Machine Learning

Apprentissage par renforcement

Algorithmes classiques

Marc Métivier

marc.metivier@u-paris.fr

Université Paris Cité

L'apprentissage par renforcement (RL)

La programmation dynamique s'applique à un MDP connu

Que faire si le MDP est inconnu ?

➠ ***Apprentissage par renforcement***

Deux catégories de méthodes

- ***Apprentissage par renforcement direct*** ("model-free" en anglais)
  - Pas de modèle de l'environnement
  - Apprendre une fonction valeur (et/ou une politique) par l'expérience
- ***Apprentissage par renforcement indirect*** ("model-based" en anglais)
  - Apprendre un modèle de l'environnement par l'expérience
  - Apprendre une fonction valeur (et/ou une politique) à partir du modèle

Deux objectifs possibles

Pour la prédiction :

- Evaluer une politique dans l'environnement
- Compte tenu d'une certaine politique π, estimer V^π
- Algos : Prédiction Monte-Carlo, TD-Learning, ...

Pour le contrôle :

- Optimiser le comportement d'un agent dans l'environnement
- Estimer Q^* afin de trouver une politique optimale
- Algos : Contrôle Monte-Carlo, Sarsa, Q-learning, Dyna, MCTS...

Algorithmes pour la prédiction

Méthodes Monte-Carlo (MC)

Principe : apprendre à partir d'épisodes d'interaction avec l'environnement

- Ne nécessite aucune connaissance à priori du MDP
- Apprentissage direct : pas besoin de modéliser le MDP
- L'apprentissage se fait après des épisodes d'interaction
- Les valeurs d'état calculées sont simplement le retour moyen

⚠ **Quelques contraintes :**

- Les épisodes doivent être complets : pas de bootstrapping
- Tous les épisodes doivent se terminer (MDPs "épisodiques")

Prédiction Monte-Carlo

Soit l'épisode : $(s_1, a_1, r_2, \dots, r_T, s_T)$

- Pour chaque état s_t avec $t = 1, \dots, T$
  - Calculer du retour $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T$
  - Incrémenter le compteur $N(s_t) \leftarrow N(s_t) + 1$
  - Incrémenter le retour total $S(s_t) \leftarrow S(s_t) + G_t$
  - La valeur est estimée comme le retour moyen $V(s_t) = S(s_t)/N(s_t)$

Convergence : $V(s) \rightarrow V^\pi(s)$ quand $N(s) \rightarrow \infty$ (loi des grands nombres)

Implémentation : parcourir l'épisode depuis la fin pour calculer G_t itérativement

Moyenne incrémentale

Les moyennes successives $\mu_1, \mu_2, \ldots$ d'une séquence $x_1, x_2, \ldots$ peuvent être calculées incrémentalement :

$$\begin{aligned}
\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\
&= \frac{1}{k}\Big(x_k + \sum_{j=1}^{k-1} x_j\Big) \\
&= \frac{1}{k}(x_k + (k-1)\mu_{k-1}) \\
&= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})
\end{aligned}$$

Mise à jour MC incrémentale

Mettre à jour V incrémentalement après l'épisode $(s_1, a_1, r_2, \ldots, r_T, s_T)$

- Pour chaque état S_t avec le retour G_t

$$N(s_t) \leftarrow N(s_t) + 1$$

$$V(s_t) \leftarrow V(s_t) + \frac{1}{N(s_t)}(G_t - V(s_t))$$

Variante avec une moyenne mobile :

- Dans les ***problèmes non-stationnaires***, il peut être utile de calculer une ***moyenne mobile***, i.e. oublier les anciens épisodes :

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$$

Temporal-Difference Learning (TD Learning)

Principe : apprendre directement pendant l'interaction avec l'environnement

- Ne nécessite aucune connaissance à priori du MDP
- Apprentissage direct : pas besoin de modéliser le MDP
- L'apprentissage se fait pendant les épisodes d'interaction

TD n'a pas les contraintes de MC

- TD apprend à partir d'épisodes ***incomplets***, par ***bootstrapping***
- TD met à jour une supposition vers une supposition

TD Learning

Algorithme le plus simple : TD(0)

Soit l'épisode : $s_1, a_1, r_2, \ldots, s_T$

- Pour chaque état s_t avec $t = 1, \ldots, T-1$:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

Terminologie :

- $r_{t+1} + \gamma V(s_{t+1})$ est appelée la ***cible TD*** (***TD target***)
- $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ est appelé l'***erreur TD*** (***TD error***)
- α est le ***taux d'apprentissage*** (***learning rate***)

Avantages et désavantages de MC vs. TD

L'utilisation du retour final

TD peut apprendre *avant* de connaître le retour final

- TD peut apprendre en-ligne (online) après **chaque transition**
- MC doit attendre la fin de l'épisode afin que le retour soit connu

TD peut apprendre *sans* le retour final

- TD peut apprendre à parti d'épisodes **incomplets**
- MC peut seulement apprendre à partir d'épisodes complets
- TD fonctionne dans les environnements **sans terminaison**
- MC fonctionne seulement dans les environnements **épisodiques** (avec terminaison)

Compromis Biais/Variance

Biais :

- Le ***retour*** G_t est un estimateur **non biaisé** de $V^\pi(s_t)$
- La ***vraie cible TD*** $(r_{t+1} + \gamma V^\pi(s_{t+1}))$ est un estimateur **non biaisé** de $V^\pi(s_t)$
- La ***cible TD*** $(r_{t+1} + \gamma V(s_{t+1}))$ est un estimateur **biaisé** de $V^\pi(s_t)$

Variance :

- La ***cible TD*** a une variance beaucoup **plus faible** que le ***retour*** :
  - Le ***retour*** dépend de beaucoup d'actions, transitions et retours
  - La ***cible TD*** ne dépend que d'une action, une transition et un retour

Avantages et désavantages de MC vs. TD (2)

Compromis Biais/Variance

MC a une grande variance et pas de biais

- Bonnes propriétés de convergence
  - même avec de l'approximation de fonction
- Peu sensible aux valeurs initiales
- Très simple à utiliser et à comprendre

TD a une variance faible et un biais

- Généralement plus efficace que MC
- TD(0) converges vers $V^{\pi}(s)$
  - mais pas toujours avec de l'approximation de fonction
- Plus sensible au valeurs initiales

Avantages et désavantages de MC vs. TD (3)

La propriété de Markov

TD exploite la propriété de Markov

- Généralement plus efficace dans les environnements markoviens

MC n'exploite pas la propriété de Markov

- Généralement plus efficace dans les environnements non-markoviens

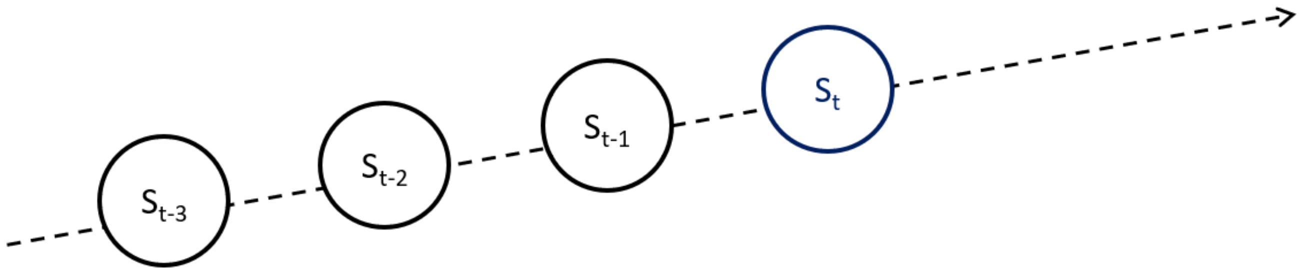
A propos de l'erreur TD

Dans TD(0), une seule valeur est mise à jour par cycle

- Dans l'état s_t, on calcule δ_t et on met à jour $V(s_t)$

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$$V(s_t) \leftarrow V(s_t) + \alpha \delta_t$$



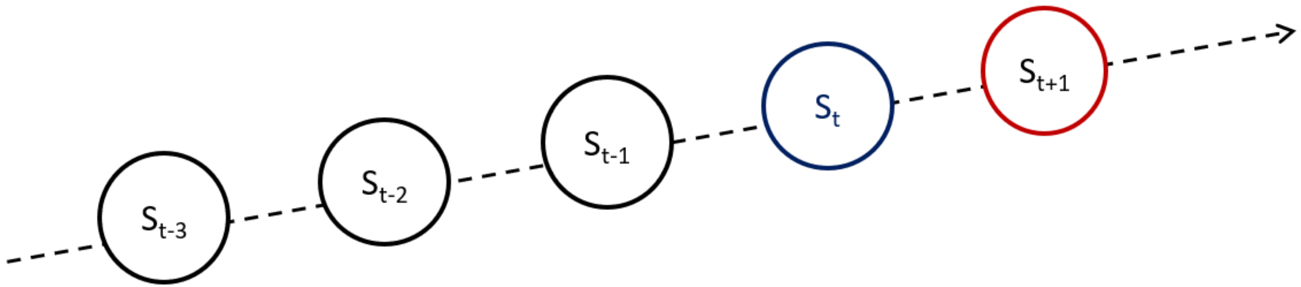
A propos de l'erreur TD

Dans TD(0), une seule valeur est mise à jour par cycle

- Dans l'état s_t, on calcule δ_t et on met à jour $V(s_t)$

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$$V(s_t) \leftarrow V(s_t) + \alpha\delta_t$$



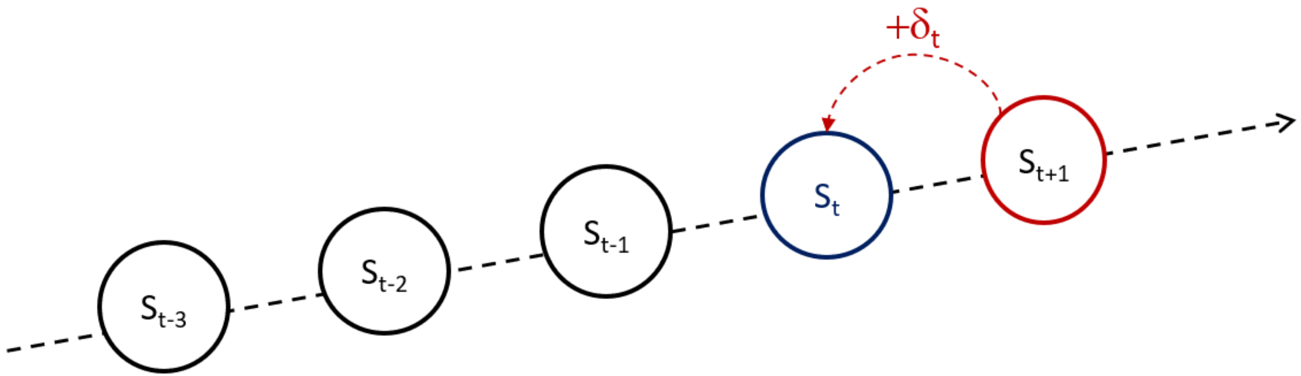
A propos de l'erreur TD

Dans TD(0), une seule valeur est mise à jour par cycle

- Dans l'état s_t, on calcule δ_t et on met à jour $V(s_t)$

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$$V(s_t) \leftarrow V(s_t) + \alpha\delta_t$$



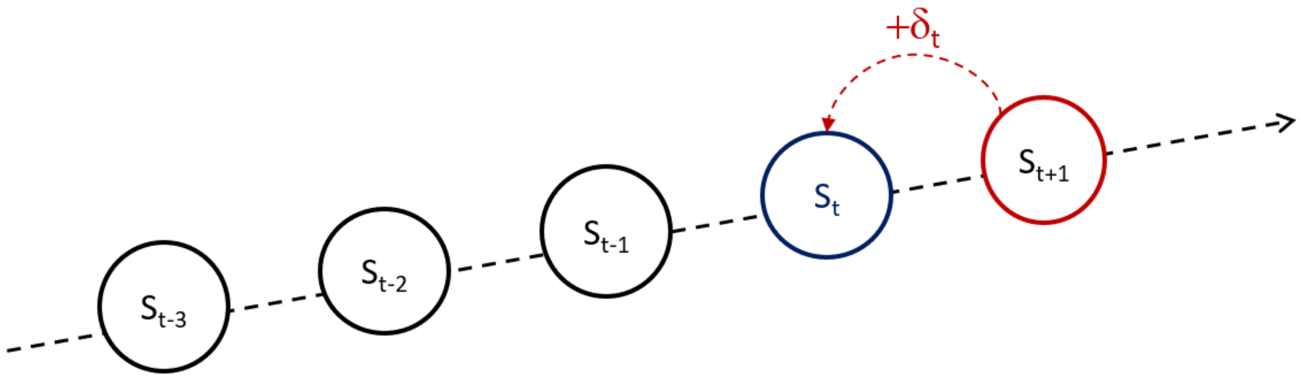
A propos de l'erreur TD

Dans TD(0), une seule valeur est mise à jour par cycle

- Dans l'état s_t, on calcule δ_t et on met à jour $V(s_t)$

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$$V(s_t) \leftarrow V(s_t) + \alpha\delta_t$$



➠ Les valeurs des états précédents ne sont plus correctes

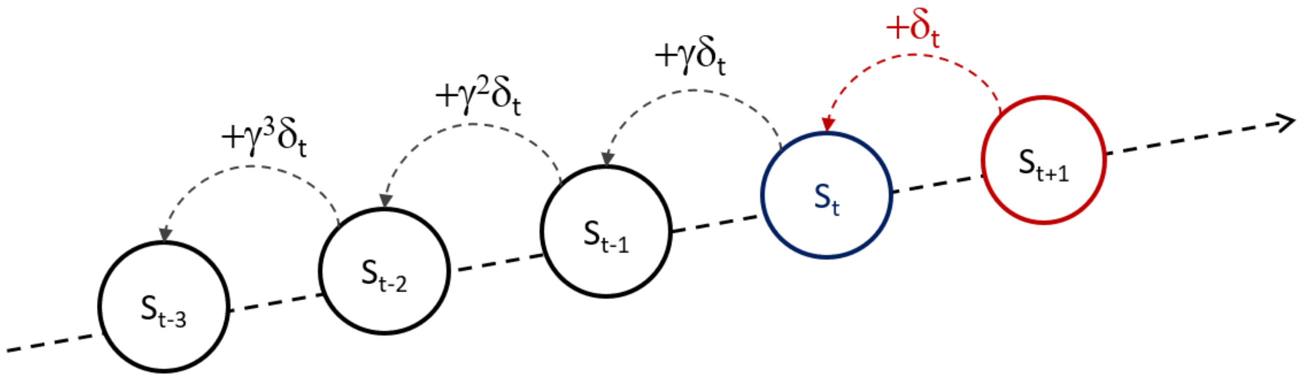
A propos de l'erreur TD

Dans TD(0), une seule valeur est mise à jour par cycle

- Dans l'état s_t, on calcule δ_t et on met à jour $V(s_t)$

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$$V(s_t) \leftarrow V(s_t) + \alpha \delta_t$$



➡ ***Amélioration possible*** : ***propager l'erreur aux valeurs précédentes***

Utiliser un retour n-step

Le *retour n-step* : un retour à horizon n

$$G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n})$$

La mise à jour n-step :

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

En conséquence :

- Il faut attendre l'étape $t+n$ pour mettre à jour $V(s_t)$
- TD(0) utilise un retour à horizon 1 : $G_t^{(1)}$
- MC utilise un retour à horizon "infini" (fin de l'épisode)

Algorithme TD(λ)

Notions préliminaires

Un paramètre $\lambda \in [0, 1]$

- Il définit une sorte de ***taux de non bootstrapping***

Une *trace d'éligibilité* $E : \mathcal{S} \rightarrow \mathbb{R}$

- Initialisation : $E_0(s) = 0 \;\; \forall s \in \mathcal{S}$
- Mise à jour :

$$E_t(s) = \begin{cases} \gamma\lambda E_{t-1}(s) & \text{si } S_t \neq s \\ \gamma\lambda E_{t-1}(s) + 1 & \text{si } S_t = s \end{cases}$$

Elle donne le degré de mise à jour à faire dans chaque état pour l' erreur δ_t

Algorithme TD(λ)

Principe

Pour chaque état s_t

- Calculer l'erreur TD

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

- Mettre à jour la trace d'éligibilité pour **tous les états**

$$\forall s \in \mathcal{S}, \quad E(s) \leftarrow \begin{cases} \gamma\lambda E(s) & \text{si } S_t \neq s \\ \gamma\lambda E(s) + 1 & \text{si } S_t = s \end{cases}$$

- Mettre la valeur de **tous les états**

$$\forall s \in \mathcal{S}, \quad V(s) \leftarrow V(s) + \alpha\delta_t E(s)$$

Algorithme TD(λ)

Un compromis entre MC et TD

Si $\lambda = 0$: il s'agit exactement de TD(0)

Si $\lambda = 1$: l'algorithme est équivalent à MC

Si $\lambda \in]0, 1[$:

- L'algorithme utilise un "***retour*** λ" qui est un mélange des retours n***-step***

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

Algorithmes pour le contrôle

Apprentissage pour le contrôle

Exemples de problèmes pouvant être modélisés par des MDPs

- Elevator (ascenseurs de gratte-ciel)
- Helicopter (manoeuvres d'hélicoptères)
- Aeroplane Logistics
- Robocup Soccer
- Quake
- Protein Folding
- Robot walking
- Game of Go

Dans le plupart des ces problèmes :

- **soit** le MDP est **inconnu**
- **soit** le MDP est **connu mais trop grand** pour être utilisé

➡ l'apprentissage se fait sur de l'expérience obtenue par échantillonnage

On/Off-Policy Learning

Deux manières d'apprendre par l'expérience :

- ***On-policy learning*** :
    - "Apprendre depuis le travail"
    - Apprendre sur la politique π en échantillonnant à partir de π
- ***Off-policy learning*** :
    - "Regarder au dessus de l'épaule de quelqu'un d'autre"
    - Apprendre sur la politique π en échantillonnant à partir d'une politique μ

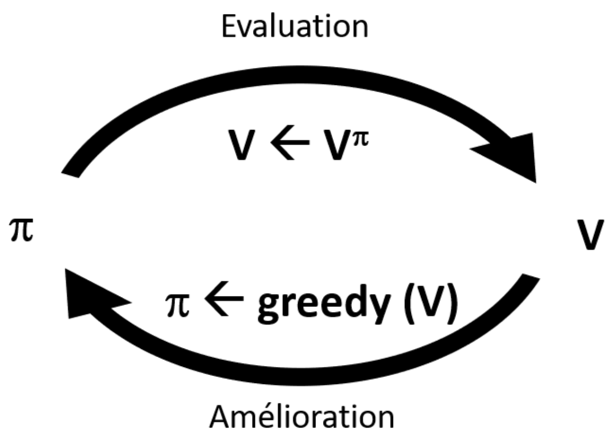
Algo des politiques itérées avec Monte-Carlo

Puisque MC permet de calculer V, on peut l'utiliser dans l'algorithme PI ?

On alterne :

- ***Evaluation*** de la politique π
  - Utiliser MC pour évaluer V^{π} ?
- ***Amélioration*** de la politique π
  - Amélioration Greedy(V) ?

jusqu'à convergence de π et V^{π}...



Ne fonctionnera pas....

Monte-Carlo Policy Iteration

Le problème de la fonction V

Amélioration *greedy*(V)

$$\pi'(a \mid s) = \begin{cases} 1 & \text{si } a = \arg\max_{a \in \mathcal{A}} Q^{\pi}(s,a) \\ 0 & \text{sinon.} \end{cases}$$

avec :

$$Q^{\pi}(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s,a) \Big[\mathcal{R}(s,a,s') + \gamma V^{\pi}(s') \Big]$$

⚠ **On ne dispose pas *à priori* des fonctions $\mathcal{P}$ et $\mathcal{R}$!**

➡ **MC doit être utilisé pour estimer Q^{π} plutôt que V^{π}**

Monte-Carlo Policy Iteration

Le problème de Greedy pour la sélection d'action

Un exemple : l'agent est face à deux portes

- Il ouvre la porte de gauche et reçoit un renforcement de 0
  - Q(gauche) = 0
- Il ouvre la porte de droite et reçoit un renforcement de +1
  - Q(droite) = 1
- Il ouvre la porte de droite et reçoit un renforcement de +3
  - Q(droite) = 2
- Il ouvre la porte de droite et reçoit un renforcement de +2
  - Q(droite) = 2
- Il continuera d'ouvrir la porte de droite indéfiniment...

Etes-vous sûr qu'il a choisi la meilleure des portes ?

Le dilemme exploration / exploitation

Si l'agent choisit toujours l'actions qui maximise Q :

- Il aura tendance à toujours suivre le même chemin
- Il risque de ne pas rencontrer pas les états menant à un meilleur chemin

La prise de décision face à l'incertain implique un choix fondemental :

- ***Exploitation*** : choisir la meilleure action à partir des connaissances actuelles
- ***Exploration*** : agir de manière à obtenir plus de connaissances

⚠ **Trouver la meilleure stratégie à long-terme peut nécessiter des sacrifices à court-terme**

- Certaines informations essentielles ne peuvent être rencontrées qu'après un certain temps d'exploration

Exploration ϵ-greedy

Une idée simple pour assurer une exploration continue

- Principe :
  - Avec une probabilité $1-\epsilon$: choisir l'action $greedy$
  - Avec une probabilité ϵ : choisir l'action au hasard
- Toutes les m actions ont une probabilité non-nulle d'être choisie

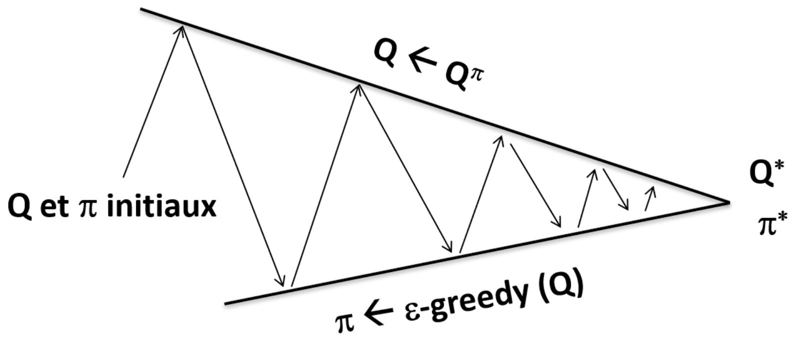
La politique ϵ-greedy :

$$\pi(a \mid s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{si } a = \arg\max_{a \in \mathcal{A}} Q(s,a) \\ \epsilon/m & \text{sinon.} \end{cases}$$

Monte-Carlo Policy Iteration

➡ On utilise Q et ϵ-greedy

Algorithme MC-PI



On alterne :

- ***Evaluation*** de la politique π
  - Estimation de Q^π par Monte-Carlo
- ***Amélioration*** de la politique π
  - Amélioration par ϵ-Greedy(Q^π)

jusqu'à convergence de π et Q^π...

Le contrôle Monte-Carlo

L'évaluation de politique peut mettre du temps à converger

- Il faut plusieurs épisodes pour estimer Q
- Mais le but n'est pas d'estimer Q mais de trouver une politique optimale

⇛ On améliore la politique ***après chaque épisode***

Principe

- Pour **chaque épisode** :
  - Evaluation Monte-Carlo de la politique (mise à jour de Q)
  - Amélioration de la politique par ϵ-greedy

Convergence : l'algorithme converge à condition que la méthode d'amélioration de politique soit de type **GLIE**, i.e. ***Greedy in the Limit with Infinite Exploration***

GLIE

Définition

Propriété ***Greedy in the Limit with Infinite Exploration*** (GLIE)

- Toutes les paires état-action sont explorées une infinité de fois

$$\lim_{k \to \infty} N_k(s, a) = \infty$$

- La politique ***converge*** vers une politique ***greedy***

$$\lim_{k \to \infty} \pi_k(a \mid s) = 1(a = \arg\max_{a' \in \mathcal{A}} Q_k(s, a'))$$

ϵ-greedy est de type GLIE si ϵ évolue et tend vers 0

- Par exemple, si on spécifie $\epsilon_k = \frac{1}{k}$

Le contrôle Monte-Carlo GLIE

Algorithme GLIE Monte-Carlo Control

Pour $k = 1, 2, \ldots$

Générer un épisode en utilisant π : $\{s_1, a_1, r_2, \ldots, s_T\} \backsim \pi$

Pour chaque état s_t action a_t de l'épisode :

$$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{1}{N(s_t, a_t)}(G_t - Q(s_t, a_t))$$

Améliorer la politique :

$$\epsilon \leftarrow 1/k$$

$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

Théorème : le contrôle Monte-Carlo GLIE converge vers la fonction valeur optimale $Q(s, a) \rightarrow Q^*(s, a)$

Mise à jour TD pour le contrôle

Temporal-Difference pour la fonction valeur d'action Q

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\mathrm{CIBLE} - Q(s,a)]$$

Mise à jour en-ligne à chaque transition

La *cible TD* :

- Méthodes "On-Policy" :
  - La ***cibe TD*** est basée sur la politique actuelle
  - Algorithme : Sarsa
- Méthodes "Off-Policy" :
  - La ***cible TD*** est dérivée d'une autre politique (par ex. greedy)
  - Algorithme : Q-Learning

SARSA

Algorithme SARSA

Initialiser arbitrairement $Q(s,a)\ \ \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$

Initialiser $Q(s,\cdot) = 0\ \ \forall s$ terminal

Pour chaque épisode :

- $s \leftarrow$ état initial
- Choisir a dans s en utilisant une politique sur Q (par ex. ϵ-greedy)
- **Répéter**:
  - Exécuter a et observer r et s'
  - Choisir a' dans s' avec une politique sur Q (par ex. ϵ-greedy)
  - $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
  - $s \leftarrow s'$; $a \leftarrow a'$
- **Jusqu'à** (s terminal)

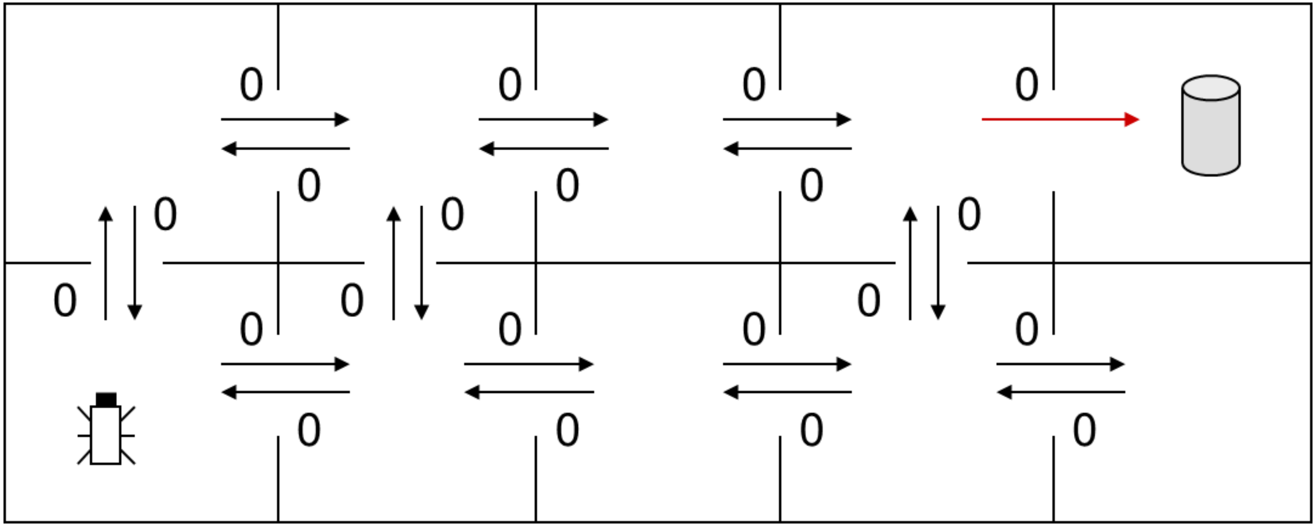
Convergence : SARSA converge vers la fonction optimale si la politique de choix de l'action est de type GLIE et sous certaines conditions sur α

SARSA

Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

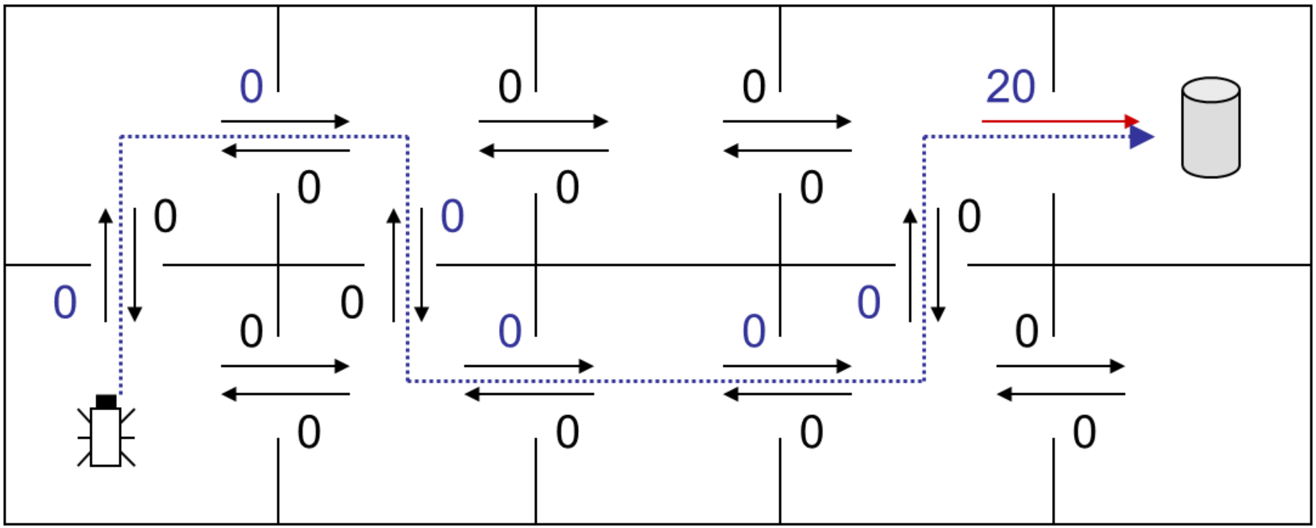
Initialisation



SARSA
Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

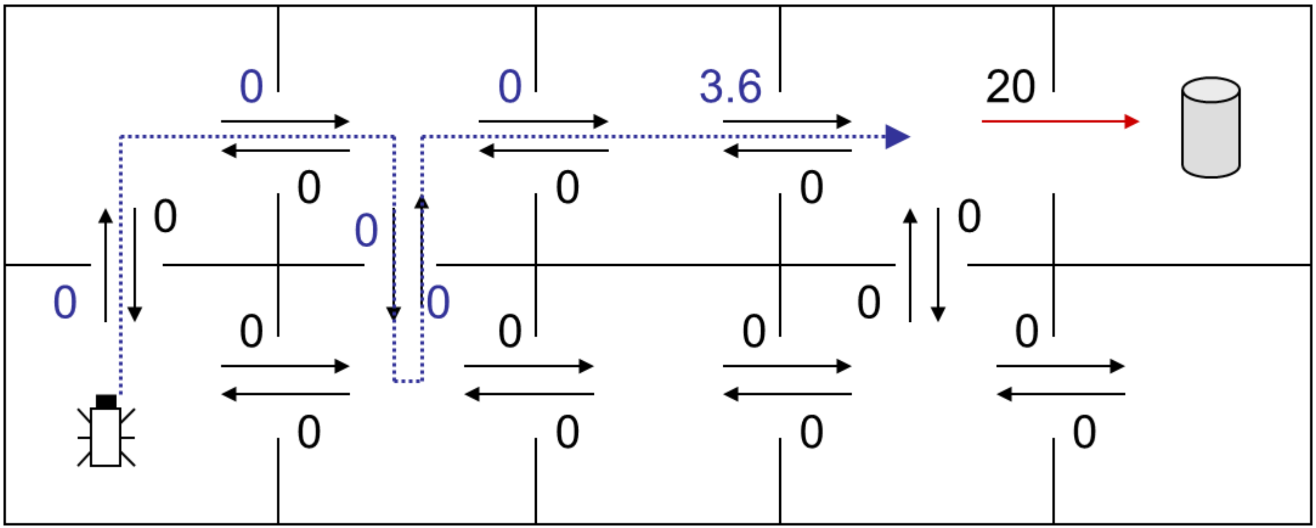
Episode 1



SARSA
Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

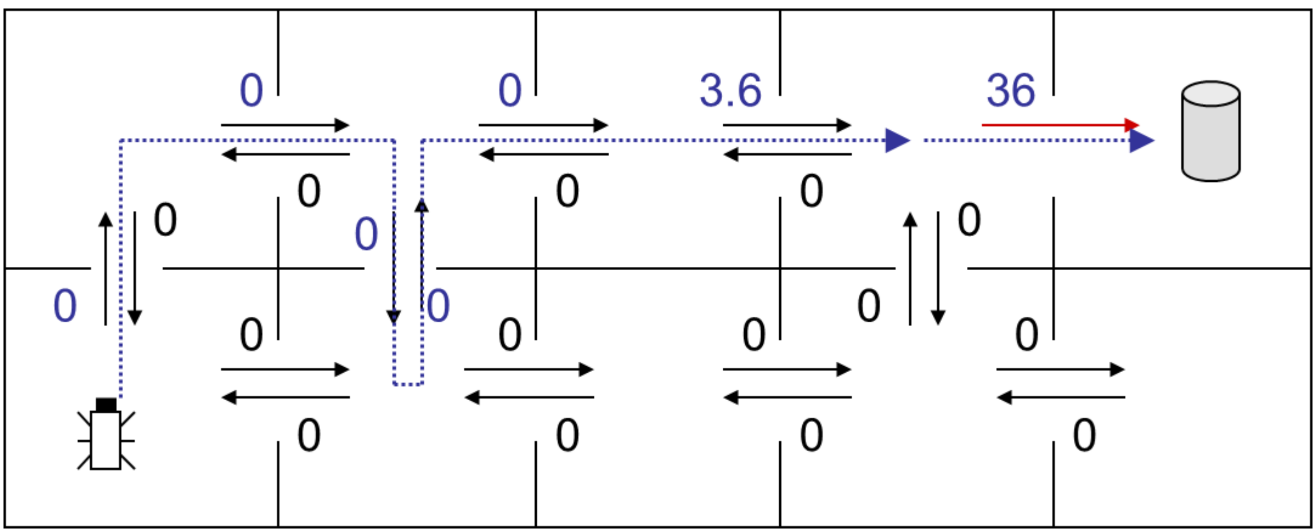
Episode 2



SARSA
Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

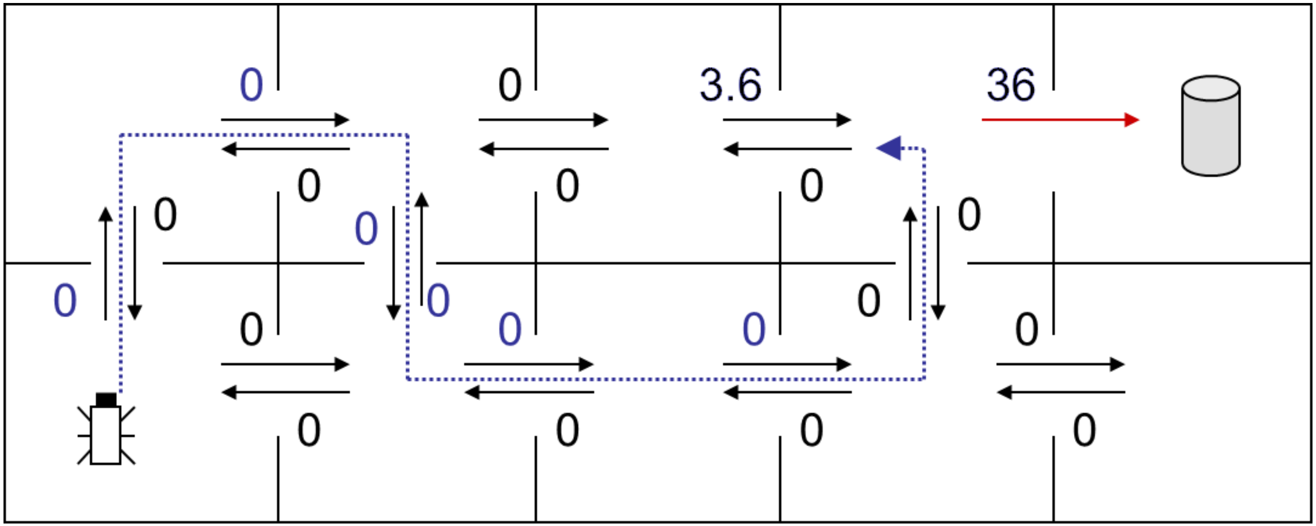
Episode 2



SARSA
Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

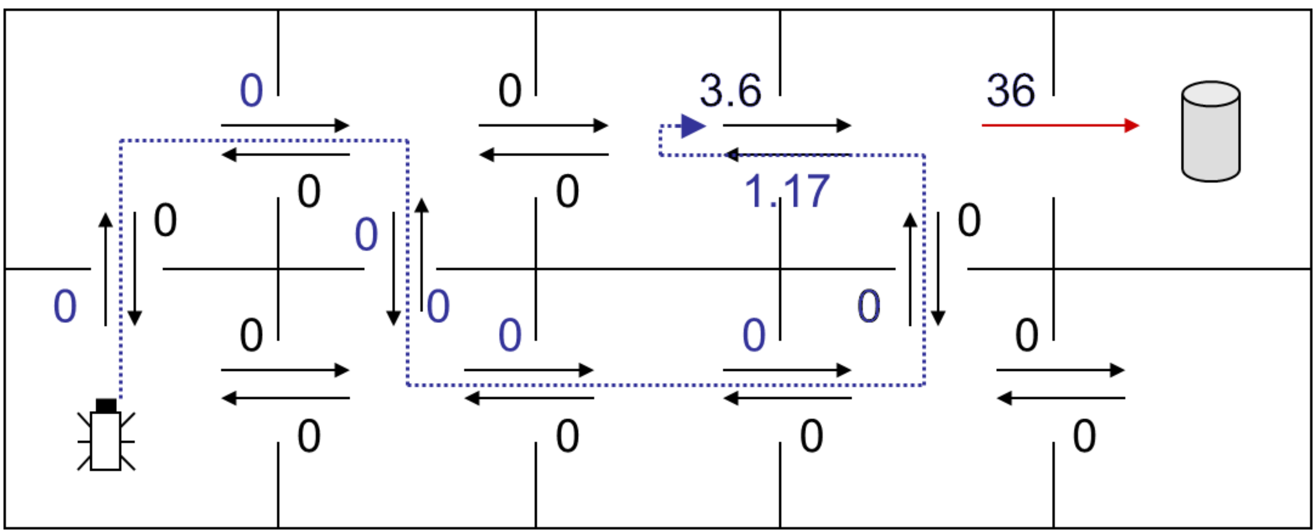
Episode 3



SARSA
Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

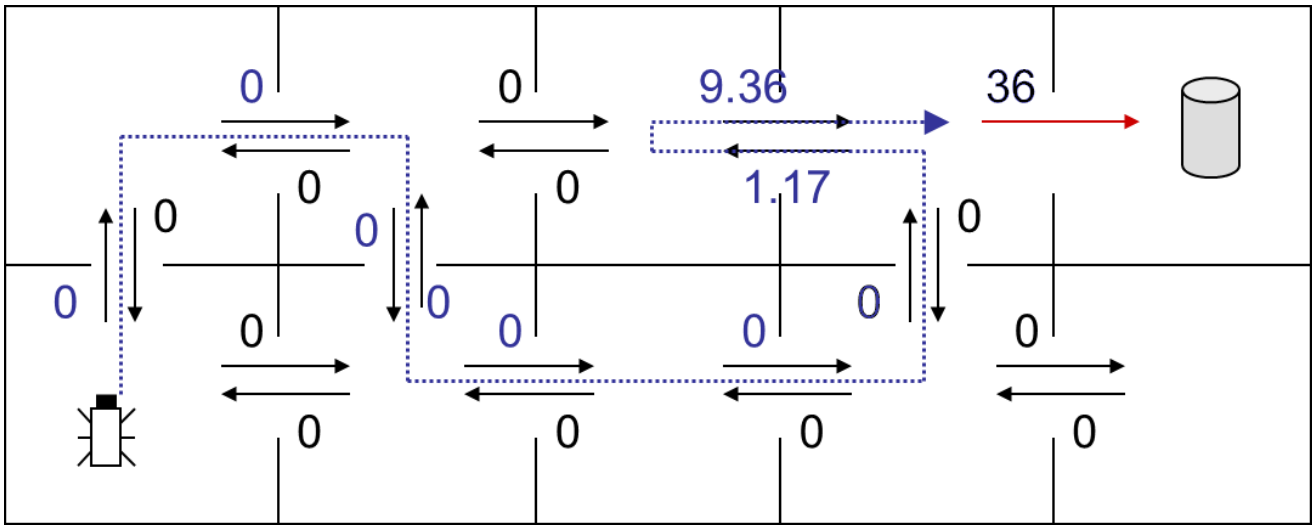
Episode 3



SARSA
Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

Episode 3

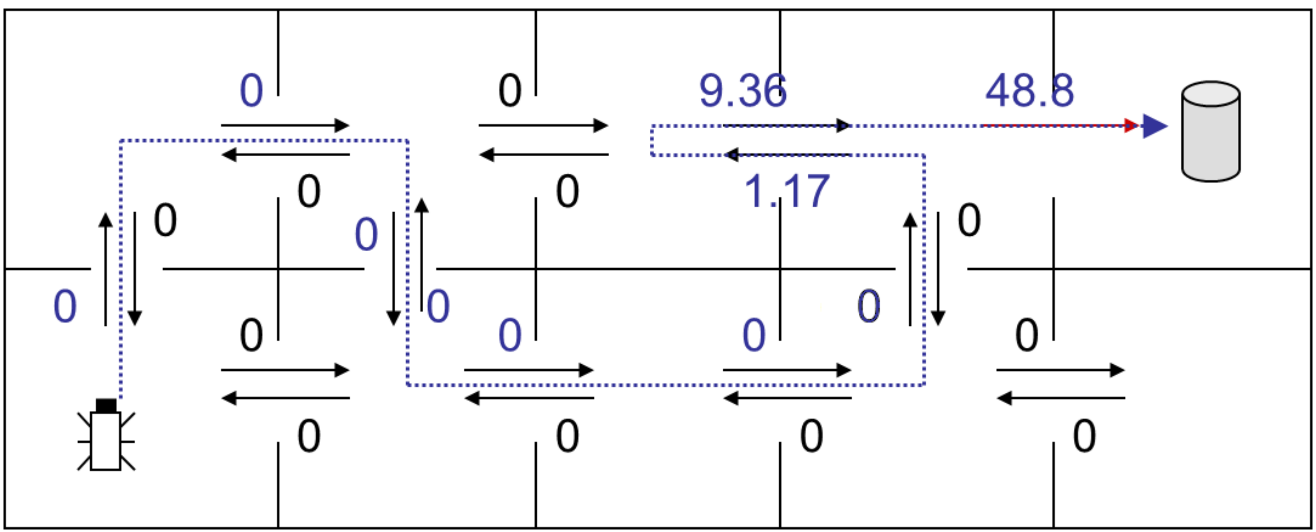


SARSA

Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

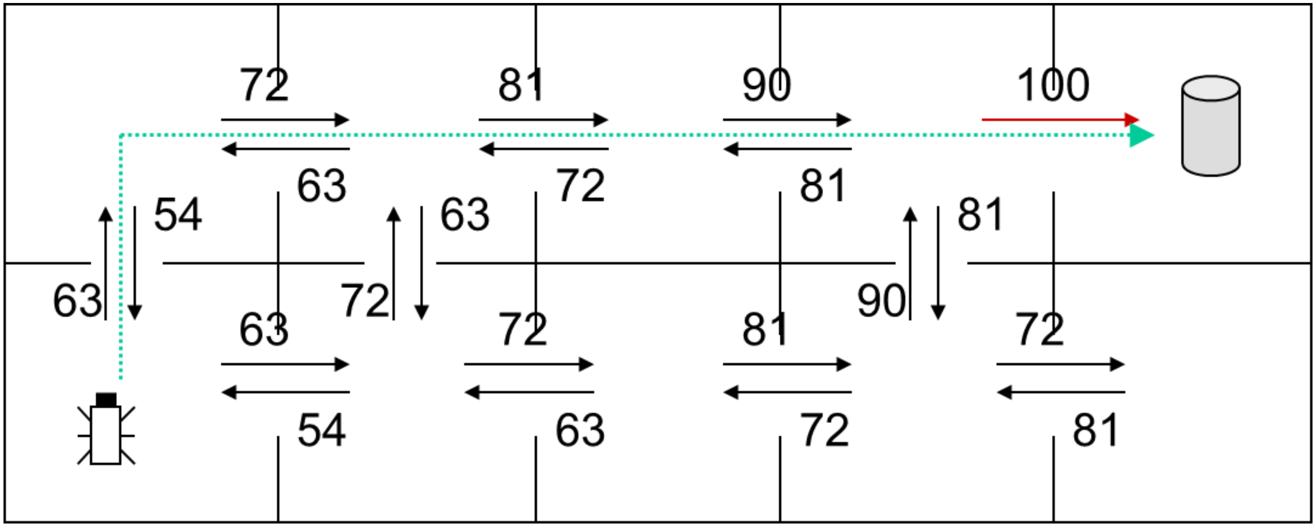
Episode 3



SARSA
Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

Après convergence...



Q-Learning

Algorithme Q-Learning

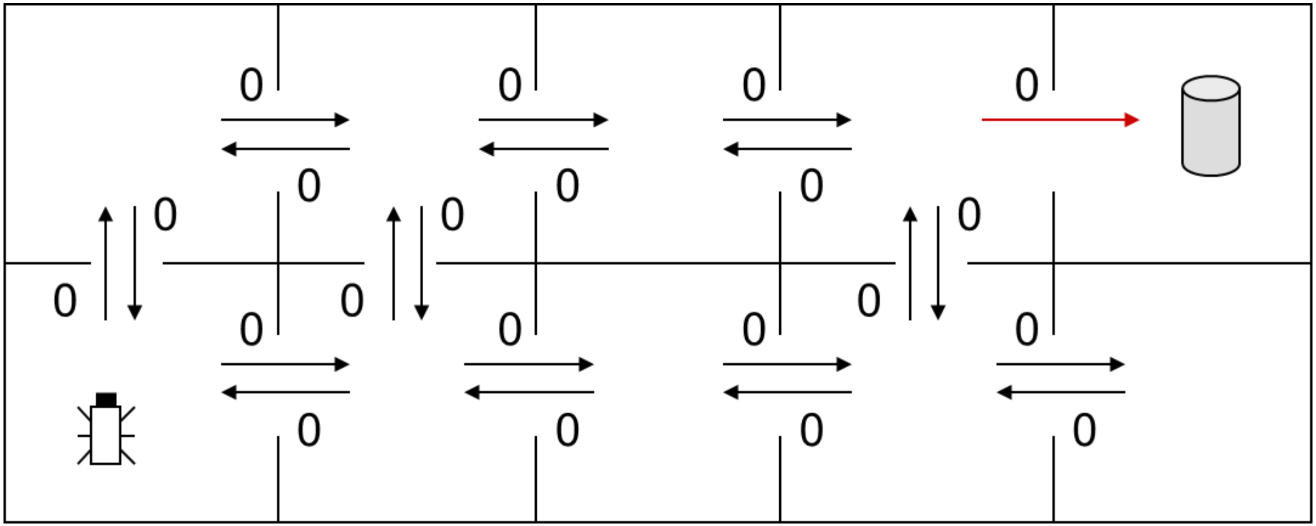
Initialiser arbitrairement $Q(s,a)\ \ \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$
Initialiser $Q(s,\cdot) = 0\ \ \forall s$ terminal
Pour chaque épisode :
  $s \leftarrow$ état initial
  Répéter:
    Choisir a dans s en utilisant une politique sur Q (par ex. ϵ-greedy)
    Exécuter a et observer r et s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$
  Jusqu'à (s terminal)

Convergence : Q-Learning converge vers la fonction optimale si la politique de choix de l'action est de type GLIE et sous certaines conditions sur α

Q-Learning
Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

Initialisation

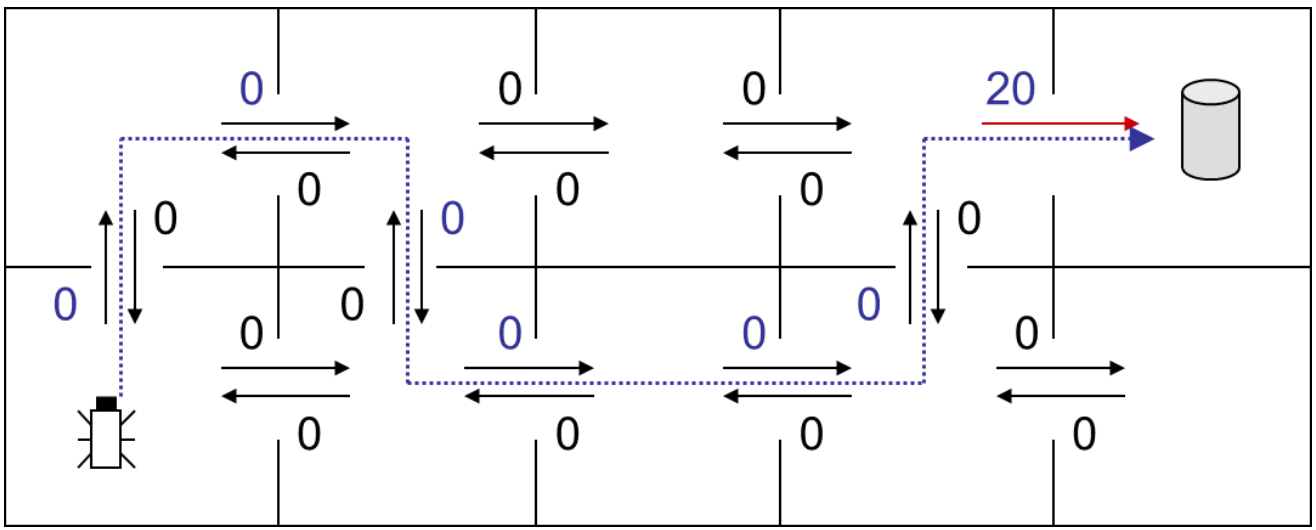


Q-Learning

Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

Episode 1

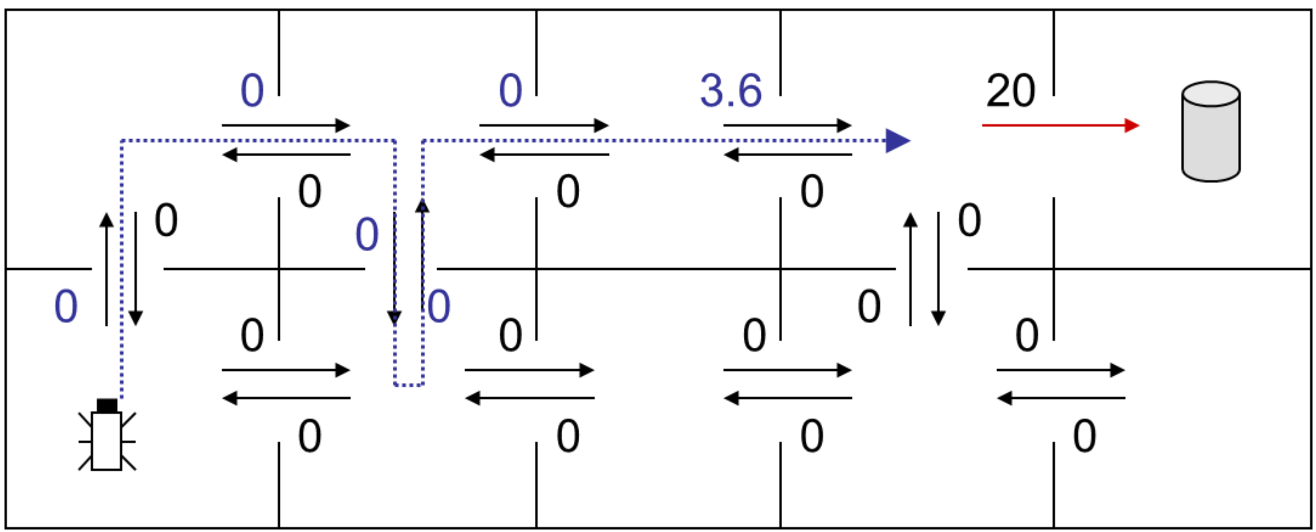


Q-Learning

Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

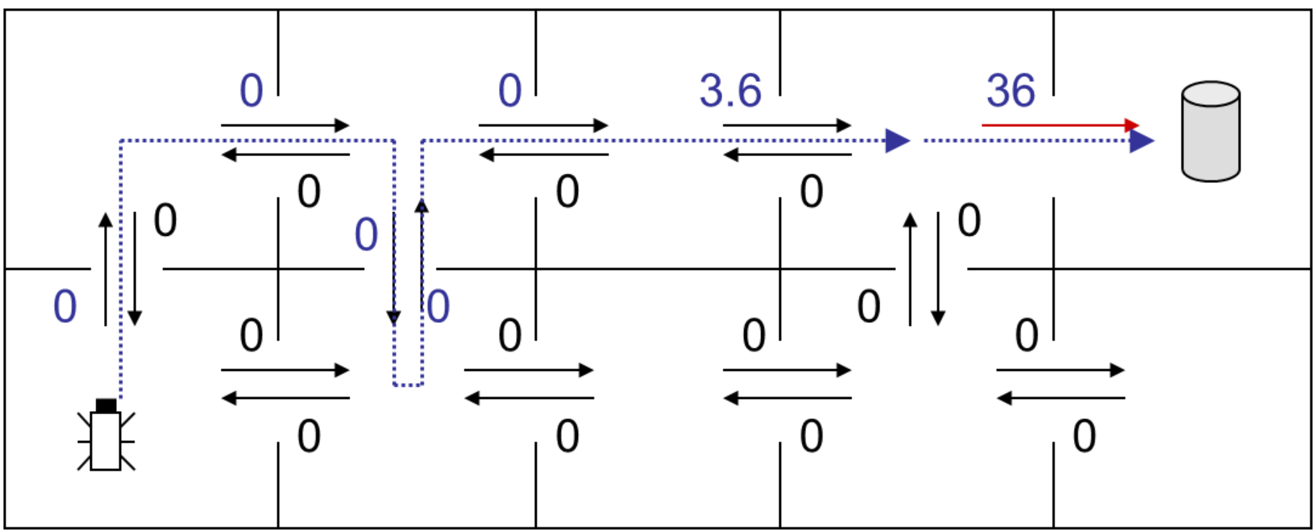
Episode 2



Q-Learning
Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

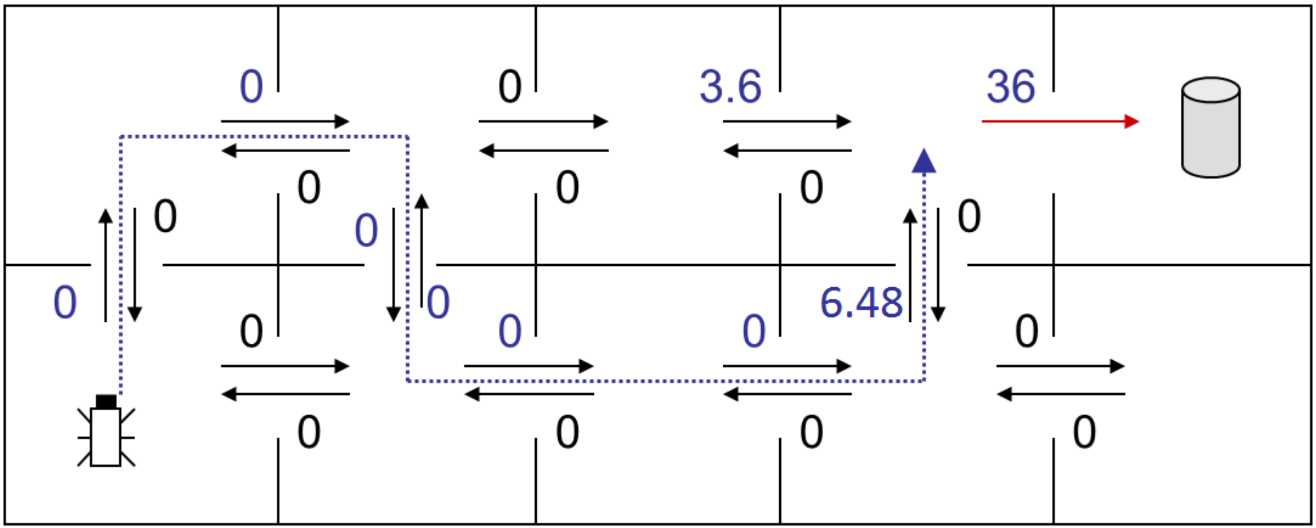
Episode 2



Q-Learning
Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

Episode 3

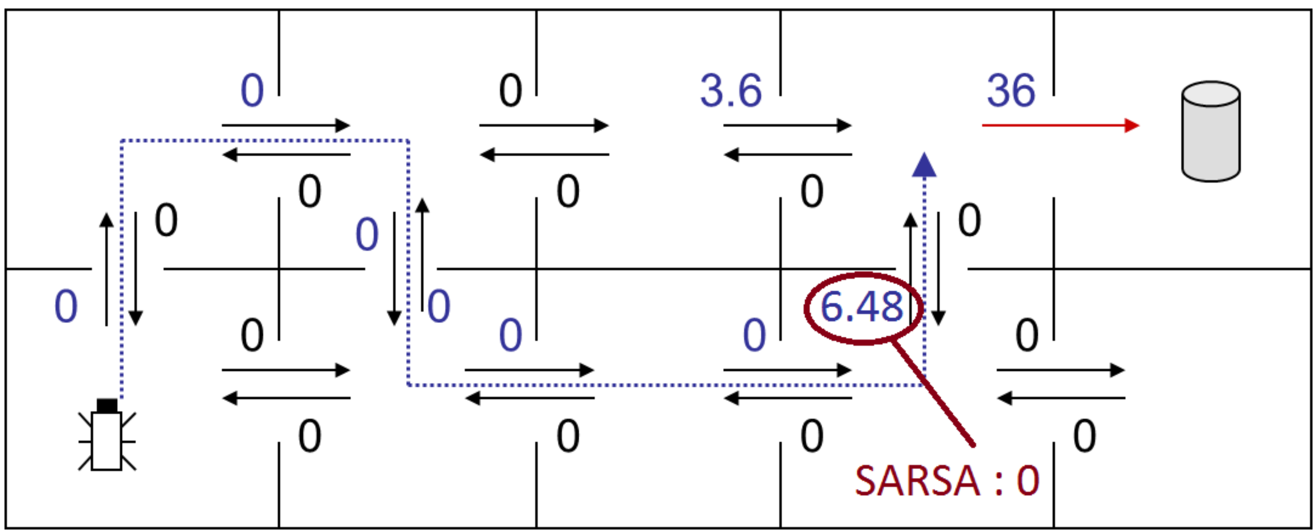


Q-Learning

Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

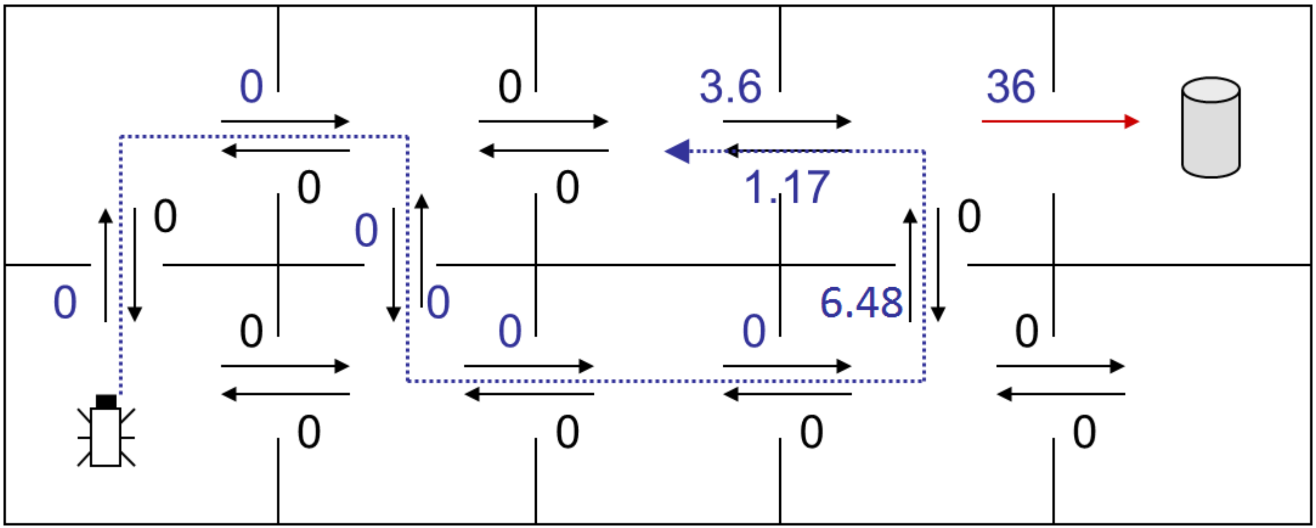
Episode 3



Q-Learning
Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

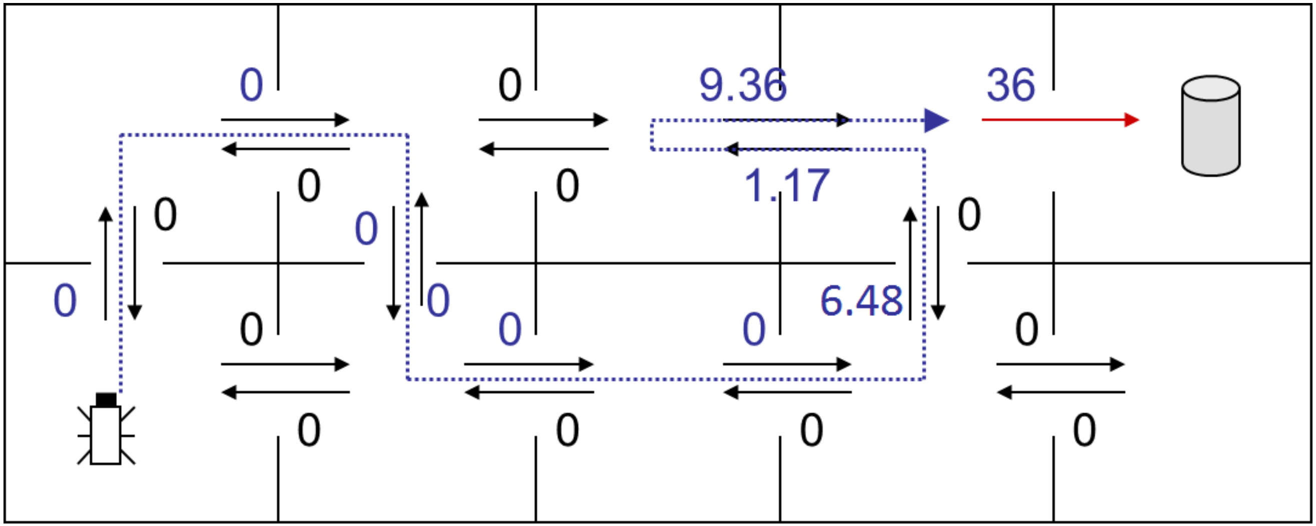
Episode 3



Q-Learning
Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

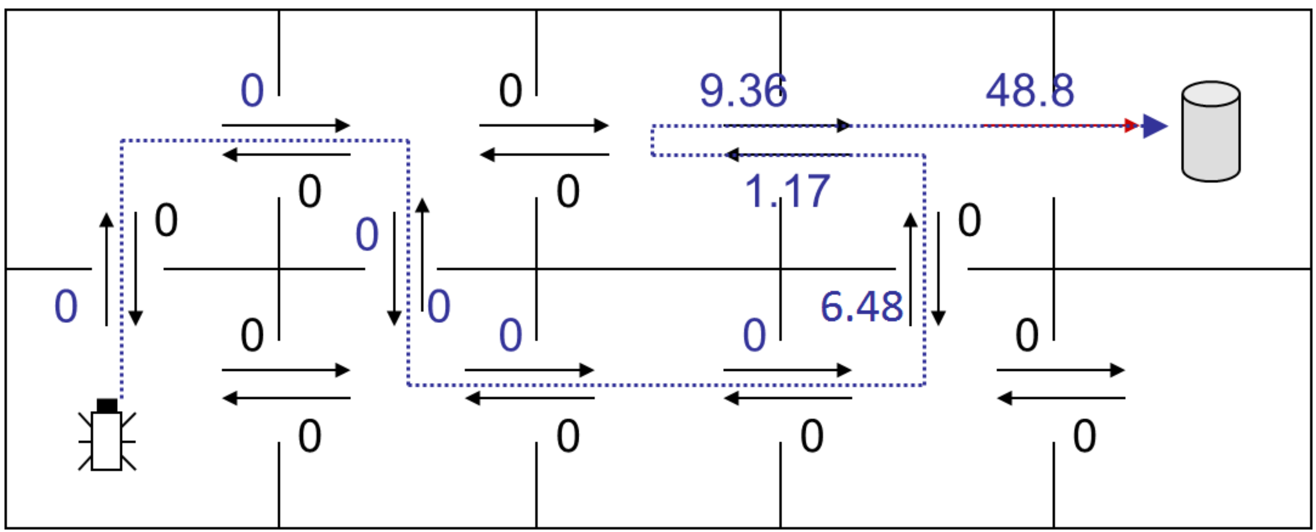
Episode 3



Q-Learning
Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

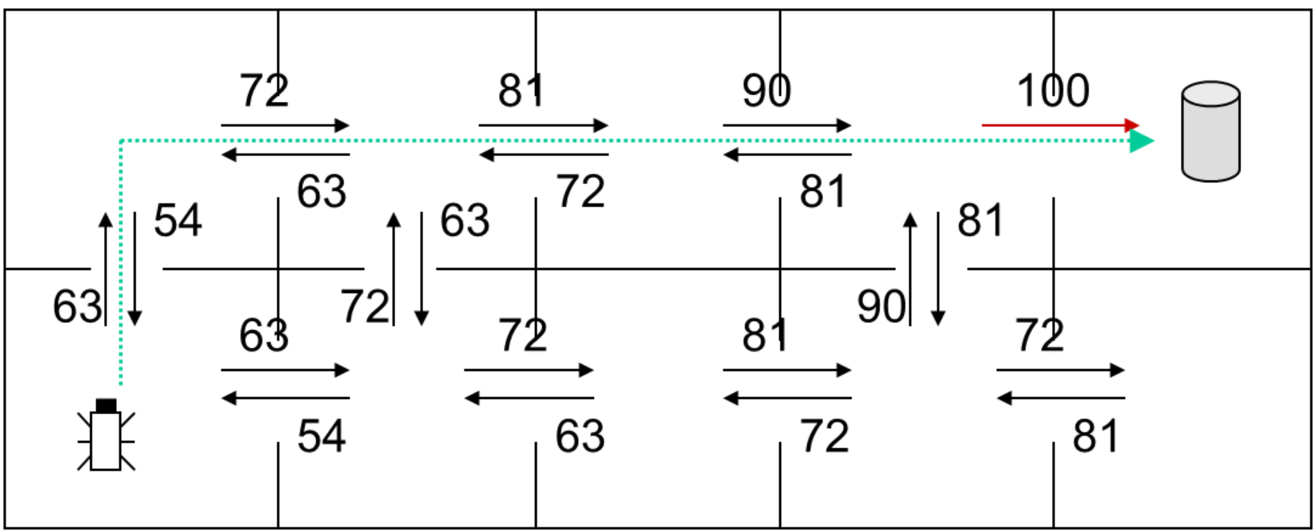
Episode 3



Q-Learning
Exemple

- Récompense de 100 à l'objectif (flèche rouge)
- Facteur d'amortissement : $\gamma = 0.9$
- Taux d'apprentissage : $\alpha = 0.2$

Après convergence...



Amélioration : mise à jour n-Step

Dans chaque état, attendre n actions avant de mettre à jour

- Soit $G_t^{(n)}$ le ***retour n-step***
- La mise à jour de Sarsa ou Q-learning devient :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[G_t^{(n)} - Q(s_t, a_t)]$$

Application à SARSA :

$$G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n Q(s_{t+n}, a_{s+n})$$

Application à Q-learning :

$$G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \max_{a' \in \mathcal{A}} Q(s_{t+n}, a')$$

Utilisation d'une trace d'éligibilité

SARSA et Q-Learning font une mise à jour à chaque transition

On peut ajouter une ***trace d'éligibilité*** pour propager les mises à jour

- Cette fois la trace concerne les couples état-action

$$E : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$$

- Initialisation : $E_0(s, a) = 0 \;\; \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$

- Mise à jour :

$$E_t(s,a) = \begin{cases} \gamma\lambda E_{t-1}(s,a) + 1 & \text{si } S_t = s \text{ et } A_t = a \\ \gamma\lambda E_{t-1}(s,a) & \text{sinon.} \end{cases}$$

SARSA(λ)

Passage de SARSA à SARSA(λ)

- On ajoute la trace d'éligibilité
- La mise à jour se fait sur tous les couples état-action

Pour chaque transition $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$:

$$\delta \leftarrow r + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$$

$$E(s_t, a_t) \leftarrow E(s_t, a_t) + 1$$

Pour tous les $s \in \mathcal{S}, a \in \mathcal{A}$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \delta E(s,a)$$
$$E(s,a) \leftarrow \gamma \lambda E(s,a)$$

Méthodes indirectes

Model-based Reinforcement Learning

Principe général :

- Apprendre un modèle de l'environnement
  - Estimer le MDP ou des éléments du MDP
- Utiliser la planification pour calculer un fonction valeurs ou une politique
  - Par exemple la programmation dynamique (DP)

Intégrer apprentissage et planification dans une même architecture

Dyna-Q

Algorithme Dyna-Q

Initialiser $Q(s,a)$ et $Model(s,a) \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$
Répéter pour chaque épisode:
- $s \leftarrow$ état courant observé
- Choisir a dans s en utilisant une politique sur Q (par ex. ϵ-greedy)
- Exécuter a et observer r et s'
- $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a)]$
- $Model(s,a) \leftarrow r, s'$ (en supposant un env. déterministe)
- **Répéter** n **fois**:
  - $t \leftarrow$ état aléatoire déjà observé
  - $a \leftarrow$ action déjà exécutée dans t
  - $r, t' \leftarrow Model(s,a)$
  - $Q(t,a) \leftarrow Q(t,a) + \alpha[r + \gamma \max_{a' \in \mathcal{A}} Q(t',a') - Q(t,a)]$

Jusqu'à (s terminal)

A propos de modèles

Un *modèle* est une représentation d'un MDP

- Il permet de prédire les évolutions de l'environnement

Estimer un modèle à partir d'épisode : apprentissage supervisé

- Apprendre $s, a \to r$ est un problème de ***régression***
- Apprendre $s, a \to s'$ est un problème d'***estimation de densité***

Un modèle permet de simuler l'environnement

- Ouvre la voie vers les méthodes de recherche basées sur la simulation

Extensions

- Dilemme exploration-exploitation
  - Algorithmes Bandit
  - Optimisme face à l'incertain
- Représentations d'état factorielles ou relationnelles
- Hiérarchies d'états ou d'actions
  - Options
  - Algorithmes : MAXQ, ...
- Observabilité partielle (POMDP, ...)
- Espaces d'états et/ou d'actions continus
- Actions duratives
- Décision en temps-réel