

Apprentissage par renforcement

Cours 3: Value-Based RL

Sylvain Lamprier

UE RLD - Master DAC

2020

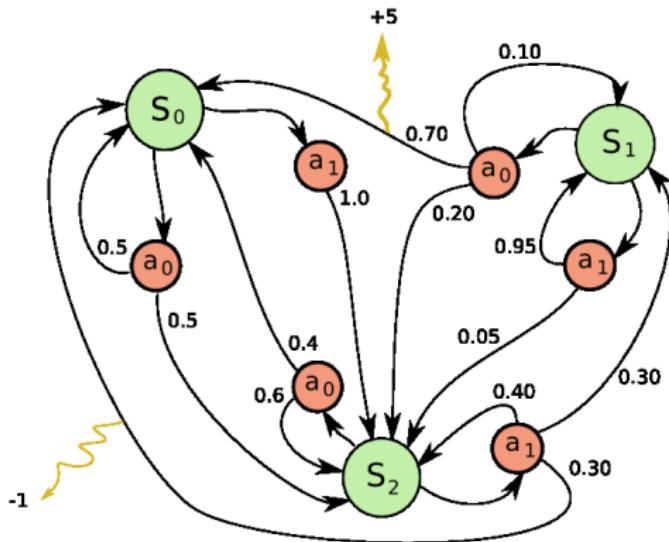
Markov Decision Process

Nous définissons un MDP comme un quadruplet $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}\}$:

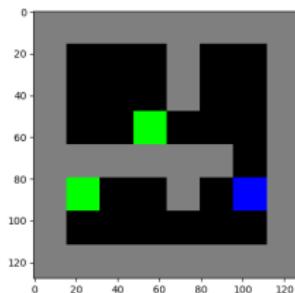
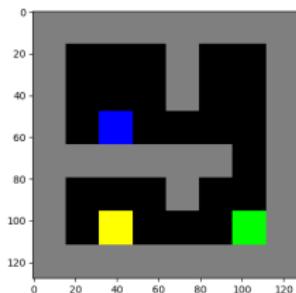
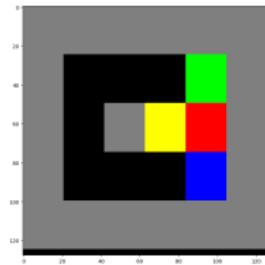
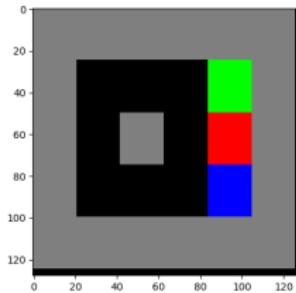
- ▶ \mathcal{S} est l'ensemble d'états (*states*)
 - ▶ \mathcal{A} est l'ensemble des actions. On note $\mathcal{A}(s)$ l'ensemble des actions dans l'état s tel que $\mathcal{A}(s) \in \mathcal{A}$
 - ▶ \mathcal{P} est la fonction de transition : $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0; 1]$. Cette fonction définit une distribution de probabilité sur les transitions $\mathcal{P}(s, a, s') = P(s' = s_{t+1} | s = s_t, a_t = a)$.
 - ▶ \mathcal{R} est la fonction de récompense (*reward*).
 $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow R$ telle que
 $R(s, a, s') = E[r_t | s_t = s, a_t = a, s_{t+1} = s']$
- ⇒ On cherche la politique π qui maximise :

$$R_t = \sum_{i=0}^{i=+\infty} \gamma^i r_{t+i}$$

MDP



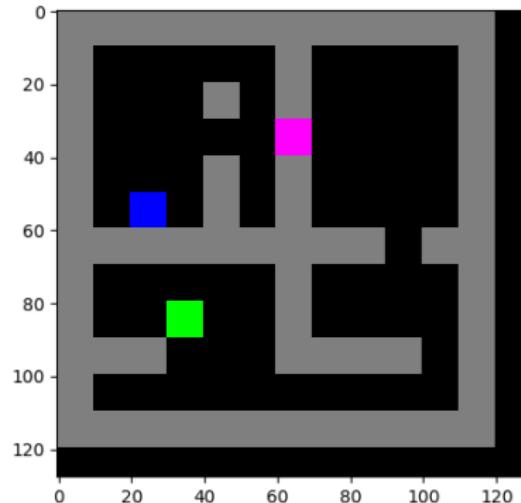
Value Iteration sur GridWorld



$R(\text{jaune})=1, R(\text{vert})=1, R(\text{rouge})=-1, R(\text{rose})=-1, R(\text{noir})=0$

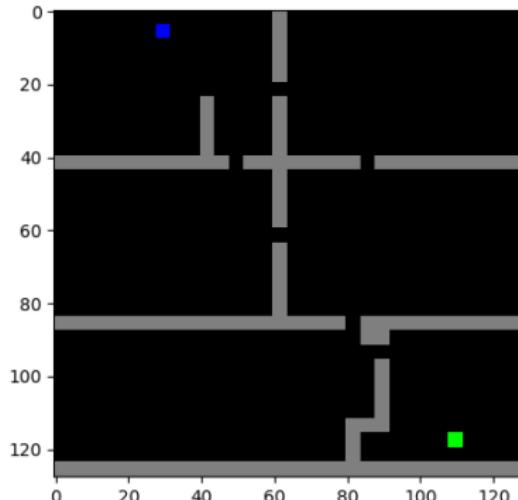
Etats terminaux = vert et rouge, discount=0.999

Value Iteration sur GridWorld



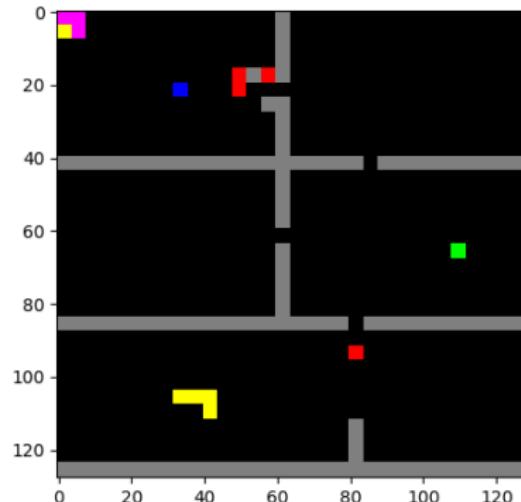
$R(\text{jaune})=1, R(\text{vert})=1, R(\text{rouge})=-1, R(\text{rose})=-1, R(\text{noir})=0$
Etats terminaux = vert et rouge, discount=0.999
⇒ Quel Problème ? Solution ?

Value Iteration sur GridWorld



$R(\text{jaune})=1$, $R(\text{vert})=1$, $R(\text{rouge})=-1$, $R(\text{rose})=-1$, $R(\text{noir})=0$
Etats terminaux = vert et rouge, discount=1
⇒ Qu'observe-t-on ?

Value Iteration sur GridWorld



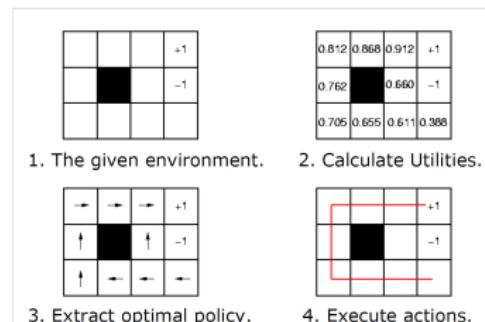
$R(\text{jaune})=1, R(\text{vert})=1, R(\text{rouge})=-1, R(\text{rose})=-1, R(\text{noir})=0$

Etats terminaux = vert et rouge, discount=0.999

⇒ MDP trop lourd à extraire ! Nombre d'états ?

Lorsque l'on connaît le MDP, deux algos de planification de trajectoires :

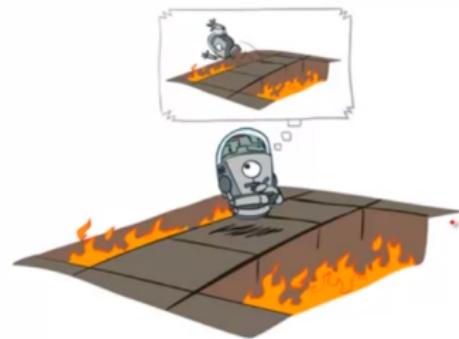
- ▶ Policy Iteration : Evaluation de la politique jusqu'à convergence puis mise à jour de la politique
- ▶ Value Iteration : Recherche itérative de la politique optimale



Comment faire lorsque l'on ne connaît ni \mathcal{P} ni \mathcal{R} ?

- ▶ Dans ce cas, l'agent doit interagir avec son environnement afin d'en comprendre le fonctionnement.
- ▶ Méthodes de Monte-Carlo (échantillonnage de trajectoires)
- ▶ Problèmes d'exploration/exploitation

Offline (MDP) Vs Online (RL)



Offline Solution



Online Learning

Value-Based RL

Known MDP: Offline Solution

Goal	Technique
Compute V^* , Q^* , π^*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

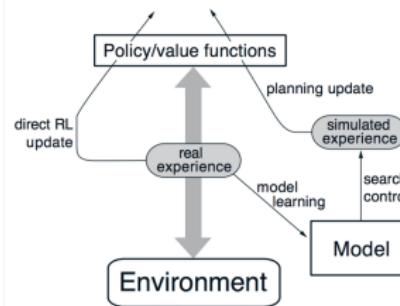
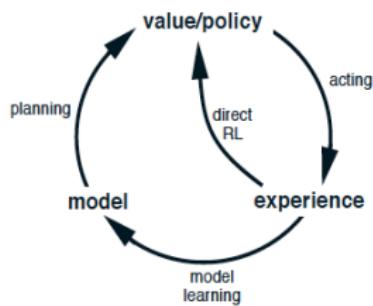
Unknown MDP: Model-Based

Goal	Technique
Compute V^* , Q^* , π^*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

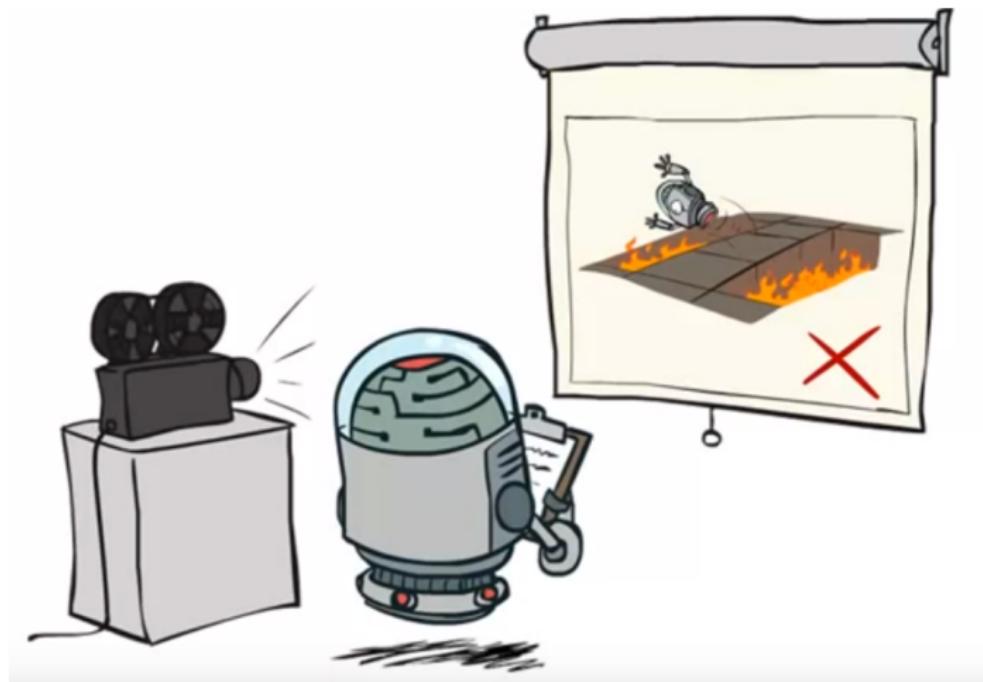
Unknown MDP: Model-Free

Goal	Technique
Compute V^* , Q^* , π^*	Q-learning
Evaluate a fixed policy π	Value Learning

Model-Free / Model-Based



Model-Based RL



Model-based RL

Idée : Apprendre le MDP avant de trouver la politique optimale dans le modèle appris.

Exemple :

Après chaque observation d'un quadruplet (s_t, a_t, s_{t+1}, r_t) :

$$\hat{R}(s_t, a_t, s_{t+1}) \leftarrow \hat{R}(s_t, a_t, s_{t+1}) + \alpha_R(r_t - \hat{R}(s_t, a_t, s_{t+1}))$$

$$\hat{P}(s_{t+1}|s_t, a_t) \leftarrow \hat{P}(s_{t+1}|s_t, a_t) + \alpha_P(1 - \hat{P}(s_{t+1}|s_t, a_t))$$

$$\forall s' \neq s_{t+1} : \hat{P}(s'|s_t, a_t) \leftarrow \hat{P}(s'|s_t, a_t) + \alpha_P(0 - \hat{P}(s'|s_t, a_t))$$

Puis estimation des valeurs selon PE :

$$V^\pi(s) \leftarrow \sum_{a \in \mathcal{A}(s)} \pi(a, s) \sum_{s'} \hat{P}(s'|s, a) \left(\hat{R}(s, a, s') + \gamma V^\pi(s') \right)$$

Et mise à jour de la politique selon PI.

Apprentissage de politique possible aussi avec VI basée sur le MDP appris :

$$V^* \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} \hat{P}(s'|s, a) \left(\hat{R}(s, a, s') + \gamma V(s') \right)$$

- ▶ Ok mais comment échantillonner les exemples ? Politique random ?
 - ▶ Si l'espace est grand on risque d'ignorer certaines zones
 - ▶ Erreurs de transition s'accumulent au fil du processus :



Model-based RL

- ▶ Possibilité de ré-échantillonner après planning :

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute those actions and add the resulting data $\{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_j\}$ to \mathcal{D}

- ▶ Ok mais on execute toutes les actions avant de ré-apprendre les dynamiques

- ▶ On risque de s'intéresser à des zones inutiles
- ▶ Model Predictive Control (replan tous les N steps) :

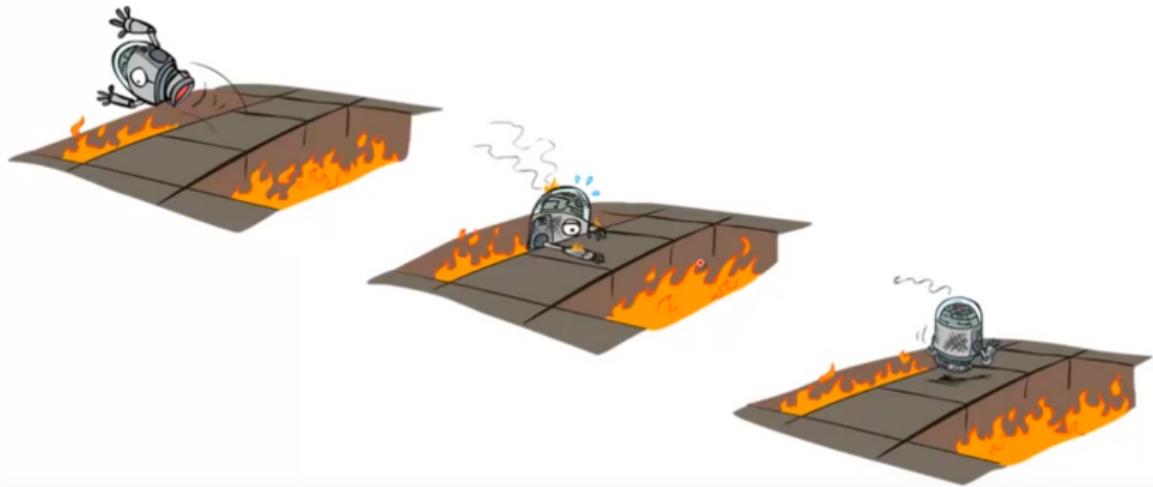
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state \mathbf{s}' (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset \mathcal{D}

every N steps



- ▶ Nombreuses méthodes globales ou locales pour modéliser les dynamiques du modèle (GPs, GMMs, NNs)
- ◀  Converge vers la politique optimale si il est possible d'apprendre parfaitement le MDP
- ◀  Apprentissage du MDP souvent plus difficile que la découverte d'une politique optimale

Model-Free RL



Évaluation directe

On peut évaluer une politique π par échantillonnages de Monte-Carlo :

Algorithme 4 Un algorithme de la famille Monte Carlo.

Nécessite: une politique π

pour chaque état s faire

pour $j \in \{0, \dots, M\}$ faire

générer une trajectoire en suivant la politique π à partir de l'état $s_0 = s$. On note $\{s_t, a_t, r_t\}_t$ la suite des triplets état, action, retour immédiat de cette trajectoire. On suppose la trajectoire de taille bornée T .

$$v(s_0, j) \leftarrow \sum_{t=0}^{t=T} \gamma^t r_t$$

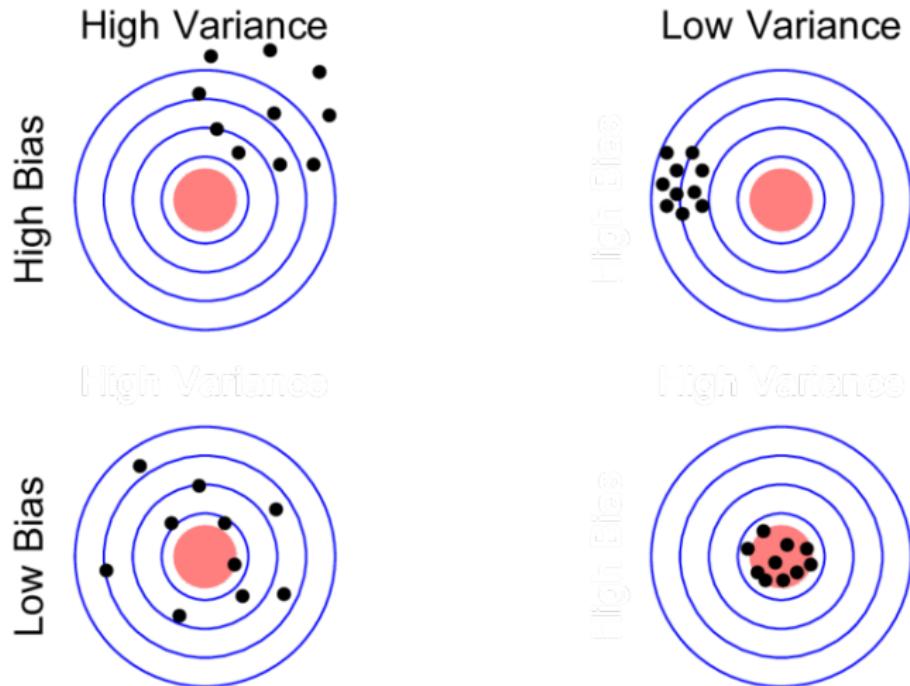
fin pour

$$V^\pi(s) \leftarrow \sum_{j=0}^{j=M} v(s, j) / M$$

fin pour

-  Pas besoin d'un modèle du monde
-  Convergence très lente car évaluation individuelle pour chaque trajectoire
-  Forte variance, pas de connexion entre les valeurs
-  Problème pour les trajectoires de taille infinie

Compromis Biais-Variance



Temporal Difference

On sait que :

$$V^\pi(s_t) = r_t + \gamma V^\pi(s_{t+1})$$

c'est à dire :

$$r_t + (\gamma V^\pi(s_{t+1}) - V^\pi(s_t)) = 0$$

$\gamma V^\pi(s_{t+1}) + r_t - V^\pi(s_t)$ est appelé la différence temporelle (*Temporal Difference*).

TD(0) Learning

Algorithme 5 L'algorithme TD(0).

Nécessite: une politique π

$V^\pi \leftarrow 0$

pour ∞ **faire**

initialiser l'état initial s_0

$t \leftarrow 0$

répéter

émettre l'action $a_t = \pi(s_t)$

observer r_t et s_{t+1}

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

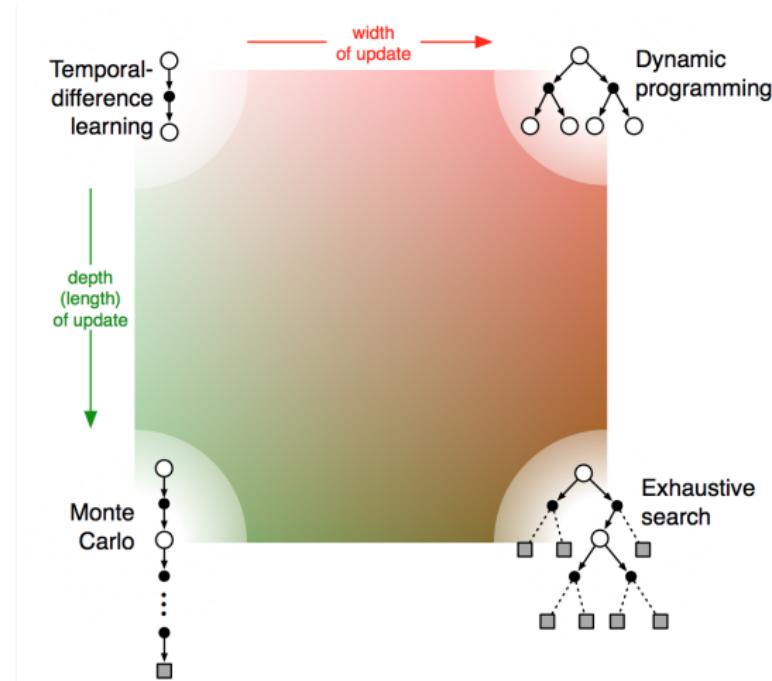
$t \leftarrow t + 1$

jusque $s_t \in \mathcal{F}$

fin pour

- 👍 Converge vers V^{π^*} si on parcourt tous les états un nombre infini de fois
- 👍 Propagation des valeurs
- 👍 Évite l'évaluation individuelle pour chaque trajectoire
- 👎 Rétro-propagation lente des valeurs (seulement d'un état à son précédent)

TD Learning



TD(λ) Learning

⇒ Idée : Considérer un horizon plus éloigné pour chaque calcul de différence temporelle

Avec $G_t^{(n)}$ la valeur cible après n étapes futures, on a ($G_t^{(1)}$ est la cible de TD(0)) :

$$n = 1 \rightarrow G_t^{(1)} = r_t + \gamma V_\pi(s_{t+1})$$

$$n = 2 \rightarrow G_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V_\pi(s_{t+2})$$

⋮

$$\forall n \rightarrow G_t^{(n)} = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n V_\pi(s_{t+n})$$

Alors, pour un n donné, notre mise à jour devient (n-step-return) :

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha(G_t^{(n)} - V_\pi(s_t))$$

TD(λ) Learning

... Mais cela paraît difficile de trouver le bon nombre d'étapes n à considérer. Et il y a peu de chances qu'un même n soit optimal pour tous les cas de figure.

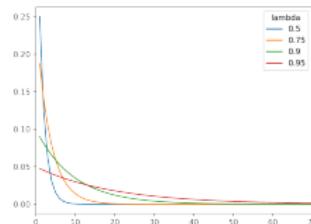
L'intuition est alors de trouver une façon de moyenner toutes les cibles jusqu'à la fin de la trajectoire, en donnant plus de poids aux mises à jour plus fiables (entre états plus proches).

Définition d'une pondération décroissant exponentiellement (λ -return) :

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

Poids somment à 1 car :

$$\sum_{n=1}^{\infty} \lambda^{n-1} = \sum_{n=0}^{\infty} \lambda^n = \frac{1}{1 - \lambda}$$



Traces d'Eligibilité

Ok mais on est obligé d'attendre la fin de la trajectoire avant de faire la moindre mise à jour ! Pas efficace, problèmes de mémoire

Heureusement on peut travailler de manière plus efficace en définissant des "traces d'éligibilité", telles que :

$$e_0(s) \leftarrow 0 \quad \forall s \in \mathcal{S}$$

$$e_t(s) \leftarrow \lambda \gamma e_{t-1}(s) + \mathbf{I}(S_t = s) \quad \forall s \in \mathcal{S}$$

Dont on peut se servir pour pondérer le passé et faire des mises à jour à chaque étape de la trajectoire :

$$\delta_t = r_t + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)$$

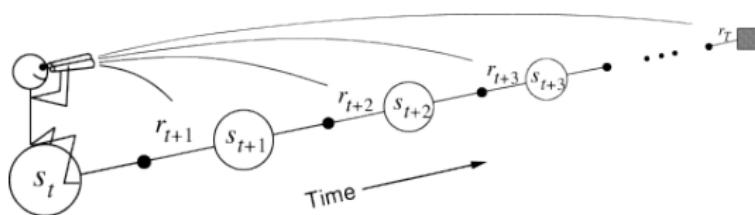
$$V_\pi(s) \leftarrow V_\pi(s) + \alpha \delta_t e_t(s) \quad \forall s \in \mathcal{S}$$

Ce qui serait équivalent à une mise à jour globale en fin de trajectoire si on travaillait sur une copie des V

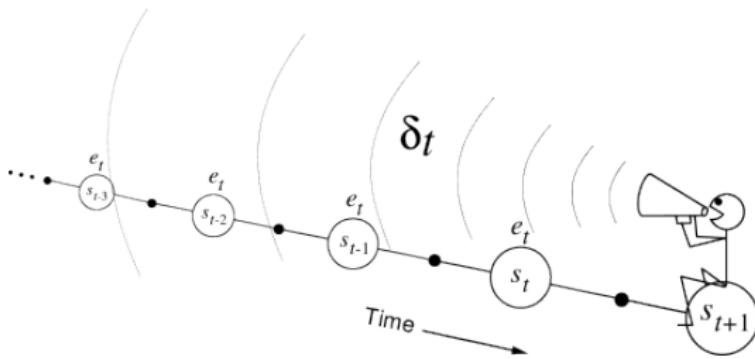
-  Mise à jour de tout le passé à chaque étape
-  Convergence plus rapide que TD(0)

TD(λ) Learning

Forward View (λ -return)



Backward View (traces d'éligibilité)



Q-Learning

Problème : comment extraire une politique à partir des valeurs ?

- ▶ On ne connaît pas \mathcal{P} ni \mathcal{R}
- ⇒ Impossible d'appliquer :

$$\pi(s) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s'|a, s)(\mathcal{R}(s, a, s') + \gamma V(s'))$$

Idée : Travailler sur les q-valeurs plutôt que sur les valeurs directement :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a' \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a') - Q(s_t, a_t))$$

Q-Learning

Algorithme 6 L'algorithme *Q-Learning*.

$$Q(s, a) \leftarrow 0, \forall (s, a) \in (\mathcal{S}, \mathcal{A})$$

pour ∞ **faire**

 initialiser l'état initial s_0

$t \leftarrow 0$

répéter

 choisir l'action à émettre a_t et l'émettre

 observer r_t et s_{t+1}

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a' \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

$t \leftarrow t + 1$

jusque $s_t \in \mathcal{F}$

fin pour

$Q(\lambda)$ -Learning

Possibilité de travailler avec des traces d'éligibilité comme sur les valeurs pour rétro-propager sur plus d'un état à chaque étape

Algorithm 10 L'algorithme $Q(\lambda)$.

```
 $Q^\pi \leftarrow 0$  (valeur arbitraire)
pour  $\infty$  faire
     $e(s, a) \leftarrow 0, \forall (s, a) \in \mathcal{S} \times \mathcal{A}$ 
     $t \leftarrow 0$ 
    initialiser l'état initial  $s_0$ 
    choisir l'action à émettre  $a_0$ 
    répéter
        émettre l'action  $a_t$ 
        observer  $r_t$  et  $s_{t+1}$ 
        choisir l'action qui sera émise dans  $s_{t+1}$ 
         $a^* \leftarrow \arg \max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a)$ 
         $\delta \leftarrow r_t + \gamma Q(s_{t+1}, a^*) - Q(s_t, a_t)$ 
         $e(s_t, a_t) \leftarrow e(s_t, a_t) + 1$ 
        pour  $(s, a) \in \mathcal{S} \times \mathcal{A}$  faire
             $Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$ 
            mise à jour de l'éligibilité de  $(s, a)$ 
        fin pour
         $t \leftarrow t + 1$ 
    jusqu'à  $s_t \in \mathcal{F}$ 
fin pour
```

Exploration

Problème : Compromis exploration/exploitation à trouver

- ▶ Stratégie greedy :
 - ▶ Exploitation pure : $a_t \leftarrow \max_{a \in \mathcal{A}(s_t)} Q(s_t, a)$
 - ▶ Sous-exploration de l'espace
 - ▶ Fort risque de converger vers une solution sous-optimale
- ▶ Stratégie ϵ -greedy : selon proba ϵ , on sélectionne l'action a_t uniformément dans $\mathcal{A}(s_t)$
 - ▶ Permet de temps en temps de sortir de la trajectoire privilégiée pour explorer les états voisins
 - ▶ Exploration long terme difficile
 - ▶ Mécanisme de decay à définir sur ϵ
- ▶ Sélection de Boltzmann :

$$p(a_t = a | s_t) = \frac{\exp(Q(s_t, a)/\tau)}{\sum_{a' \in \mathcal{A}(s_t)} \exp(Q(s_t, a')/\tau)}$$

- ▶ UCB-1 : $a_t \leftarrow \max_{a \in \mathcal{A}(s_t)} Q(s_t, a) + \lambda \sqrt{\frac{2 \log(nb(s_t))}{nb(s_t, a)}}$

Exploration

- ▶ Stratégie de Poursuite : $p(a_t = a|s_t) = \frac{\exp(p_{s_t,a})}{\sum_{a' \in \mathcal{A}(s_t)} \exp(p_{s_t,a'})}$, avec :
$$p(a_t = a|s_t) \leftarrow p_{s_t,a} + \lambda(\mathbb{I}(a = \max_{a' \in \mathcal{A}(s_t)} Q(s_t, a')) - p_{s_t,a})$$
- ▶ Optimism Face to Uncertainty :
 - ▶ Valeurs initiales de tous les états > 0
 - ▶ Permet de faire croire en le potentiel des actions inconnues en début de processus
 - ▶ Mais convergence plus lente
- ▶ Bayesian Q-Learning [DFR98]
 - ▶ Distributions Q plutôt que valeurs pour modéliser l'incertitude
 - ▶ $R_{s_t} \sim \mathcal{N}(\mu_{s_t}, \tau_{s_t})$, avec R_{s_t} les récompenses cumulées (et discountées) à partir de s_t
 - ▶ $(\mu_{s_t}, \tau_{s_t}) \sim \text{NormalGamma}(\mu_{s_t}^0, \lambda_{s_t}, a_{s_t}, b_{s_t})$
 - ▶ Sampling de Q selon leurs distributions puis mises à jour des distributions selon observations

Exploration

► Dyna-Q : Hybrid Model-based / Value-based

► A chaque étape du Q-Learning

- ▶ Choix et émission de a_t
- ▶ Observation du quadruplet (s_t, a_t, s_{t+1}, r_t)
- ▶ Mise à jour de Q selon le quadruplet observé

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a' \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a') - Q(s_t, a_t))$$

► Mise à jour du modèle de MDP

$$\hat{R}(s_t, a_t, s_{t+1}) \leftarrow \hat{R}(s_t, a_t, s_{t+1}) + \alpha_R(r_t - \hat{R}(s_t, a_t, s_{t+1}))$$

$$\hat{P}(s_{t+1}|s_t, a_t) \leftarrow \hat{P}(s_{t+1}|s_t, a_t) + \alpha_R(1 - \hat{P}(s_{t+1}|s_t, a_t))$$

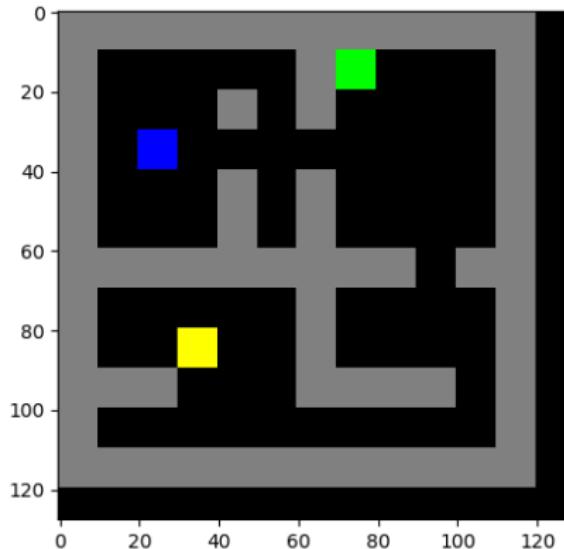
$$\forall s' \neq s_{t+1} : \hat{P}(s'|s_t, a_t) \leftarrow \hat{P}(s'|s_t, a_t) + \alpha_R(0 - \hat{P}(s'|s_t, a_t))$$

► Puis sample de k couples etat-action (éventuellement selon une stratégie) et pour chaque couple (s^i, a^i) :

$$Q(s^i, a^i) \leftarrow Q(s^i, a^i) + \alpha \left(\sum_{s' \in \mathcal{S}} \hat{P}(s'|s^i, a^i) (\hat{R}(s^i, a^i, s') + \gamma \max_{a' \in \mathcal{A}(s')} Q(s', a')) - Q(s^i, a^i) \right)$$

Exploration

Problème difficile :



- ▶ Nécessite une stratégie d'exploration élaborée pour ne pas converger vers une politique se dirigeant directement vers l'état terminal

Off-policy Vs. On-policy

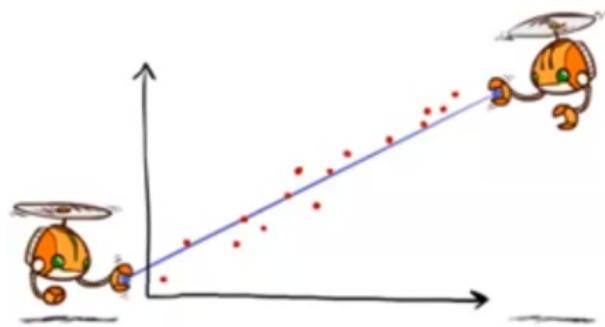
Considération de l'exploration dans le processus de mise à jour :

Algorithme 7 L'algorithme SARSA.

```
 $Q(s, a) \leftarrow 0, \forall (s, a) \in (\mathcal{S}, \mathcal{A})$ 
pour  $\infty$  faire
    initialiser l'état initial  $s_0$ 
     $t \leftarrow 0$ 
    choisir l'action à émettre  $a_t$  en fonction de la politique dérivée de  $Q$  ( $\epsilon$ -gloutonne par exemple) et
    l'émettre
    répéter
        émettre  $a_t$ 
        observer  $r_t$  et  $s_{t+1}$ 
        choisir l'action à émettre  $a_{t+1}$  en fonction de la politique dérivée de  $Q$  ( $\epsilon$ -gloutonne par exemple)
         $Q(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$ 
         $t \leftarrow t + 1$ 
    jusqu'à  $s_t \in \mathcal{F}$ 
fin pour
```

- ▶ Plus adapté à des problèmes de décision en ligne où
l'exploration est conservée tout au long du processus
- ▶ Convergence parfois plus difficile

Approximation



Approximation

- ▶ Les algorithmes classiques présentés jusqu'alors utilisent une version tabulaire du monde et s'appuient sur des estimations individuelles des valeurs pour chaque état-action
- ▶ ... Mais cela peut sembler très limité :
 - ▶ Cas des grands espaces (voire infinis si s et/ou a sont des variables continues)
 - ▶ Problème de stockage si V (ou Q) ne rentre pas en mémoire
 - ▶ Problème de l'apprentissage des valeurs (pas de généralisation des connaissances)
 - ▶ Cas des environnements à récompenses parcimonieuses / pas régulières
 - ▶ Problème d'exploration, fort risque de converger vers des solutions sous-optimales

Généralisation



Nous avons appris que cette situation est mauvaise



Pb : cela ne nous renseigne pas sur celle-ci

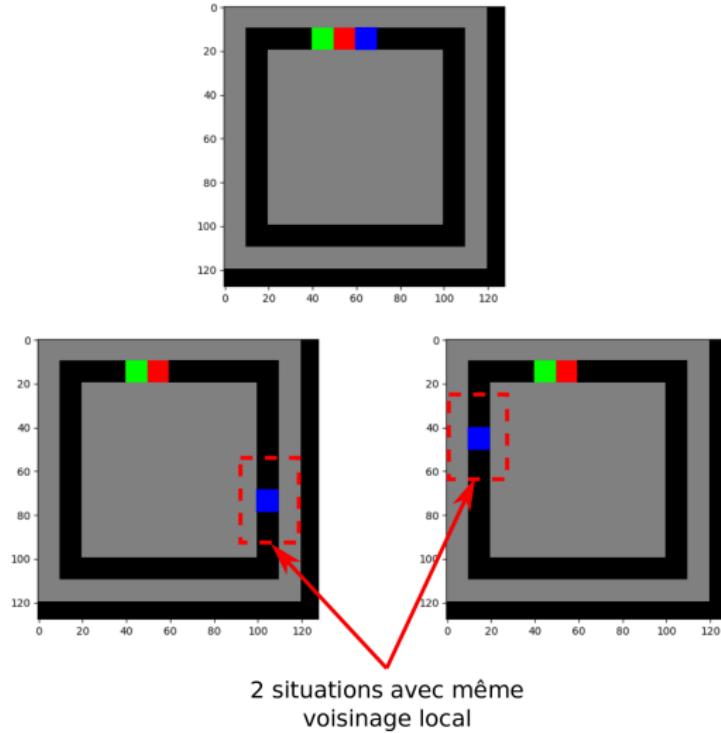


Ni même sur celle-là !

Proposition : Fusion d'états

- ▶ Si des situations se ressemblent, on considère qu'elles correspondent au même état
- ▶ Recherche de regroupements des situations similaires
 - ▶ Voisinage local identique
 - ▶ Symétrie
 - ▶ Clustering selon fonction de similarité
 - ▶ Quantization de l'espace des états (pour le cas continu)
- ▶ Le MDP dans lequel on évolue est alors simplifié
- ▶ Difficulté pour définir ces regroupements
 - ▶ Implique d'utiliser des connaissances expert du problème
 - ▶ Risque de définir des raccourcis dangereux
- ▶ Problème pour les environnements complexes (peu de regroupements raisonnables possibles)

Généralisation



Faut-il les considérer selon un même état ?

Q-Learning Approximé

Plutôt que de chercher un mapping de situations similaires à des états discrets

- ⇒ Apprentissage d'une fonction d'approximation de Q
- ▶ Soit θ un ensemble de paramètres : $\theta << |s|$
- ▶ On pose : $Q(s, a) \approx Q_{\theta, a}(\phi(s))$, avec $Q_{\theta} : \mathbb{R}^d \rightarrow \mathbb{R}^{|\mathcal{A}(s)|}$
- ▶ $\phi(s)$ est le vecteur de caractéristiques de s de taille d
- ▶ **Intérêts :**
 - ▶ On stocke θ au lieu de $Q(s, a)$
 - ▶ Apprentissage conjoint des q-valeurs optimales : selon le degré de liberté de Q_{θ} on peut généraliser entre situations similaires

Q-Learning Approximé

La fonction $\Phi(s)$ décrit, sous forme de vecteurs, les caractéristiques des différents s possibles.

- ▶ Plusieurs fonctions possibles
- ▶ Information souvent incomplète : $\Phi(s) = \Phi(s') \Rightarrow s = s'$
- ▶ Fixée manuellement



$$\Phi(s) = \begin{pmatrix} \text{nombre de trous} \\ \text{hauteur moyenne} \\ \text{nombre de briques supprimées} \\ \text{nombre de transitions vide/plein par ligne} \\ \vdots \\ \text{piece en cours} \end{pmatrix}$$

⇒ Peut aussi correspondre à une représentation simplifiée des k dernières images du jeu

Q-Learning Approximé

Différence temporelle :

$$\begin{aligned}\delta_t &= \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q^\pi(s_{t+1}, a) + r_t - Q^\pi(s_t, a_t) \\ &\approx \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q_{\theta, a}(\phi(s_{t+1})) + r_t - Q_{\theta, a_t}(\phi(s_t))\end{aligned}\quad (1)$$

On veut faire tendre la différence temporelle vers 0, on va donc utiliser une descente de gradient :

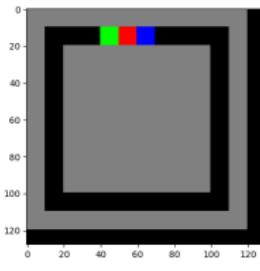
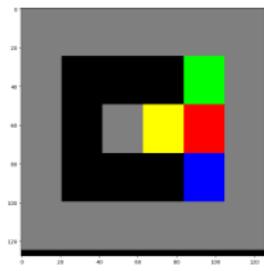
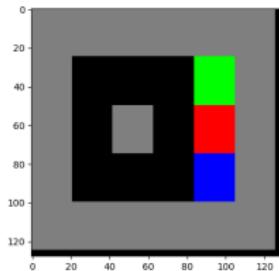
$$\theta \leftarrow \theta - \epsilon \nabla_\theta (\delta_t^2)$$

Cas Linéaire : $Q_{\theta, a}(\phi(s)) = \langle \theta_a, \phi(s) \rangle$ (régression linéaire). Dans ce cas, on a :

$$\delta_t \approx \gamma \max_{a \in \mathcal{A}(s_{t+1})} \langle \theta_a, \phi(s_{t+1}) \rangle + r_t - \langle \theta_{a_t}, \phi(s_t) \rangle$$

Q-Learning Approximé

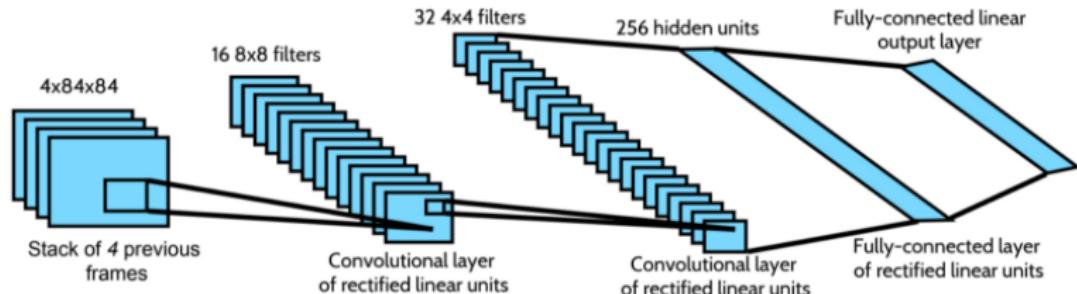
Importance de la représentation choisie



- ▶ $\Phi(s) = \begin{pmatrix} \text{point vert dans une case adjacente} \\ \text{point rouge dans une case adjacente} \\ \text{point jaune dans une case adjacente} \end{pmatrix}$?
- ▶ $\Phi(s) = \begin{pmatrix} \text{distance point vert} \\ \text{distance point rouge} \\ \text{distance point jaune} \end{pmatrix}$?
- ▶ $\Phi(s) = (\text{vector one-hot avec 1 indique position point bleu})$?
- ▶ $\Phi(s) = \begin{pmatrix} \text{vector one-hot avec 1 indique position point bleu} \\ \text{nb points jaunes, nb points rose} \end{pmatrix}$?

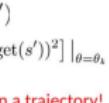
Deep Q-Network (DQN) [Mni+13]

- ▶ Choix de la représentation difficile
 - ▶ Nécessite une connaissance expert du problème
 - ▶ Dépend fortement de la fonction Q
 - ▶ Dépend de la capacité de perception de l'agent
 - ⇒ Travail avec version brute des données + apprentissage profond
- ▶ Dans le cadre des jeux video type ATARI games
 - ▶ Agent = Joueur
 - ▶ Perception = Images observées
 - ▶ Choix classique : Φ = pixels (en niveaux de gris) des 4 images parmi celles de la dernière seconde
 - ▶ Fonction Q : Réseau de convolution 2D (considérations locales dans l'image) puis MLP



Deep Q-Network (DQN) [Mni+13]

- ▶ En apprentissage supervisé, on s'appuie sur la propriété iid des données :
 - ▶ Tous les exemples suivent la même distribution ;
 - ▶ Les exemples sont indépendants les uns des autres.
- ▶ En renforcement, c'est plus compliqué :

```
Start with  $Q_0(s, a)$  for all  $s, a$ .  
Get initial state  $s$   
For  $k = 1, 2, \dots$  till convergence  
    Sample action  $a$ , get next state  $s'$   
    If  $s'$  is terminal:  
        target =  $R(s, a, s')$   
        Sample new initial state  $s'$   
    else:  
        target =  $R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$   
          
         $\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \mathbb{E}_{s' \sim P(s'|s,a)} [(Q_\theta(s,a) - \text{target}(s'))^2] |_{\theta=\theta_k}$   
          
         $s \leftarrow s'$   
        Updates are correlated within a trajectory!
```

- ▶ Corrélations entre échantillons d'une même trajectoire
- ▶ Valeurs cibles mouvantes

Deep Q-Network (DQN) [Mni+13]

► Experience Replay

- Enregistrement des transitions dans une mémoire D et apprentissage à partir de mini-batchs échantillonés dans D
- ⇒ Réduit les corrélations entre exemples d'apprentissage
- ⇒ Permet d'accélérer l'apprentissage par optimisation sur mini-batchs (parallelisation)
- ⇒ Réutilise des transitions passées pour éviter l'oubli d'informations importantes (catastrophic forgetting)

► Target Network

- Deux réseaux séparés pour l'apprenant et la cible
- Toutes les k itérations, copie des paramètres de l'apprenant θ dans ceux de la cible θ^-

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r \sim D} \left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i) \right)^2$$

- ⇒ Stabilisation de la cible
- ⇒ Réduction des corrélations entre cibles

Deep Q-Network (DQN) [Mni+13]

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ε select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

Deep Q-Network (DQN) [Mni+13]



Gains du Experience Replay et du Target Network sur divers jeux ATARI :

Replay	○	○	✗	✗
Target	○	✗	○	✗
Breakout	316.8	240.7	10.2	3.2
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

Prioritized Experience Replay [Sch+15]

Version basique de Experience Replay sélectionne les exemples uniformément dans la base D . Cependant :

- ▶ Certains exemples sont déjà à leur valeur cible
- ▶ Beaucoup de travail pour rien (gradients quasi nuls sur ces exemples)

Plutôt que d'échantillonner les exemples uniformément :

- ▶ Sélection des exemples en fonction de leur écart δ_t à leur valeur cible :

Transition i samplée selon $i \sim P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ avec $p_i = |\delta_i| + \epsilon$

$$\text{où } \delta_t = \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q^\pi(s_{t+1}, a) + r_t - Q^\pi(s_t, a_t)$$

- 👍 Permet de se concentrer plus probablement sur les exemples utiles...

Prioritized Experience Replay [Sch+15]

👎 ... Mais induit un biais d'évaluation des valeurs Q

- ▶ Car minimisation de $\frac{1}{N} \sum_i (\delta^{(i)})^2$ pour faite sur des exemples i échantillonnés selon $P(i)$ plutôt que uniformément

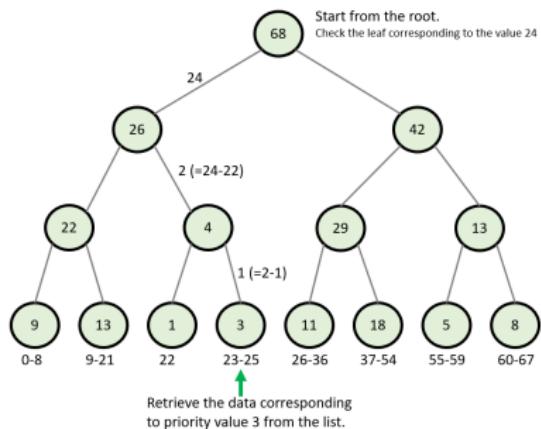
importance sampling

$$\begin{aligned} E_{x \sim p(x)}[f(x)] &= \int p(x)f(x)dx \\ &= \int \frac{q(x)}{q(x)}p(x)f(x)dx \\ &= \int q(x)\frac{p(x)}{q(x)}f(x)dx \\ &= E_{x \sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right] \end{aligned}$$

- ⇒ Possibilité de débiaiser en pondérant par $w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$ le gradient de chaque exemple i
- ▶ Auteurs de PER font croître β de 0 à 1 au cours du processus (biais devant surtout être corrigé en fin de processus)
 - ▶ Afin de réduire la variance, possibilité de diviser tous les gradients d'un batch par la constante $\max_i w_i$

Prioritized Experience Replay [Sch+15]

- ⇒ Requiert de pouvoir sampler efficacement selon $P(i)$
 - ▶ Échantillonnage par roulette classique a une complexité qui croît en N
 - ▶ Pas envisageable pour PER qui requiert souvent des buffers contenant des centaines de milliers de transitions
- thumb-up Possibilité d'utiliser des structures efficaces type SumTree :



* Double DQN [Has10]

Pb : Q-Learning tend à sur-estimer la valeur Q cible

Pour s'en rendre compte, prenons le MDP suivant avec A l'état initial et C et D deux états terminaux :

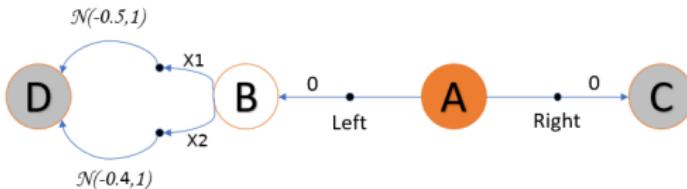


Table des rewards associés aux actions X_1 et X_2 :

Steps	X_1	μ_1	X_2	μ_2	Max(μ)
1	3	3.00	-8	-8.00	3.00
2	-2	0.50	-5	-6.50	0.50
3	4	1.67	-8	-7.00	1.67
4	-1	1.00	-10	-7.75	1.00
5	-6	-0.20	5	-5.20	-0.20
6	2	0.17	-7	-5.50	0.17
7	-6	-0.71	-3	-5.14	-0.71
8	-3	-1.00	3	-4.13	-1.00
9	8	0.00	8	-2.78	0.00
10	-6	-0.60	9	-1.60	-0.60
11	-9	-1.36	2	-1.27	-1.36
12	9	-0.50	-1	-1.25	-0.50
13	1	-0.38	2	-1.00	-0.38
14	-1	-0.43	8	-0.99	-0.38
E(X)	0.43		0.38	E(Max(μ))	0.09
Max E(X)	-0.36				

Q-Learning utilise $\text{Max}Q(s', a)$, soit $\text{Max}(\mu)$ dans la table, $> \text{Max}(E[X_1], E[X_2])$ qui nous intéresse. Ca nous amène à estimer que Left est une meilleure action que Right !

* Double DQN [Has10]

Plus généralement, soient deux variables aléatoires X_1 et X_2 , on a :

$$\mathbb{E}[\max(X_1, X_2)] \geq \max(\mathbb{E}[X_1], \mathbb{E}[X_2])$$

DQN sur-estime la valeur de Q cible (car $Q_{\theta^-}(\phi(s), a)$ bruité) :

$$\mathbb{E}[r_t + \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q_{\theta^-}(\phi(s_{t+1}), a)] \geq \mathbb{E}[r_t] + \gamma \max_{a \in \mathcal{A}(s_{t+1})} (\mathbb{E}[Q_{\theta^-}(\phi(s_{t+1}), a)])$$

Notons que : $\max_{a \in \mathcal{A}(s)} Q_{\theta}(\phi(s), a) = Q_{\theta}(\phi(s), \arg \max_{a \in \mathcal{A}(s)} Q_{\theta}(\phi(s), a))$

L'idée est de décorrélérer l'erreur d'estimation de la valeur de celle du choix de la valeur maximale :

$$\delta_t = r_t + \gamma Q_{\theta^-}(\phi(s_{t+1}), \arg \max_{a \in \mathcal{A}(s_{t+1})} Q_{\theta}(\phi(s_{t+1}), a)) - Q_{\theta}(\phi(s_t), a_t)$$

- ▶ Utilisation du réseau courant θ pour le choix de la prochaine action
- ▶ Utilisation du réseau cible θ^- pour l'estimation de sa valeur
- ⇒ Sous-estimation théorique de Q ...
- ⇒ ... Mais fonctionne bien en pratique

* Double DQN [Has10]

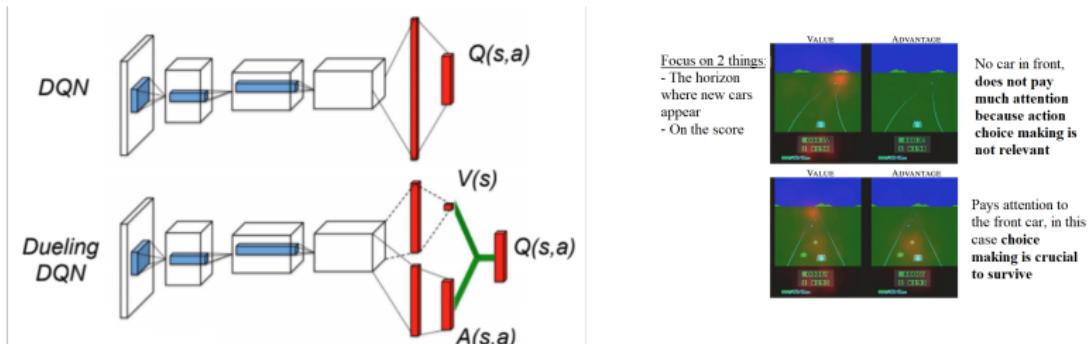
Algorithm 1 Double DQN with proportional prioritization

```
1: Input: minibatch  $k$ , step-size  $\eta$ , replay period  $K$  and size  $N$ , exponents  $\alpha$  and  $\beta$ , budget  $T$ .
2: Initialize replay memory  $\mathcal{H} = \emptyset$ ,  $\Delta = 0$ ,  $p_1 = 1$ 
3: Observe  $S_0$  and choose  $A_0 \sim \pi_\theta(S_0)$ 
4: for  $t = 1$  to  $T$  do
5:   Observe  $S_t, R_t, \gamma_t$ 
6:   Store transition  $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$  in  $\mathcal{H}$  with maximal priority  $p_t = \max_{i < t} p_i$ 
7:   if  $t \equiv 0 \pmod K$  then
8:     for  $j = 1$  to  $k$  do
9:       Sample transition  $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$ 
10:      Compute importance-sampling weight  $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$ 
11:      Compute TD-error  $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg \max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$ 
12:      Update transition priority  $p_j \leftarrow |\delta_j|$ 
13:      Accumulate weight-change  $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$ 
14:    end for
15:    Update weights  $\theta \leftarrow \theta + \eta \cdot \Delta$ , reset  $\Delta = 0$ 
16:    From time to time copy weights into target network  $\theta_{\text{target}} \leftarrow \theta$ 
17:  end if
18:  Choose action  $A_t \sim \pi_\theta(S_t)$ 
19: end for
```

*Dueling DQN [Wan+16]

Dueling DQN définit Q comme une combinaison d'une fonction de valeur V et une fonction d'avantage A pour chaque action :

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{|\mathcal{A}(s)|} \sum_{a' \in \mathcal{A}(s)} A(s, a')$$



- ▶ Permet de faire des mises à jour avec prises en compte des valeurs relatives des différentes actions
- ▶ Valeur d'espérance plus stable dans V , spécificité de chaque action dans A

*Noisy Nets for Exploration [For+17]

Plutôt que ϵ -greedy pour l'exploration :

- ▶ Ajout d'un bruit $\epsilon \sim \mathcal{N}(0, 1)$ à chaque paramètre de la couche de sortie
- ▶ Politique greedy selon les valeurs de sortie

Consider a linear layer of a neural network with p inputs and q outputs, represented by

$$y = wx + b,$$

where $x \in \mathbb{R}^p$ is the layer input, $w \in \mathbb{R}^{q \times p}$ the weight matrix, and $b \in \mathbb{R}^q$ the bias. The corresponding noisy linear layer is defined as:

$$y \stackrel{\text{def}}{=} (\mu^w + \sigma^w \odot \varepsilon^w)x + \mu^b + \sigma^b \odot \varepsilon^b,$$

where $\mu^w + \sigma^w \odot \varepsilon^w$ and $\mu^b + \sigma^b \odot \varepsilon^b$ replace w and b in Eq. (8), respectively. The parameters $\mu^w \in \mathbb{R}^{q \times p}$, $\mu^b \in \mathbb{R}^q$, $\sigma^w \in \mathbb{R}^{q \times p}$ and $\sigma^b \in \mathbb{R}^q$ are learnable whereas $\varepsilon^w \in \mathbb{R}^{q \times p}$ and $\varepsilon^b \in \mathbb{R}^q$ are noise random variables. □

□

- ◀ Exploration adaptative (selon σ)
- ◀ Exploration différente selon états
- ◀ Réduit le risque de sur-apprentissage

*Distributional DQN [BDM17]

Propagation de distributions de récompenses Z plutôt que propagation d'espérances Q :

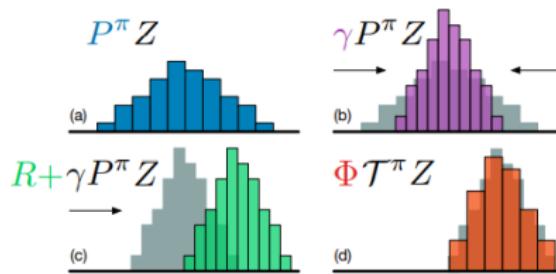


Figure 1. A distributional Bellman operator with a deterministic reward function: (a) Next state distribution under policy π , (b) Discounting shrinks the distribution towards 0, (c) The reward shifts it, and (d) Projection step

- ▶ Introduction de l'opérateur de Bellman distributionnel :

$$\mathcal{T}^\pi Z(x, a) := R(x, a) + \gamma P^\pi Z(x, a) \text{ avec } P^\pi Z(x, a) := Z(X', A')$$

- ▶ Projection $\Phi \mathcal{T}^\pi Z(x, a)$ de $\mathcal{T}^\pi Z(x, a)$ sur le support de $Z(x, a)$
- ▶ Minimisation de la divergence $D_{\text{KL}} \left(\Phi \hat{\mathcal{T}} Z_{\tilde{\theta}}(x, a) \| Z_\theta(x, a) \right)$

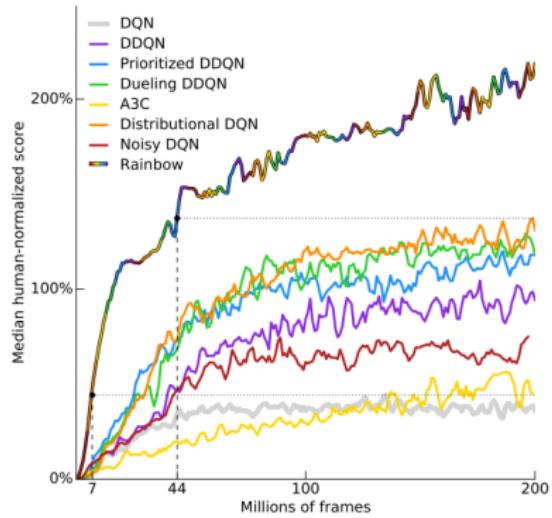
*Rainbow : On combine tout ! [Hes+17]

Combinaison dans DQN de :

- ▶ Multi-step return : $R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1}$

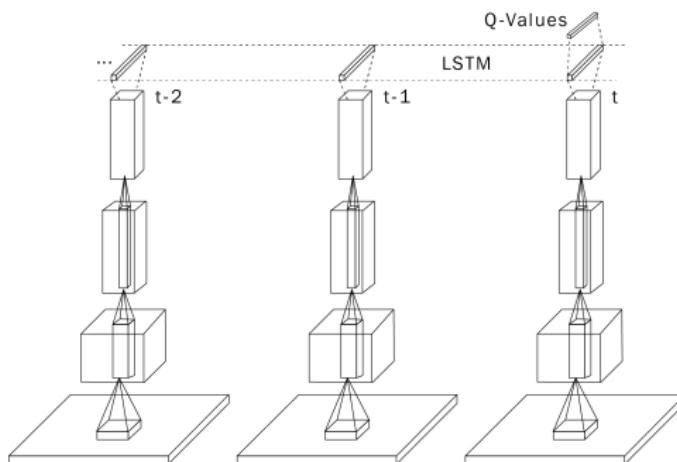
$$\delta_t = R_t^{(n)} + \gamma_t^n \max_{a'} q_{\bar{\theta}}(S_{t+n}, a') - q_{\theta}(S_t, A_t)$$

- ▶ Prioritized Experience Replay
- ▶ Double Q-Learning
- ▶ Dueling DQN
- ▶ DQN Distributionnel
- ▶ Noisy Nets



*DRQN : Deep Recurrent Q-Network [HS15]

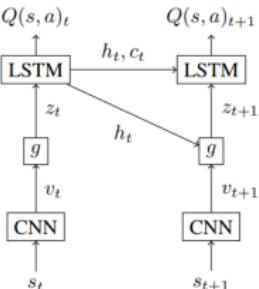
Prise en compte de l'historique de la trajectoire



- 👍 Bien adapté pour les POMDP : mémoire du RNN correspond à une représentation du monde selon les observations passées
- 👍 Permet de dépasser l'hypothèse de Markov d'ordre 1
- 👎 Plus difficile à apprendre

*DARQN : Deep Attention Recurrent Q-Network [Sor+15]

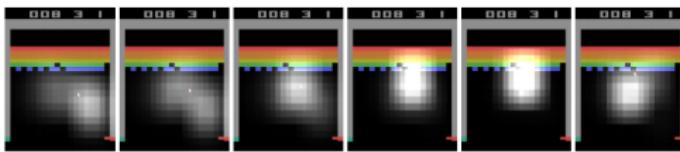
Prise en compte de l'historique de la trajectoire + attention sur des zones de l'entrée



avec :

- ▶ $v_t = (v_t^1, v_t^2, \dots, v_t^L)$ les L cartes de sorties du CNN au temps t ;
- ▶ $g(v_t^i, h_{t-1}) = \exp(\theta(\tanh(\phi v_t^i + \omega h_{t-1}))) / Z$, où θ , ϕ et ω sont des paramètres appris et Z une constante de normalisation (softmax) ;
- ▶ $z_t = \sum_{i=1}^L g(v_t^i, h_{t-1}) v_t$

👍 Par rapport à DRQN : permet de se focaliser sur des zones de l'entrée en fonction de l'état du RNN



Sources

- ▶ Ludovic Denoyer (cours à l'ENSIIE)
- ▶ Pieter Abbeel (UC Berkeley, Spring 2013)
- ▶ Sergey Levine (UC Berkeley, Spring 2017)
- ▶ Reinforce.js : https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html
- ▶ Milica Gasic : <http://mi.eng.cam.ac.uk/~mg436/LectureSlides/MLSALT7/L3.pdf>
- ▶ Jonathan Hui : https://medium.com/@jonathan_hui/rl-dqn-deep-q-network-e207751f7ae4
- ▶ Ziad Salloum : <https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3>

References I

- [BDM17] Marc G BELLEMARE, Will DABNEY et Rémi MUNOS. “A distributional perspective on reinforcement learning”. In : *arXiv preprint arXiv :1707.06887* (2017).
- [DFR98] Richard DEARDEN, Nir FRIEDMAN et Stuart RUSSELL. “Bayesian Q-learning”. In : *Aaaai/iaai. 1998*, p. 761-768.
- [For+17] Meire FORTUNATO et al. “Noisy networks for exploration”. In : *arXiv preprint arXiv :1706.10295* (2017).
- [Has10] Hado V HASSELT. “Double Q-learning”. In : *Advances in neural information processing systems. 2010*, p. 2613-2621.
- [Hes+17] Matteo HESSEL et al. “Rainbow : Combining improvements in deep reinforcement learning”. In : *arXiv preprint arXiv :1710.02298* (2017).
- [HS15] Matthew HAUSKNECHT et Peter STONE. “Deep recurrent q-learning for partially observable mdps”. In : *arXiv preprint arXiv :1507.06527* (2015).
- [Mni+13] Volodymyr MNIH et al. “Playing atari with deep reinforcement learning”. In : *arXiv preprint arXiv :1312.5602* (2013).
- [Sch+15] Tom SCHAUL et al. “Prioritized experience replay”. In : *arXiv preprint arXiv :1511.05952* (2015).

References II

- [Sor+15] Ivan SOROKIN et al. “Deep attention recurrent Q-network”. In : *arXiv preprint arXiv :1512.01693* (2015).
- [Wan+16] Ziyu WANG et al. “Dueling network architectures for deep reinforcement learning”. In : *International conference on machine learning*. 2016, p. 1995-2003.