s')]$

Schéma général

Répéter :

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

Choisir l'action qui optimise V :

- ⇒ garantie l'amélioration de la politique.
- ⇒ garantie de convergence vers la politique optimale.

Différentes approches

- Le MDP est donné : problème d'optimisation et de planification (programmation dynamique).
 - Apprentissage par renforcement : le MDP est inconnu (seules les actions sont connues).
 - ▶ on cherche uniquement à trouver une politique π : model-free
 - ▶ on estime le MDP puis on en déduit π : model-based
 - Value iteration : itère uniquement sur la fonction de valeur d'actions, puis en déduit une politique
 - Policy iteration : mise-à-jour simultanée de la politique et la fonction valeur
 - Et pour les états continus ? actions continus ?
- ⇒ approximation par apprentissage.

Plan

- 1 Introduction
- 2 Formalisation et outils
- 3 Reinforcement Learning**

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
- ⇒ limite : les épisodes doivent se terminer.

Moyenne incrémentale

Pour x_1, \dots, 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 ?

⇒ sans exploration, pas d'amélioration possible !

Policy iteration généralisée

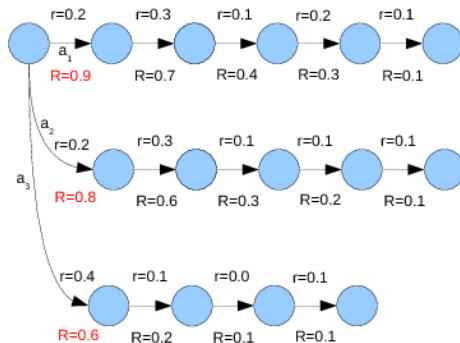
- rendre non déterministe notre politique π greedy estimée (ϵ -greedy, UCB, softmax, ...).
- Trois étapes à répéter :
 - 1 Jouer un scénario selon la politique π avec une dose d'exploration
 - 2 Evaluer la politique : mettre à jour V et Q
 - 3 Améliorer la politique : mettre à jour de manière greedy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

Problématique

Comment choisir une action ?

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

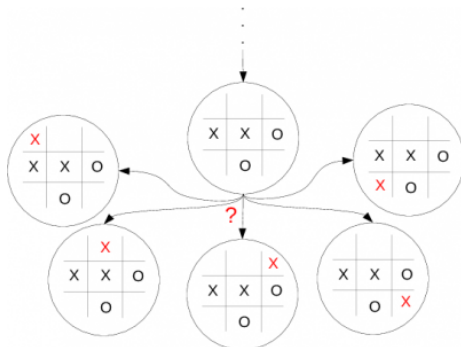
⇒ **fonction de valeur** d'états (ou d'état/action) : indication sur le long terme des récompenses attendues (\neq récompenses immédiates)



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 → exploitation
 - ▶ non → exploration des autres actions possibles
- L'exploration peut-être dangereuse
- A-t-on assez exploré ? (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 : $\operatorname{argmax}_a \mu_a + \sqrt{\frac{2\log(T)}{T_a}}$, et T_a nombre de fois où a a été choisie ($T = \sum_a T_a$).

Algorithme de contrôle

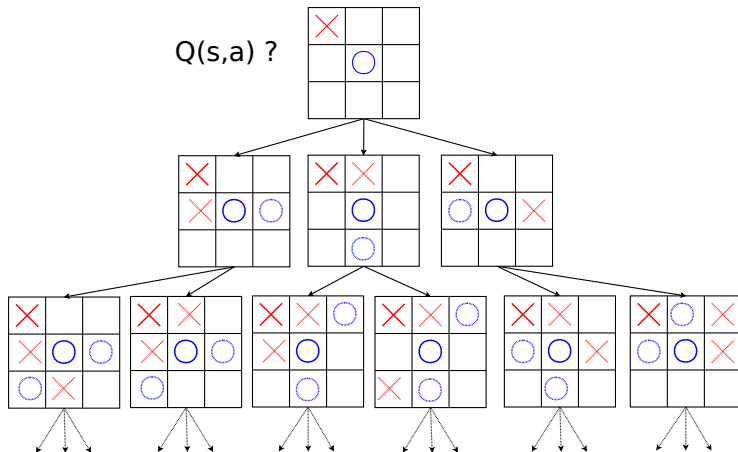
Répéter pour chaque épisode

- ➊ Jouer un scénario selon la politique π avec une dose d'exploration
- ➋ 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 :
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} \sum_{s'} p(s'|s_t, a_t) Q(s', a') - Q(s_t, a_t))$$
- ➌ Améliorer la politique : mettre à jour de manière greedy π .

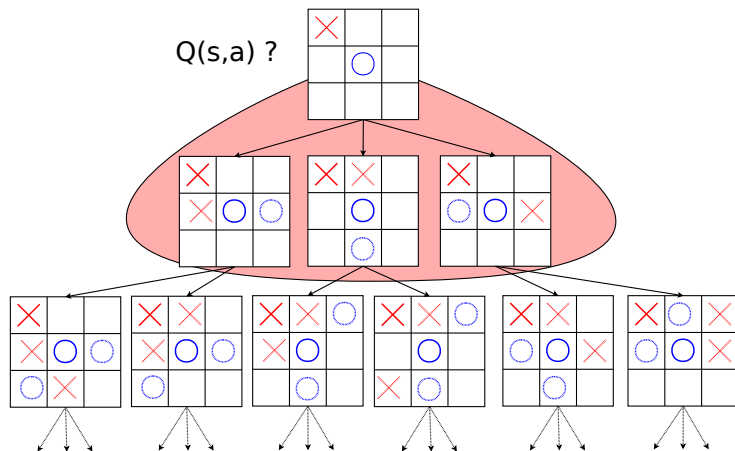
Résumé

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

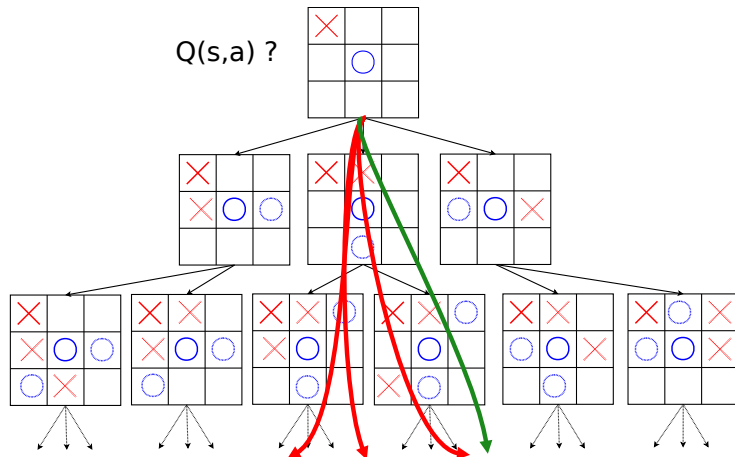


Résumé - DP



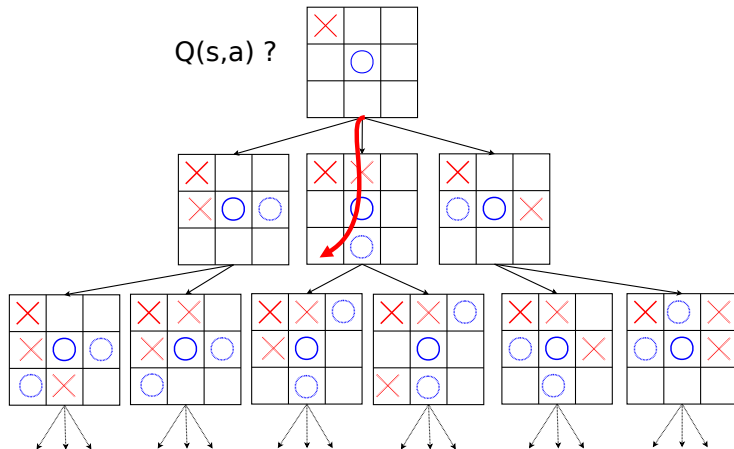
Calcul itéré sur les prochains états : $V(s_t) = \mathbb{E}_\pi[r_{t+1} + \gamma V(s_{t+1})]$

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)
- ⇒ problème rapidement intractable

Solution : problème de régression

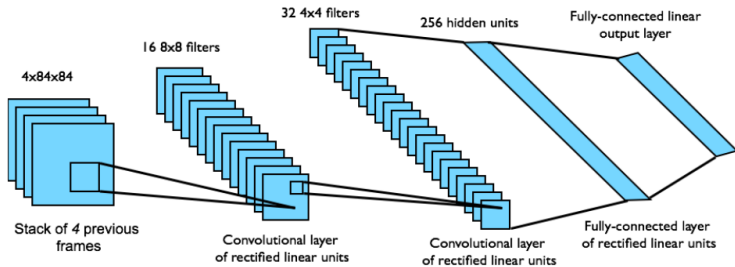
- On cherche une approximation de V, Q
- On fixe une famille de fonctions paramétrées par $\mathbf{w} \in \mathbb{R}^d$
 - ▶ $\hat{v}(s, \mathbf{w}) \sim V_\pi(s)$
 - ▶ $\hat{q}(s, a, \mathbf{w}') \sim Q_\pi(s, a)$
 - ▶ \mathbf{w}, \mathbf{w}' sont optimisés en utilisant Monte-Carlo ou TD-learning
- 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).

Formalisation

- Trouver $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \mathbb{E}_{\pi}[(V_{\pi}(s) - \hat{v}(s, \mathbf{w}))^2]$
- ⇒ 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 ,
 $\mathbf{x}(s) = (x_1(s), \dots, x_d(s))$
- 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

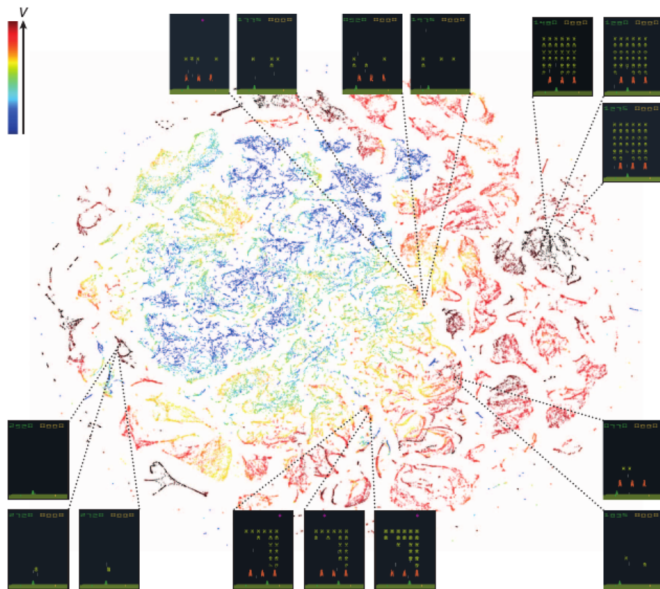
slides de M. Herrmann, (U. Edinburgh) et Google DeepMind



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

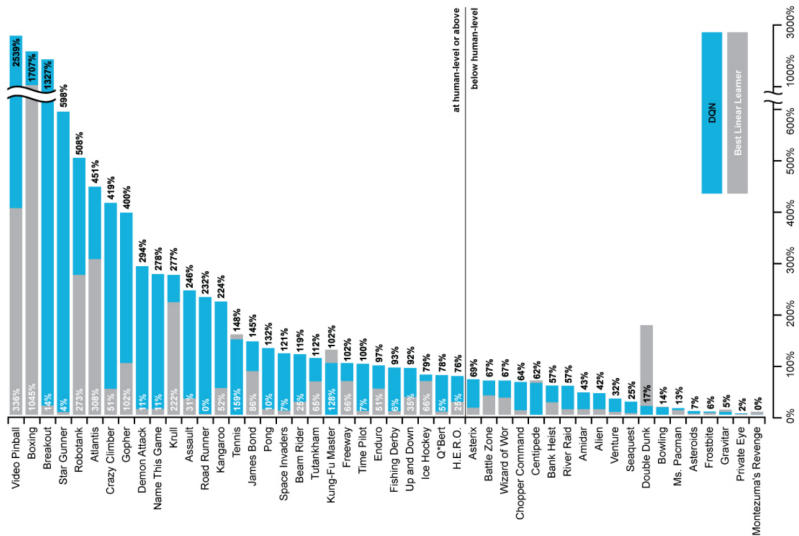
Deep RL en quelques mots

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

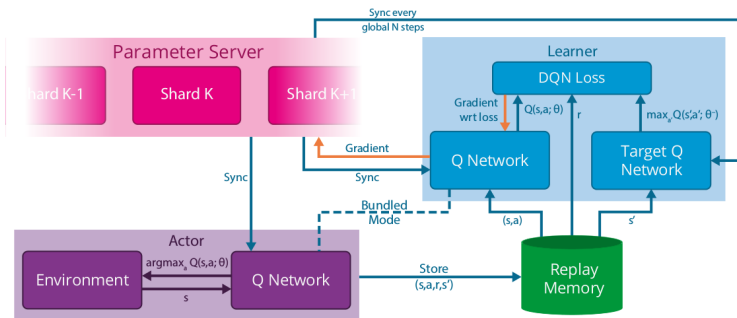


Deep RL en quelques mots

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



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