Agent Oriented Learning

Introduction apprentissage par renforcement: Q-learning & Sarsa

Stéphane Airiau

Université Paris-Dauphine

Temporal-difference Learning

Combine des idées de
la programmation dynamique (DP)
avec
les méthodes de Monte Carlo (MC)

Méthodes "Temporal-difference"

- elles apprennent directement avec l'expérience (comme MC)
- elles sont sans modèle : pas besoin de connaître les modèles de transition ou de récompenses (comme MC)
- elles peuvent apprendre d'épisodes incomplets (comme PD)
- elles utilisent des estimations pour mettre à jour son estimation (comme DP)

Méthodes "Temporal-difference"pour la fonction de valeurs

On veut estimer la valeur des états pour une politique fixe π.

- Méthode Monte Carlo "chaque visite"
  - mise à jour à l'aide du véritable gain G_t

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

  - G_t est accessible à la fin de l'épisode ⇨ peut on éviter cette atttente ?
- Méthode la plus simple : TD(0)

$$v(s_t) \leftarrow v(s_t) + \alpha\left[r_{t+1} + \gamma v(s_{t+1}) - v(s_t)\right]$$

mise à jour à l'aide de $r_{t+1} + \gamma v(s_{t+1})$

$$\begin{aligned} \text{Rappel : } v_\pi(s) &= \mathbb{E}_\pi\left[G_t \mid s_t = s\right] \\ &= \mathbb{E}_\pi\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s\right] \\ &= \mathbb{E}_\pi\left[r_{t+1} + \gamma v_\pi(s_{t+1}) \mid s_t = s\right] \end{aligned}$$

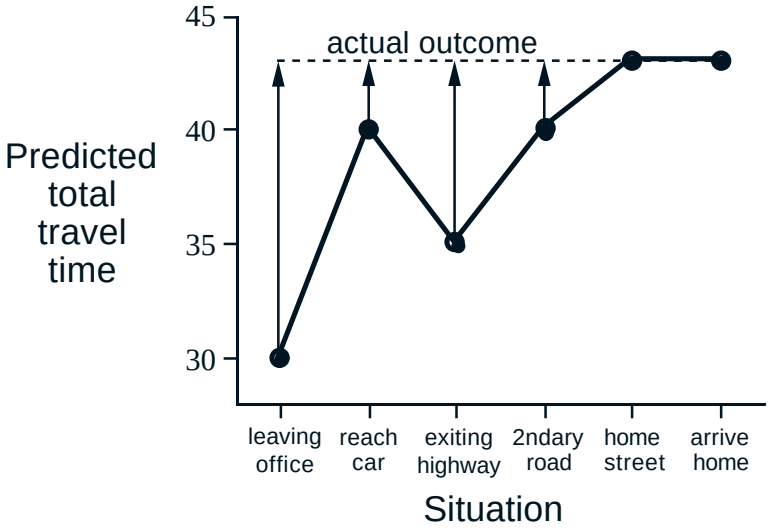
exemple du temps de trajet

Etat	temps écoulé	temps estimé	temps total prédit
départ du bureau	0	30	30
arrivée à la voiture, il pleut	5	35	40
sortie de l'autoroute	20	15	35
camion lent	30	10	40
arrivée dans le quartier	40	3	43
arrivée à la maison	43	0	43

exemple du temps de trajet

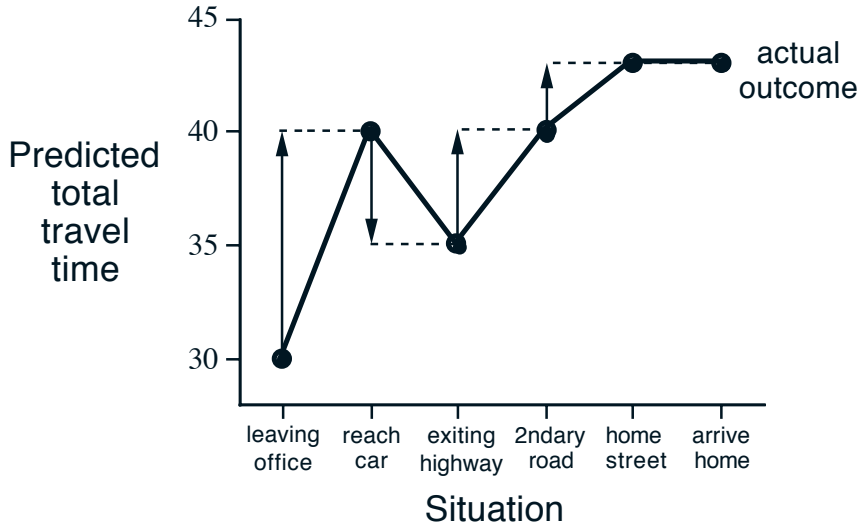
update avec méthode de Monte Carlo

$\alpha = 1$



update avec TD(0)

$\alpha = 1$



Avantages des méthodes TD

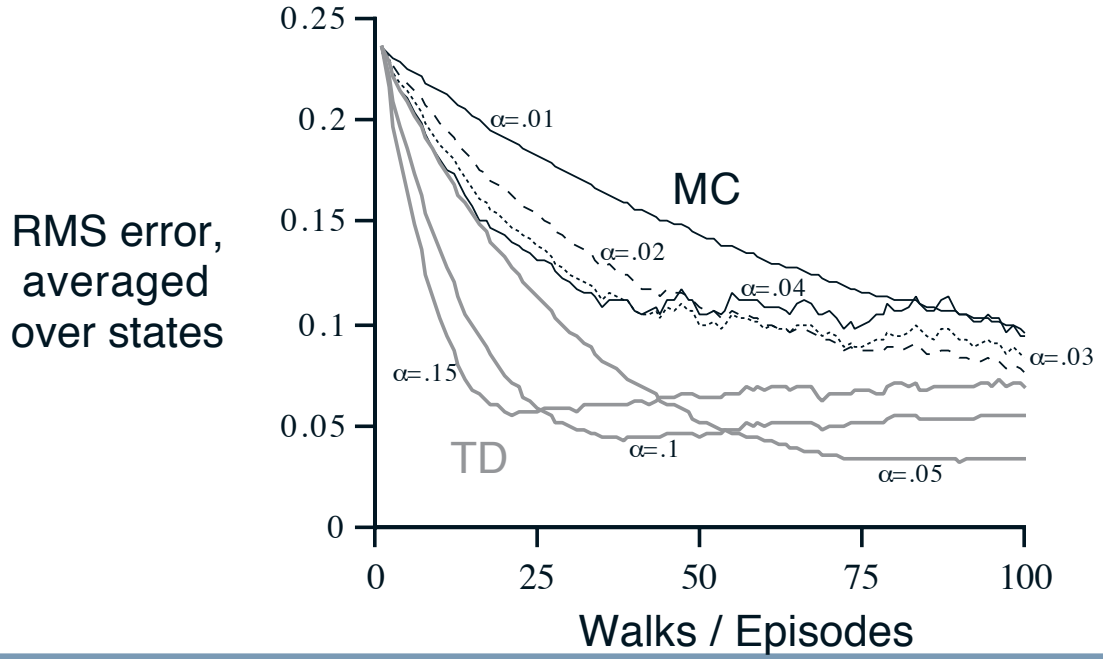
- pas besoin de connaissances des modèles
- méthode adaptée pour une utilisation online, et pas besoin d'attendre la fin de l'épisode
  les méthodes TD font une mise à jour après chaque itération
- il y a des garanties théoriques de convergence

Avantages des méthodes TD

- Pas de résultats théoriques comparant les performances des méthodes TD aux méthodes Monte Carlo.

⇨ en pratique, les méthodes TD sont plus rapides sur des problèmes stochastiques.

équiprobabilité d'aller à gauche ou à droite.



Exemple intuitif

On a un PDM a deux états A et B. Supposons qu'on observe les huit épisodes suivants :

$A,0,B,0$	$B,1$
$B,1$	$B,1$
$B,1$	$B,1$
$B,1$	$B,0$

Quel est votre évaluation pour $v(A)$ et $v(B)$?

Evaluation de la politique optimale : TD "on policy"

- On apprend la fonction de valeur des actions.
- Comme pour TD(0) pour la fonction de valeurs, on a :

$$q(s_t,a_t) \leftarrow q(s_t,a_t) + \alpha[r_{t+1} + \gamma q(s_{t+1},a_{t+1}) - q(s_t,a_t)]$$

State–action–reward–state–action (SARSA)

Initialise $q(s) \in \Delta(A)$ arbitrairement
Répète (éternellement) pour chaque épisode
    aller à l'état initial s
    choisir action $a \in A$ pour s à l'aide d'une politique dérivée de q (ex : ϵ-greedy)
    Répète pour chaque étape de l'épisode
        Exécute action a, observe $r \in \mathbb{R}$ et état suivant $s' \in S$
        choisir action $a' \in A$ pour s' à l'aide d'une politique dérivée de q
        $q(s,a) \leftarrow q(s,a) + \alpha[r + \gamma q(s',a') - q(s,a)]$
        $s \leftarrow s'$
        $a \leftarrow a'$
    jusqu'à ce que s soit terminal

Evaluation de la politique optimale : TD "on policy"

Theorème

L'algorithme converge vers la fonction optimale de valeur des actions sous les conditions suivantes :

- Glouton à la Limite avec Exploration Infinie
  - toutes les paires `(état,action)` sont explorées infiniement souvent $\lim_{k\to\infty} n_k(s,a) = \infty$
  - la politique converge vers une politique gloutonne $\lim_{k\to\infty} \pi_k(a|s) = 1$ pour $a = arg\max_{a' \in A} q(s,a')$
- $\sum_{t=1}^{\infty} \alpha_t = \infty$
- $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$

ϵ-greedy est GLEI si ϵ est une fonction décroissante (ex $e_k = \frac{1}{k}$)

Evaluation de la politique optimale : TD "off policy"

Q-learning (Watkins 1989)

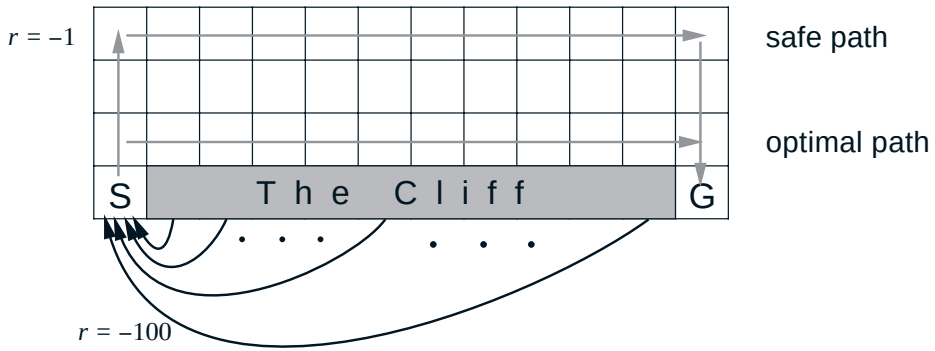
$$q(s_t,a_t) \leftarrow q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_{a\in A} q(s_{t+1},a) - q(s_t,a_t)\right]$$

Apprend une estimation de $q^\star$ de façon indépendante à la politique suivie.

1. Initialise $q(s) \in \Delta(A)$ arbitrairement
2. Répète (éternellement) pour chaque épisode
3. aller à l'état initial s
4. choisir action $a \in A$ pour s à l'aide d'une politique dérivée de q (ex : ϵ-greedy)
5. Répète pour chaque étape de l'épisode
6. Exécute action a, observe $r \in \mathbb{R}$ et état suivant $s' \in S$
7. choisir action $a' \in A$ pour s' à l'aide d'une politique dérivée de q
8. $q(s,a) \leftarrow q(s,a) + \alpha[r + \gamma \max_{a"\in A} q(s',a") - q(s,a)]$
9. $s \leftarrow s'$
10. $a \leftarrow a'$
11. jusqu'à ce que s soit terminal

Evaluation de la politique optimale : TD "off policy"

- Pour assurer la convergence, il faut s'assurer de visiter suffisemment souvent les paires `(action,état)`.
- sous les hypothèses GLIE, Q-learning converge



Autres méthodes d'exploration

- soft max : choisir l'action a avec probabilité

$$\frac{e^{\frac{q_t(s,a)}{\tau}}}{\sum_{a' \in A} e^{\frac{q_t(s,a')}{\tau}}}$$

  - $\tau > 0$ est appelée la température
  - température haute ⇨ probabilité uniforme
  - température basse ⇨approche le comportement glouton
- initialisation optimiste : initialiser les valeurs de manière optimiste puis être glouton (Even-Dar & Mansour, NIPS 1994)
  ⇨ force l'exploration à regarder les états qui semblent prometteur

Dilemme central : explorer ou exploiter

- contrairement au cas supervisé, les données sur lesquelles on travaille dépendent du comportement de l'agent !
- exploration : le but est d'apprendre le mieux possible
- exploitaiton : le but est d'optimiser au mieux ses récompenses
- défi de l'exploration : quelles actions vont améliorer au plus vite la connaissance de l'agent pour obtenir de meilleures récompenses.

⇨ l'exploration est un trait d'intelligence

- quelle politique doit suivre l'agent pour ne pas manquer les états qui donnent les bonnes récompenses (et sans passer trop de temps dans les états qui donnent de mauvaises récompenses)
- exploitation : préfère des actions qui ont mené a de "bons états"
- exploration : prendre une action qui pourrait nous mener à de bons états.

Stratégie d'exploration

- ϵ-greedy
  - facile à implémenter et très utilisée
  - convergence garantie (avec un taux d'exploration qui décroit de bonne manière)
  - il faut un nombre exponentiel d'échantillons pour garantir convergence
- Boltzmann
  - même problème pour le nombre d'échantillons

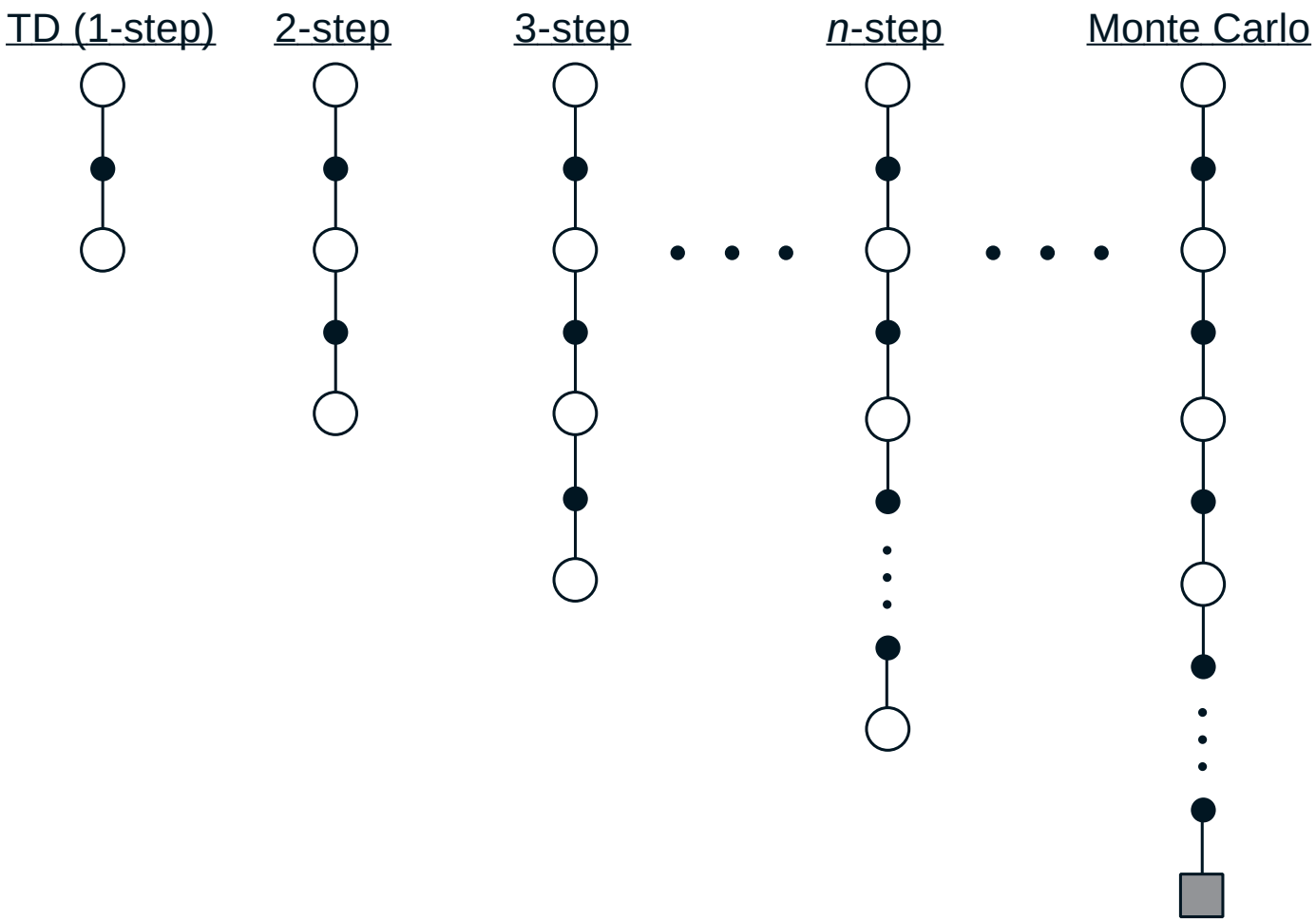
Explicit-Exploit-or-Explore E^3 (Kearns & singh 2002)

- Construit un modèle de PDM
  - optimiste
  - connait la récompense maximales
- compte le nombre de fois où chaque paire `(action,état)` a été visité pour estimer la qualité du modèle
  - ⇨ on connait l'état si on l'a visité un certain nombre de fois
  - Garantie statistique pour définir le nombre "suffisant" de viste
    $O((NTG^T_{max}/\epsilon)^4 \mathrm{Var}_{max} log(\frac{1}{\delta}))$ ou Var_{max} est la variance maximale des récompenses sur tous les états
- E^3 gère deux modèles : le modèle avec les états connus, et celui avec les états par encore connu.
  connu ⇨ exploitation
  pas encore connu ⇨ exploration

R_{max} (Brafman & Tennenholtz 2002)

- utilise la même idée, mais avec un seul modèle
- initialement, tout est inconnu, on estime avec une valeur maximale R_{max} toutes les récompenses, le modèle de transition est estimé déterministe
- on compte aussi les transitions observées pour déterminer ensuite lesquelles sont connues
- on calcule une politique optimale pour ce qu'on connait jusqu'ici
  - si l'état était connu ⇨ exploitation
  - si l'état n'était pas connu ⇨ on explore l'état le plus prometteur

Entre TD(0) et Monte Carlo : TD(n)



baser la mise à jour après quelques récompenses
(entre 1 et la fin de l'épisode)

On peut donc bâtir des algorithmes plus compliqués

mais c'est assez pour le moment !

Résumé

Types d'algorithmes pour

- évaluer une politique donnée
- trouver une politique optimale

Ce qu'on apprend

- $v_\pi : S \to \mathbb{R}$ la fonction de valeurs qui donne la valeur **à long terme** de chaque état en suivant une politique
- $\pi : A \times S \to \mathbb{R}$ la fonction qui donne la valeur **à long terme** de prendre une action puis de suivre une politique π depuis un état.

Algorithmes :

- modèle de transition et de récompenses connus ⇨ policy/value iteration
- Algorithmes pour domaines épisodique ⇨ méthodes de Monte Carlo
- Algorithme qui fonctione sans connaître ni bâtir un modèle Attention au dilemme Exploration Vs Exploitation ⇨ SARSA , Q-learning
- Algorithme qui fonctionne en bâtissant un modèle ⇨ E^3, R_{max}
- on policy : on améliore la politique que l'on suit / off policy : on utilise une stratégie pour apprendre une autre

Quelques limitations

- backgammon : $\approx 10^{20}$ états
- les échecs : $\approx 10^{50}$ états
- Go $\approx 10^{170}$ états, 400 actions
- Robotique : beaucoup de degrés de liberté

avec un stockage dans des tables, on ne peut pas résoudre ces problèmes !

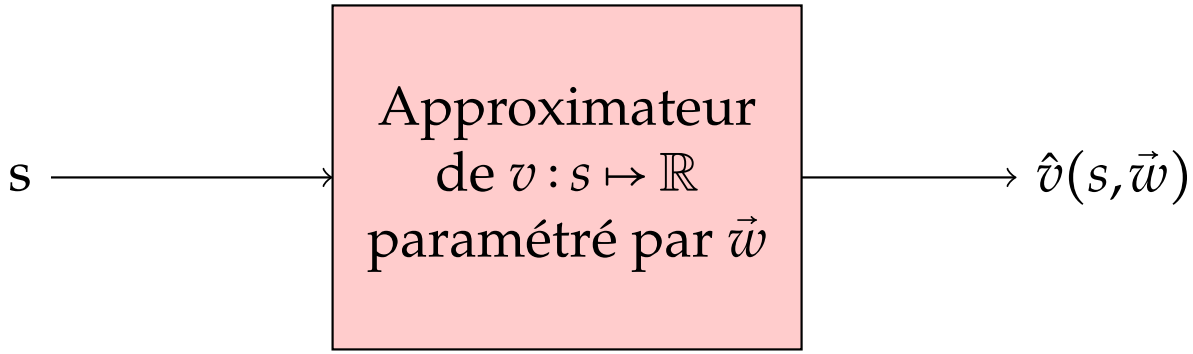
⇨ comment gérer des problèmes avec un grand nombre d'états et/ou d'action au niveau de la mémoire

⇨ comment va-t-on apprendre assez vite et généraliser sur des états/actions ?

Approximer la fonction de valeurs

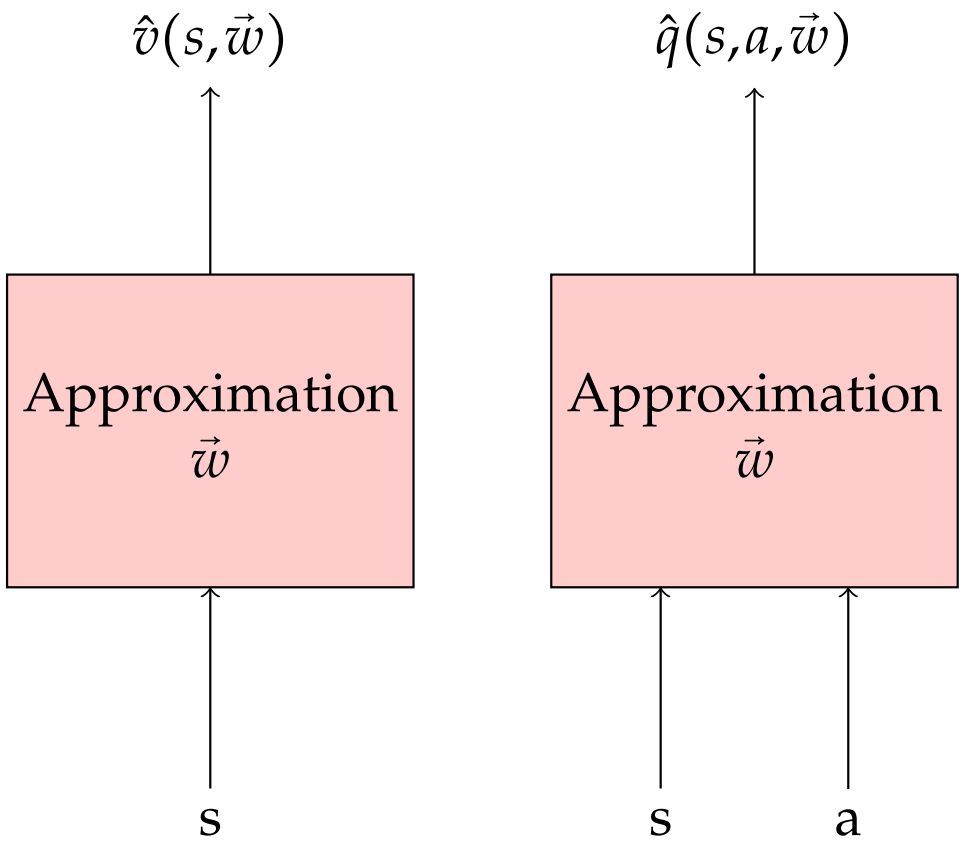
RL : on réalise une action a dans un état $s \in S$: on observe l'état suivant s' et on obtient la récompense r.

On peut voir RL + approximation comme un problème d'apprentissage supervisé : on observe s,a,s',r et on veut une fonction qui pour s donne une valeur à long terme u : on apprend la fonction $s \mapsto u$.



- Avec les méthodes tabulaires : on met à jour la valeur de s, et on ne change pas les valeurs pour les autres états.
- Avec l'approximation, on met à jour l'approximation
  ⇨ peut changer la valeur des autres états !

Type d'approximation de fonctions de valeurs



Quelle technique utiliser ?

- combinaison linéaire d'attributs
- réseau de neurones
- arbre de décision
- transformée de Fourier
- …

On va considérer les fonctions d'approximations qui sont différentiables.

Attention !

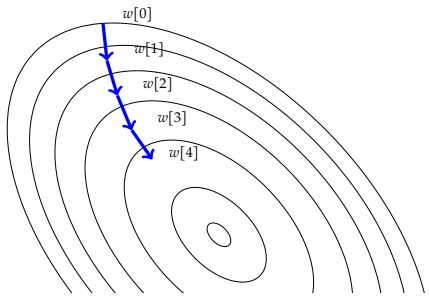
- nos données peuvent être non-stationnaires
  (i.e. elles peuvent dépendre du temps)
- elles **ne** sont **pas** i.i.d !

Descente de gradient

- Soit $J(\vec{w})$ une fonction différentiable par rapport au paramètre $\vec{w}$.
- Le gradient $\nabla J(\vec{w}) = \begin{pmatrix} \frac{\partial J(\vec{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\vec{w})}{\partial w_n} \end{pmatrix}$
- Pour trouver un minimum local de J,
  on ajuste $\vec{w}$ dans la direction opposée au gradient

$$\Delta\vec{w} = -\frac{1}{2}\alpha\nabla J(\vec{w}),$$

où α est un paramètre qui mesure la taille du pas de la mise à jour.



Descente stochastique de Gradient

- $\vec{w} \in \mathbb{R}^d$ vecteur de poids
- $\hat{v}$ est notre approximation
- $\hat{v}(s,\vec{w})$ approxime l'état s et est differentiable par rapport à $\vec{w}$ pour tout $s \in S$.

⇨ on va mettre à jour $\vec{w}$ pour améliorer l'approximation à chaque étape.

- But : minimiser l'erreur moyenne

$$J(\vec{w}) = \mathbb{E}_\pi\left[(v_\pi(s) - \hat{v}(s,\vec{w}))^2\right]$$

- Le gradient est donc

$$\begin{aligned}\Delta\vec{w} &= -\frac{1}{2}\alpha\nabla J(\vec{w}) \\ &= \alpha\mathbb{E}_\pi\left[(v_\pi(s) - \hat{v}(s,\vec{w}))\nabla\hat{v}(S,\vec{w})\right]\end{aligned}$$

Gradient stochastique

Supposons qu'on ait accès à des exemples $(s, v_\pi(s))$. Une bonne stratégie est d'effectuer une descente de gradient : ajuster les poids dans la direction qui réduit le plus possible l'erreur sur ces exemples.

$$\begin{aligned} \vec{w}_{t+1} &= \vec{w}_t - \frac{1}{2}\nabla\left[v_\pi(s) - \hat{v}(s_t, \vec{w}_t)\right]^2 \\ &= \vec{w}_t + \alpha\left[v_\pi(s) - \hat{v}(s_t, \vec{w}_t)\right]\nabla\hat{v}(s_t, \vec{w}_t) \end{aligned}$$

où $\nabla f(\vec{w}) = \left(\frac{\partial f(\vec{w})}{\partial w_1}, \frac{\partial f(\vec{w})}{\partial w_2}, \ldots, \frac{\partial f(\vec{w})}{\partial w_d}\right)^\top$ est le gradient de f par rapport à $\vec{w}$.

On dit que le gradient est stochastique quand on fait une mise à jour à chaque example.

Attention : on pourrait chercher à minimiser l'erreur sur les exemples que l'on a observé, mais il faut trouver une bonne approximation pour **tout** l'espace !

Gradient stochastique

On ne connait pas $v_\pi(s)$. Par contre, on peut utiliser une de nos approximations via

- Monte Carlo : $u_t = G_t$
- $TD(0)$: $u_t = r_{t+1} + \gamma \hat{v}(s_{t+1}, \vec{w}_t)$
- programmation dynamique : $u_t = \mathbb{E}\left[r_{t+1} + \gamma \hat{v}(s_{t+1}, \vec{w}_t) | S_t = s\right]$

On aura alors :

$$\vec{w}_{t+1} = \vec{w}_t + \alpha \left[u_t - \hat{v}(s_t, \vec{w}_t)\right] \nabla \hat{v}(s_t, \vec{w}_t)$$

Si l'approximation u_t est non-biaisée, alors on a la garantie de convergence vers un minimum local avec des conditions classiques pour un α qui diminue.

C'est le cas pour Monte Carlo !

Gradient stochastique

Pour TD(0) et la programmation dynamique, l'estimation dépend de la valeur de $\vec{w}$, donc on perd le caractère non biaisé : le passage entre les deux expressions n'est pas valide !

$$\begin{aligned}\vec{w}_{t+1} &= \vec{w}_t - \frac{1}{2}\nabla\left[v_\pi(s) - \hat{v}(s_t,\vec{w}_t)\right]^2 \\ &= \vec{w}_t + \alpha\left[v_\pi(s) - \hat{v}(s_t,\vec{w}_t)\right]\nabla\hat{v}(s_t,\vec{w}_t)\end{aligned}$$

Mais on peut utiliser l'expression, ce ne sera pas le vrai gradient, mais cela permettra parfois convergence dans certains cas intéressants.

- on prend bien en compte l'effet du changement de $\vec{w}$ sur l'estimation de $\hat{v}$
- mais pas son effet sur la cible $u_t = r_{t+1} + \gamma\hat{v}(s_t,\vec{w}_t)$

⇨ on parle de "semi gradient"

Semi-gradient TD(0) pour estimer une politique π donnée

```
1  pour chaque épisode
2      s ← S_initial on part d'un état initial
3      tant que l'état s n'est pas terminal
4          choisir une action a à l'aide de π(s)
5          exécute l'action a, oberve l'état suivant s' et la récompense r
6          w⃗ ← w⃗ + α[r + γv̂(s', w⃗) − v̂(s', w⃗)] ∇v̂(s, w⃗)
7          s ← s'
```

On a donc "juste" besoin d'utiliser une fonction d'approximation dont on sait calculer le gradient.

⇨ on va donc regarder avec plus de détails les méthodes linéaires.

Cas particulier 1 : Méthodes linéaires

$\hat{v}(\cdot, \vec{w})$ est une fonction linéaire du vecteur de poids $\vec{w}$:

à chaque état s correspond un vecteur $x(s) = (x_1(s), \ldots, x_d(s))^\top$ $x(s)$ qui représente l'état s, où chaque x_i représente un "attribut" de l'état s. Par exemple

- la distance d'un robot par rapport à certaines cibles
- la présence de certaines configurations de pièces sur un échiquier

Méthodes linéaires

$$\hat{v}(s,\vec{w}) = \sum_{i=1}^{d} w_i x_i(s) = \vec{w}^\top \cdot \vec{x}(s).$$

Avec les méthodes linéaires, on veut que les attributs soient les bases de l'espace des états.

A cause de la linéarité, on obtient donc :

$$\nabla \hat{v}(s,\vec{w}) = x(s)$$

et ainsi :

$$\begin{aligned} \vec{w}_{t+1} &= \vec{w}_t + \alpha\left[u_t - \hat{v}(s_t,\vec{w}_t)\right]x(s_t) \\ &= \vec{w}_t + \alpha\left[u_t - \vec{w}_t^\top x(s_t)\right]x(s_t) \end{aligned}$$

Méthodes linéaires

- Pour le cas linéaire, l'optimum est unique et toutes les méthodes vont converger !
- Le gradient de Monte Carlo va converger (avec hypothèses sur la fonction α qui décroit au cours du temps) vers l'optimum.
- Le semi gradient $TD(0)$ converge, mais vers un point proche de l'optimal.

Méthodes linéaires : constructions des bases

On peut utiliser des techniques classiques pour construire les bases :

- bases de polynomes
- séries de Fourier
- "codage grossier"
- codage en mosaïque
- fonctions de base radiale

Optimisation avec approximation de fonction

On vient de voir comment on peut évaluer une politique donnée.

On cherche cependant à trouver une politique optimale.

On utilise une nouvelle fois notre stratégie classique

- évaluation de la politique : approximation $\hat{q}(\cdot,\cdot,\vec{w})$
- amélioration de la politique : par exemple avec ϵ-glouton.

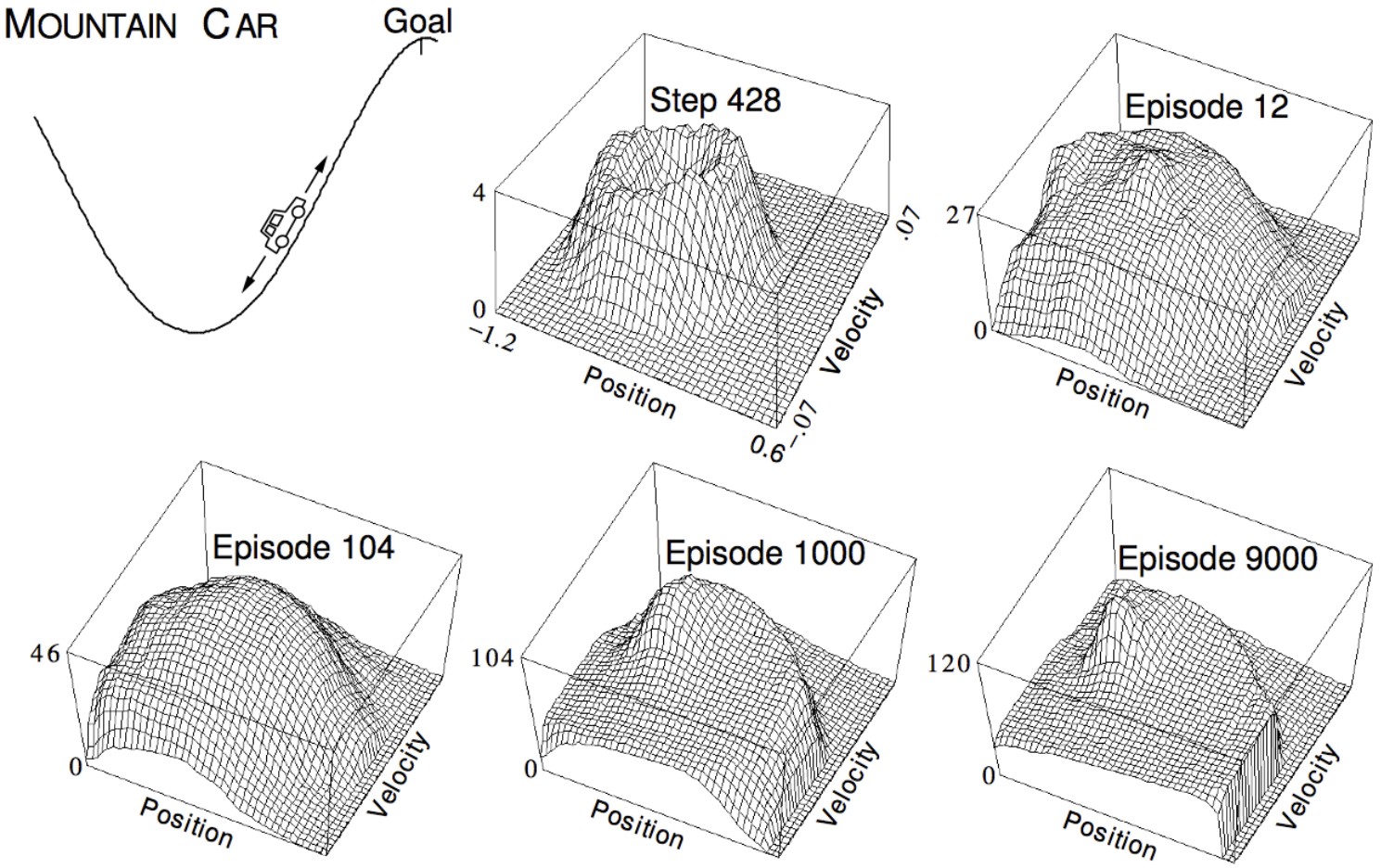
⇨ pas forcémment besoin d'utiliser trop d'échantillons pour trouver la bonne approximation $\hat{q}$

SARSA semi gradient pour estimer $q^\star$

State–action–reward–state–action (SARSA)

Initialise $\hat{q} : S \times A \times \mathbb{R}^d \to \mathbb{R}$ arbitrairement
Répète (éternellement) pour chaque épisode
    aller à un état initial s
    choisir action $a \in A$ pour s à l'aide d'une politique dérivée de $\hat{q}(s,\cdot,\vec{w})$ (ex : ϵ-greedy
    Répète pour chaque étape de l'épisode
        Exécute action a, observe $r \in \mathbb{R}$ et état suivant $s' \in S$
        choisir action $a' \in A$ pour s' à l'aide d'une politique dérivée de $\hat{q}(s',\cdot,\vec{w})$
        $\vec{w} \leftarrow \vec{w} + \alpha[r + \gamma\hat{q}(s',a',\vec{w}) - \hat{q}(s,a,\vec{w})]\nabla\hat{q}(s,a,\vec{w})$
        $s \leftarrow s'$
        $a \leftarrow a'$
    jusqu'à ce que s soit terminal
    $\vec{w} \leftarrow \vec{w} + \alpha[r - \hat{q}(s,a,\vec{w})]\nabla\hat{q}(s,a,\vec{w})$

Exemple : SARSA linéaire (coarse coding)



Résultats de Convergence pour l'évaluation de politique

Pour Monte Carlo, l'estimation est non biaisée et on a des garanties de convergence.

Pour les méthodes TD, on peut construire des exemples dans lesquels les poids divergent ! Cependant en pratique, cela marche très souvent.

- TD ne suit pas le gradient d'une fonction objective !
- TD peut donc diverger avec des méthodes off policy ou en utilisant des approximations de fonctions non linéaires.
- certains nouveaux algorithmes arrivent à corriger le gradient pour assurer la convergence

On/Off Policy	Algorithme	Tabulaire	Linéaire	non-linéaire
On-Policy	Monte Carlo	✔	✔	✔
	TD(0)	✔	✔	✘
Off-Policy	Monte Carlo	✔	✔	✔
	TD(0)	✔	✘	✘

Résultats de Convergence pour l'optimisation

Algorithme	Tabulaire	Linéaire	non-linéaire
Monte Carlo	✔	(✔)	✘
Sarsa	✔	(✔)	✘
Q-learning	✔	✘	✘

(✔) on peut osciller autour de la solution optimale