Chapitre 6

Apprentissage par renforcement

6.1 Motivation

D'une façon générale, nous savons que les récompenses ou les punitions ("renforceurs") permettent de conditionner les actions d'un animal. Plus précisément, lorsque ces renforceurs sont associés aux actions de l'animal, ce dernier adaptera son comportement de façon à augmenter ses récompenses et à diminuer ses punitions.

La même idée peut aussi s'appliquer pour les jeux ou pour l'optimisation des commandes d'un agent robotique. Pour fixer les idées, prenons le jeu du Backgammon. Dans ce cas, la récompense est attribuée au vainqueur uniquement à la fin de la partie. La question à laquelle nous désirons répondre dans ce chapitre peut être formulée de la façon suivante: étant donné un état donné du jeu, quelle doit être l'action choisie par le joueur de façon à maximiser ses chances de victoire?

Il est à noter que, dans ce cas, le choix d'une action ne conduit pas le joueur de façon déterministe dans un nouvel état, puisque l'adversaire est également libre de choisir une action. Les dés ajoutent une autre source de stochasticité.

Pour une description plus précise du problème, considérons les éléments suivants:

- Un agent, pouvant effectuer les actions a_1, a_2, a_3, etc.
- Un environnement, dans lequel se trouve l'agent. L'agent peut être dans les états s_1, s_2, s_3, etc.
- Une récompense qui, au temps t, vaut r_t, et qui dépend de a et de s.

Typiquement, on aura $r_t = R^a_{s \to s'}$ c.a.d., la récompense reçue pour une action a qui a mené à la transition de l'état s à l'état s'; cf. Fig. 6.1

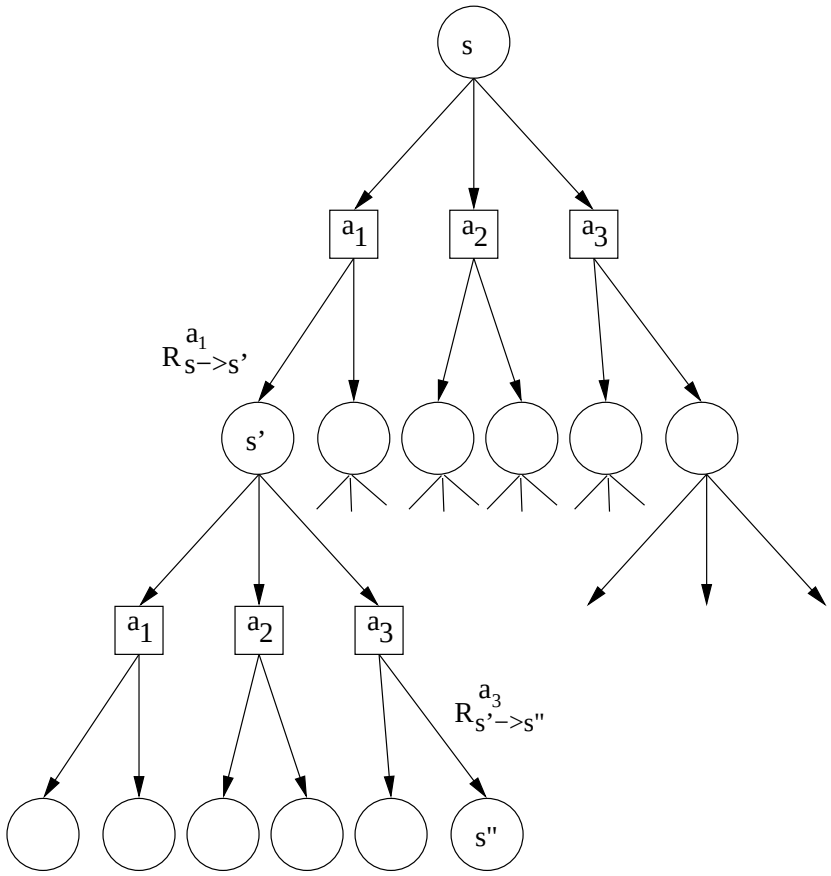

FIG. 6.1 – *Transition entre états s avec des actions a*

Notre but est de maximiser l'espérance de la somme des récompenses futures $r_t + r_{t+1} + r_{t+2} + \ldots$. En principe cette somme pourrait diverger. Pour éviter les problèmes de divergences, on ajoute un facteur d'amortissement γ^k pour une récompense k pas dans le futur. On veut donc optimiser l'espérance de la somme

$$E(r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \ldots) = E(\sum_{k=0}^{\infty} \gamma^k\, r_{t+k})\,. \tag{6.1}$$

Avec $\gamma < 1$ et une récompense maximale $r_t \leq r^{max}$, on garantit que la somme converge. Dans la pratique on choisit souvent $0.9 \leq \gamma \leq 1$.

Pour arriver à notre but d'optimisation de récompenses futures, on utilise une stratégie (en anglais, "*policy*"). Une stratégie revient à définir pour chaque état

s la meilleure action a. Plus généralement, quand on ne sait pas encore quelle est la meilleure action, on choisit dans l'état s une action a aléatoirement avec une probabilité $\Pi(s,a)$. Une distribution de probabilité $\Pi(s,a)$ définit la stratégie utilisée.

6.2 Evaluation, Exploitation, Exploration

Pour mieux comprendre les concepts introduits en haut, nous commençons par un cas simple. Nous avons un seul état, et faisons un choix d'action comme dans un jeu de hasard. Les transitions sont décrites par une loi probabiliste. Par exemple, si on choisit l'action a_3, on arrive avec la probabilité $P^{a_3}_{s\to s''}$ dans l'état s''.

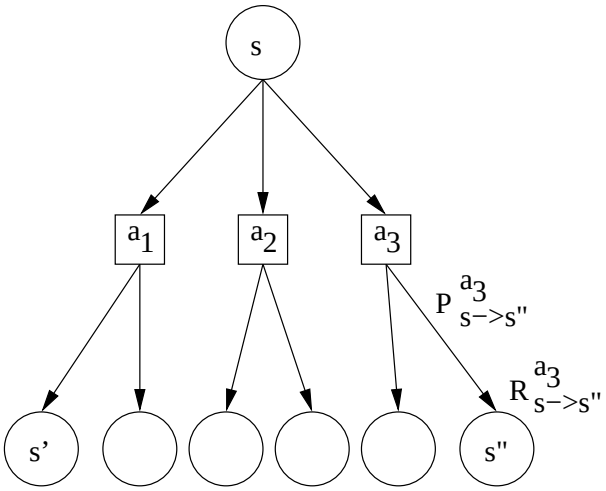

FIG. 6.2 – *Choix d'action dans un jeu de hasard.*

But. Optimiser E(récompense future).

Pour cela, nous attribuons des valeurs à chaque action: la valeur de l'action a_i se note $Q^*(a_i)$, et vaut E(récompense pour a_i). Avec les probabilités de transition on a

$$Q^*(a_i) = \sum_{s'} P^{a_i}_{s\to s'} \cdot R^{a_i}_{s\to s'} \,. \tag{6.2}$$

Clairement, la stratégie optimale est de choisir a^*, tel que

$$Q^*(a^*) = \max_a Q^*(a)$$

c'est-à-dire de prendre l'action qui retourne (en moyenne) la récompense optimale.

Le problème est que, d'habitude, les valeurs d'action Q ne sont pas connues. En effet, on sait que $Q^*(a_i) = \sum_{s'} P^{a_i}_{s\to s'} \cdot R^{a_i}_{s\to s'}$, mais on ne connait pas $P^{a_i}_{s\to s'}$.

La méthode que nous proposons ici donne une estimation de Q, sans passer par le calcul de $P^{a_i}_{s\to s'}$. Il suffit de jouer un grand nombre (disons K_a) de fois l'action a et de retenir à chaque fois la récompense obtenue, et on peut alors estimer Q ainsi:

$$Q^{K_a}(a) = \frac{r_1 + \ldots + r_{K_a-1} + r_{K_a}}{K_a} \tag{6.3}$$

Plutôt que de jouer K_a actions de suite et estimer la valeur Q seulement à la fin ("batch mode"), on peut aussi arriver à une estimation itérative ("on-line"), en mettant à jour la valeur $Q(a)$ chaque fois qu'on a joué l'action a

$$\Delta Q(a) = \eta\,[r_t - Q(a)] \tag{6.4}$$

avec un taux d'apprentissage η qui tend vers zero pour $K_a \to \infty$.

Démonstration de 6.4. La prochaine fois que je jouerai l'action a, la formule (6.3) se réécrira:

$$Q^{K_a+1}(a) = \frac{r_1 + \ldots + r_{K_a} + r_{K_a+1}}{K_a + 1},$$

ce qui peut s'écrire comme

$$Q^{K_a+1}(a) = \frac{K_a}{K_a+1} \underbrace{\frac{r_1 + \ldots + r_{K_a}}{K_a}}_{Q^{K_a}(a)} + \frac{r_{K_a+1}}{K_a+1}$$

Calculons maintenant $\Delta Q(a)$:

$$\Delta Q(a) = Q^{K_a+1}(a) - Q^{K_a}(a) = \left[\frac{K_a}{K_a+1} - 1\right] Q^{K_a}(a) + \frac{1}{K_a+1}\, r_{K_a+1}$$

$$\Delta Q(a) = \frac{K_a - (K_a+1)}{K_a+1}\, Q^{K_a}(a) + \frac{1}{K_a+1}\, r_{K_a+1}$$

$$\Delta Q(a) = \underbrace{\frac{1}{K_a+1}}_{\eta} \left[\underbrace{r_{K_a+1}}_{r_t} - \underbrace{Q^{K_a}(a)}_{Q(a)}\right]$$

$$\Delta Q(a) = \eta\,[r_t - Q(a)] \tag{6.5}$$

On trouve alors Eq. (6.4) avec $\eta = 1/(K_a+1)$.

Après convergence (c'est-à-dire après beaucoup de jeux) on a en moyenne

$$\langle \Delta Q(a) \rangle = 0$$

Ce qui, par 6.4, est équivalent à

$$Q(a) = \langle r_t \rangle = \sum_{s'} P^a_{s \to s'} \cdot R^a_{s \to s'}$$

Donc, $Q(a)$ converge vers la solution correcte $Q^*(a)$, Eq. (6.2). *q.e.d.*

Problème de la meilleure stratégie Si on connaît les valeurs correctes $Q^*(a)$ de chaque action a, il faut jouer a^* tel que $Q^*(a^*) = \max_a Q^*(a)$. Cette stratégie s'appelle *greedy*. Malheureusement, dans la pratique, une stratégie 'greedy' n'est pas toujours optimale.

La raison est que les 'vraies' valeurs Q^* ne sont jamais connues. Pour connaître les valeurs Q^*, il faudrait connaître les valeurs P, ce qui n'est pas notre cas. On a seulement des estimations approximatives qu'on notera Q (sans l'asterisque).

En fait, on peut estimer Q par itérations (cf ci-dessus), mais tant que les valeurs de Q ne sont pas bonnes, on ne sait pas quelle tactique appliquer. Or, les valeurs de Q mettent tout de même un certain temps à converger. On doit donc faire un choix entre deux stratégies non compatibles:

- Exploration: explorer toutes les possibilités pour estimer $Q(a)$.
- Exploitation: prendre la meilleure action connue pour gagner la meilleure récompense.

Solutions. Il existe plusieurs solutions heuristiques au problème ci-dessus. En voici trois.

- ε-greedy.
  Prendre avec une probabilité $1 - \varepsilon$ l'action a^* qui semble être la meilleure action se basant sur les estimations $Q(a)$ actuelles:

$$\Pi(s,a) = 1 - \varepsilon \qquad \text{si } a = a^*, \text{ avec } Q(a^*) = \max_a Q(a) \tag{6.6}$$

  Avec une probabilité ϵ on choisit (aléatoirement) une des autres actions.
- Stratégie optimiste.
  - Initialiser $Q(a)$ avec une valeur trop grande. (Il faut déjà avoir une idée de la valeur qui sera obtenue.)
  - Dès qu'on reçoit une récompense petite, $Q(a)$ diminue à cause de 6.4.
  - On prend l'action optimale a^* telle que $Q^*(a^*) = \max_a Q^*(a)$

  Au début, chaque action fait baisser $Q(a)$ de manière presque égale. Puis, les meilleures actions commencent à s'identifier.

– Soft Max: Choisir $\Pi(s,a)$ ainsi:

$$\Pi(s,a) = \frac{e^{\beta\, Q(s,a)}}{\sum_{a'} e^{\beta\, Q(s,a')}}$$

On note que si β tend vers l'infini, alors on prend l'action a^* avec une probabilité de 1.
Pour passer du régime d'exploration au régime d'exploitation, il suffit de faire passer β d'une valeur petite à une valeur grande. La stratégie consiste donc à poser $\beta = 0.1$ au début, puis à augmenter régulièrement β.

6.3 Valeurs des actions et équation de Bellmann

Retournons maintenant à la situation générale. Un agent part de l'état s en prenant une première action a_1. Arrivé au prochain état s'_1 il prends une deuxième action a' et ainsi de suite. La figure 6.3 représente les différents déroulements qu'une telle suite d'actions peut prendre, partant de l'état s.

Dans cette figure, Q représente l'espérance de recevoir une récompense. Nous cherchons à calculer Q. Par définition, Q est égal à l'espérance de la récompense dans tout le futur. Cela peut se formaliser ainsi:

$$Q(s,a) = E(R(s,a)) = E(\gamma^0\, r_{t+1} + \gamma^1\, r_{t+2} + \gamma^2\, r_{t+3} + \ldots)$$

$$Q(s,a) = E\left(\sum_{k=0}^{\infty} \gamma^k\, r_{t+1+k}\right)$$

$$Q(s,a) = E\left(\gamma^0\, r_{t+1} + \sum_{k=1}^{\infty} \gamma^k\, r_{t+1+k}\right)$$

Comme $\gamma^0 = 1$, on peut écrire:

$$Q(s,a) = E\left(r_{t+1} + \sum_{k=1}^{\infty} \gamma^k\, r_{t+1+k}\right)$$

r_{t+1} décrit la récompense immédiate qu'on reçoit, par exemple, pour la transition de s à s'. $\sum_{k=1}^{\infty} \gamma^k\, r_{t+1+k}$ décrit la somme des récompenses futures qu'on peut atteindre à partir de l'état s'. En exprimant ce terme par les valeurs $Q(s',a')$, on arrive à l'équation de Bellman, qui s'écrit:

$$Q^{\Pi}(s,a) = \sum_{s'} P^a_{s\to s'} \left[R^a_{s\to s'} + \gamma \sum_{a'} \Pi(s',a')\, Q^{\Pi}(s',a')\right] \tag{6.7}$$

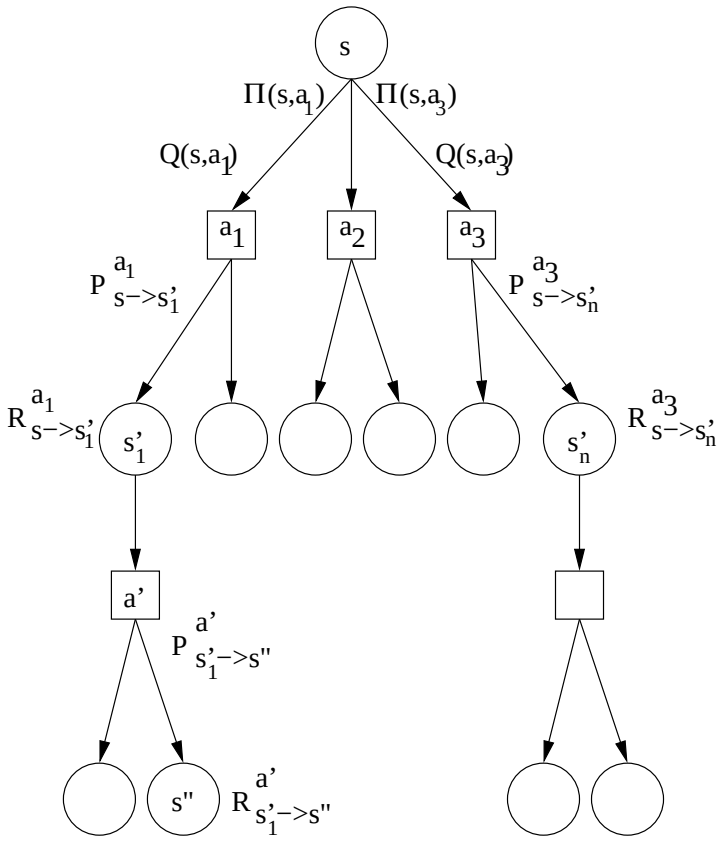

FIG. 6.3 – *Déroulements d'états et d'actions (arbre de jeu).*

La solution de l'équation (6.7) donne un ensemble de valeurs $Q^{\Pi}(s,a)$. Notons que cette solution dépend de notre choix de stratégie $\Pi(s,a)$ ce que nous indiquons par l'indice Π. Si on choisit la stratégie optimale ("greedy"),

$$\text{Prendre l'action } a^* \text{ avec } Q(s,a^*) = \max_a Q(s,a)\,, \tag{6.8}$$

l'équation de Bellman s'exprime ainsi:

$$Q^*(s,a) = \sum_{s'} P^a_{s\to s'} \left[R^a_{s\to s'} + \gamma \max_{a'} Q^*(s',a') \right] . \tag{6.9}$$

Ici, Q avec asterisque dénote la 'vraie' valeur des actions. La solution de (6.9) est équivalente à la programmation dynamique. Notons que (6.9) est une formulation au niveau des valeurs d'*actions*. Il est possible de trouver une autre forme equivalente en notation avec des valeurs d'*états*.

Remarque. On peut réécrire l'équation de Bellman dans une manière

différente si on remplace les valeurs Q par des *valeurs d'états*

$$V(s) = \sum_{a'} \Pi(s,a')\, Q(s,a')\,.$$

L'équation de Bellman s'écrit alors comme

$$V(s) = \sum_{a} \Pi(s,a) \sum_{s'} P^a_{s\to s'}\ [R^a_{s\to s'} + \gamma\, V(s')]\,. \tag{6.10}$$

Pour la stratégie optimale, on a $V^*(s) = \max_{a'} Q^*(s,a')$ et donc

$$V^*(s) = \max_a \sum_{s'} P^a_{s\to s'}\ [R^a_{s\to s'} + \gamma\, V^*(s')\,]\,. \tag{6.11}$$

La notation en valeurs d'état est plus habituelle que celle en valeurs d'actions, mais les deux sont équivalentes.

SARSA. Jusqu'ici, on a fait l'hypothèse que $P^a_{s\to s'}$ étaient connus. Dans ce cas-là, la solution de l'équation de Bellman nous donne des valeurs $Q(s,a)$ et le choix d'action devient simple.

En général, les probabilités de transition ne sont pas disponible. On applique donc le même principe que dans la section 6.2. Plus précisement, on essaie d'estimer Q d'une façon itérative en explorant l'espace des états et actions. L'algorithme de SARSA nous donne une solution à ce problème.

Si on résume l'équation de Bellman à son essentiel (en supprimant pour le moment les moyennes), on peut écrire qu'approximativement

$$Q(s,a) \approx r_{t+1} + \gamma\, Q(s',a')\,.$$

Qualitativement, la valeur $Q(s,a)$ doit s'expliquer par la récompense r_{t+1} dans le prochain pas (reçue en partant de l'état s avec action a) plus la valeur $Q(s',a')$ pour la nouvelle action a' dans le nouvel état s'. Nous pouvons alors poser l'équation d'erreur:

$$E = \frac{1}{2}\left\{Q(s,a) - [r_{t+1} + \gamma Q(s',a')]\right\}^2\,. \tag{6.12}$$

Dans cette équation, la variable à optimiser est $Q(s,a)$. Nous pouvons faire cela par une descente de gradient:

$$\Delta Q(s,a) = -\eta\, \frac{\partial E}{\partial Q(s,a)} \tag{6.13}$$

$$\Delta Q(s,a) = -\eta\, \{Q(s,a) - [r_{t+1} + \gamma Q(s',a')]\}$$

$$\Delta Q(s,a) = \eta\, \{r_{t+1} + \gamma\, \underbrace{Q(s',a')}_{\text{etat futur}} - \underbrace{Q(s,a)}_{\text{etat actuel}}\ \} \tag{6.14}$$

Remarque. L'equation (6.14) est une généralisation de (6.4). La seule différence est que le récompense dans (6.4) doit être remplacée par $r_{t+1} + \gamma\, Q(s'a')$. Une autre interprétation vient d'un regroupement différent des termes:

$$\Delta Q(s,a) = \eta\,\{\underbrace{r_{t+1}}_{r} - [\underbrace{Q(s,a) - \gamma Q(s',a')}_{\langle r\rangle}]\} \tag{6.15}$$

La mise-à-jour de Q est positive si la récompense reçue r est plus large que la récompense espérée $\langle r\rangle = Q(s,a) - \gamma Q(s',a')$.

L'algorithme de SARSA utilise la règle de mise-à-jour (6.14) et se décrit ainsi:

- initialiser $Q(s,a)$
- répéter
  1. commencer en **s**
  2. choisir action **a** selon la politique $\Pi(s,a)$ (Par exemple, ε-greedy)
  3. observer la récompense **r**
  4. observer le prochain état **s'**
  5. choisir l'action **a'**. Faire une mise-à-jour selon l'équation 6.14.
  6. poser $s := s'$ et $a := a'$. Puis, recommencer en 3

Remarque. SARSA est l'acronym pour la suite des évenement **s**tate – **a**ction – **r**écompense – **s**tate – **a**ction.

Théorème 6.1 *SARSA converge (pour $\eta \to 0$) vers une solution de l'équation de Bellman.*

On ne cherche pas ici à démontrer le théorème 6.1. Toutefois, nous aimerions donner au lecteur une idée intuitive de la démonstration:

$$\Delta Q(s,a) = \eta\ \{r_{t+1} + \gamma Q(s',a') - Q(s,a)\}$$

Si η est petit, le changement est lent. On peut alors moyenner sur beaucoup d'itérations:

$$< \Delta Q(s,a) >= \eta\ < r_{t+1} + \gamma Q(s',a') - Q(s,a) >$$

Après convergence, $\Delta Q = 0$ et $Q(s,a)$ atteint une valeur constante. Donc,

$$0 =< r_{t+1} + \gamma Q(s',a') - Q(s,a) >$$

$$0 =< r_{t+1} + \gamma Q(s',a') > -Q(s,a)\,.$$

Rappelons que r_{t+1} est la récompense immédiate qu'on reçoit en partant de l'état s avec l'action a. En mettant $Q(s,a)$ de l'autre côté et en évaluant l'espérance qui reste, on arrive à

$$Q(s,a) = \sum_{s'} P^a_{s\to s'} \left[R^a_{s\to s'} + \gamma \sum_{a'} \Pi(s',a')\, Q(s',a') \right].$$

Il ne nous reste plus qu'à comparer ce dernier résultat avec l'équation de Bellman (6.7)...

Remarque. On peut refaire le même raisonnement pour les valeurs d'états $V(s)$ que celui fait jusqu'ici pour les valeurs d'actions $Q(s,a)$. On peut alors dériver l'algorithme **Temporal Difference (TD) learning** d'une manière analogue à la dérivation pour l'algorithme SARSA. Si l'on part de l'équation (6.11) et suit les même pas de calcul qu'avant, on arrive à une règle itérative pour la mise à jour des valeurs d'etats:

$$\Delta V(s) = r_{t+1} + \gamma\, V(s') - V(s) \tag{6.16}$$

s est l'état actuel, s' l'état une itération (un pas de temps) plus tard. Si on choisit $\gamma = 1$, on voit que $V(s') - V(s)$ est la difference "temporelle" (d'un pas de temps à l'autre) de la valeur d'état. Le nom *Temporal Difference Learning* vient de cette interprétation.

6.4 Etats continus

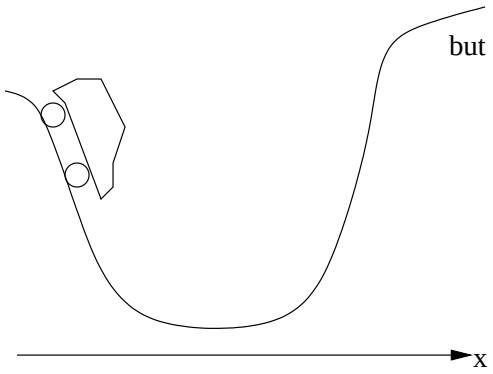

FIG. 6.4 – *Problème à état continu.*

Considérons le problème illustré par la figure 6.4. On aimerait arriver avec la voiture à la zone du but, en utilisant un moteur qui est trop faible pour monter

la pente. Si l'état initial de la voiture est en bas de la vallée, on doit donc d'abord reculer et prendre de la vitesse pour monter de l'autre côté. Nous cherchons à appliquer un apprentissage par renforcement pour optimiser une telle stratégie. La situation physique correspond clairement à un problème à états continus. Or, jusqu'ici nous avons seulement discuté des états discrets. Comment procéder?

Le premier pas consisterait à faire une discrétisation des états, comme illustré sur la figure 6.5. Chaque état est caracterisé par une combinaison de position et vitesse. Pour chaque état on doit choisir une action appropriée. La structure du probléme suggère alors d'établir une table similaire à la table 6.1.

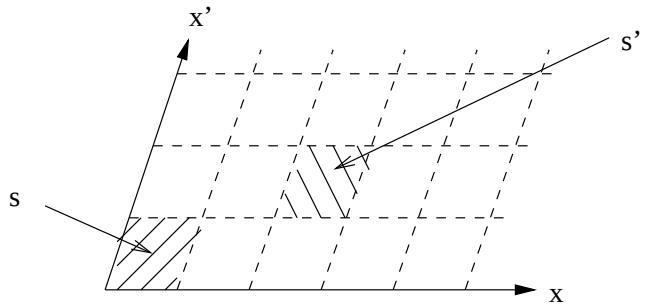

FIG. 6.5 – *Discrétisation des états d'un problème à états continus: Ici, on discrétise x et x'. Chaque case représente un état s.*

actions	a_1 (aller à gauche)	a_2 (rien)	a_3 (aller à droite
$s_{(x,x')}$			
$s'_{(x,x')}$	$Q(s',a_1)$		
$s''_{(x,x')}$		$Q(s'',a_2)$	
$\vdots$			

TAB. 6.1 – *Table des actions.*

Cette table changerait si on choisissait une discrétisation différente. Or, le problème physique ne devrait pas dépendre de notre choix de discrétisation. Alors, on peut poser la question si on peut faire mieux que cela.

La vraie situation correspond à un problème avec des états continus. Si on se focalise sur une action spécifique, disons a_1, on peut considèrer $Q(s,a_1)$ comme une fonction de l'état s ce qui est possible même si s est une variable continue. Si on fait l'hypothèse que Q est une fonction continue de l'état s, on peut par exemple modéliser $Q(s,a_1)$ avec un réseau de neurones, comme sur la figure 6.6. Notons que $\phi_k(s)$ est une fonction à base radiale qui représente l'activité dans l'état s: Plus l'état sera proche du centre de ϕ_k , plus la valeur de $\phi_k(s)$ sera

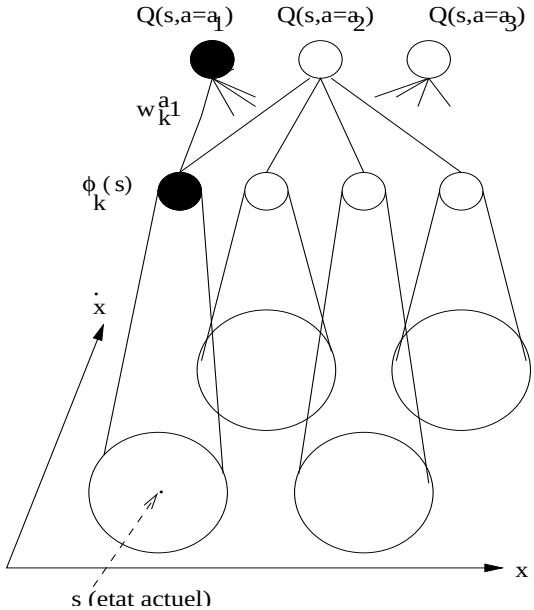

FIG. 6.6 – *Modélisation de $Q(s,a_1)$ par un réseau de neurones.*

forte. Explicitement:

$$Q(s,a) = \sum_k w_k^a \cdot \phi_k(s)\,. \tag{6.17}$$

Pour chaque action a on construit un réseau de neurones avec des poids w_k^a.

L'estimation des valeurs Q revient donc à une optimisation des poids w_k^a. pour la mise-à-jour des w_k^a on utilise la fonction d'erreur

$$E = \frac{1}{2}\left\{Q(s,a) - [r + Q(s',a')]\right\}^2 \tag{6.18}$$

qui est identique à l'équation (6.12) utilisée précédemment pour SARSA. Avec une descente de gradient, on trouve la mise-à-jour des poids:

$$\Delta w_i^a = -\eta\,\frac{\partial E}{\partial w_i^a}$$

$$\Delta w_i^a = -\eta\,\left\{Q(s,a) - [r + Q(s',a')]\right\}\frac{\partial Q(s,a)}{\partial w_i^a}$$

Comme $\frac{\partial Q(s,a)}{\partial w_i^a} = \Phi_i(s)$ à cause de (6.17), on a:

$$\Delta w_i^a = -\eta\,\left\{Q(s,a) - [r + Q(s',a')]\right\}\Phi_i(s) \tag{6.19}$$

$$\Delta w_i^a = \eta\,\left\{r + Q(s',a') - Q(s,a)\right\}\Phi_i(s) \tag{6.20}$$

Ainsi, si on a choisi l'action a_1 au temps t, le changement des poids est:

$$\begin{cases} \Delta w_i^{a_1} = \eta \, \{r + Q(s',a') - Q(s,a_1)\} \, \Phi_i(s) \\ \Delta w_i^{a} = 0 \qquad \text{pour } a \neq a_1 \end{cases} \tag{6.21}$$

Puis, on peut appliquer l'algorithme de SARSA avec:

$$\Delta w_i^a = \eta \underbrace{\{r + Q(s',a') - Q(s,a)\}}_{\rho} \Phi_i(s) \cdot \delta_{a,a_1} \tag{6.22}$$

Où ρ est la récompense actuelle opérée ($r - \langle r \rangle$ ou $r - [Q(s,a) - Q(s',a')]$), $\Phi_i(s)$ est l'activité de i et δ_{a,a_1} vaut 1 si $a = a_1$, 0 sinon.

Remarques.
(i) La règle (6.22) a une interprétation biologique intuitive [DA01]. Une connexion entre un neurone d'état (neurone pré-synaptique) et un neurone d'action (neurone post-synpatique) est renforcée si les deux neurones connectés sont actifs simultanément, (c.a.d. l'action a_1 est choisie dans l'etat s) et si cette action a eu du succès. Le succès ρ est mesuré comme la différence entre la récompense reçue r et la récompense espérée $\langle r \rangle = Q(s,a) - Q(s',a')$. Un tel signal de succès est disponible dans le cerveau sous forme de dopamine, qui est diffusée en cas de success dans presque tout le cortex. [1]
(ii) Si l'on prend comme fonction ϕ une fonction indicateur on retrouve le cas discret. Pour illustration regardons la situation unidimensionelle avec $\phi_i(s) = 1$ si $i\,\Delta s < s \leq (i+1)\,\Delta s$ et zéro sinon. L'équation (6.22) devient alors identique à la règle de SARSA, Eq. (6.14), avec $\gamma = 1$.

6.5 Trace d'éligibilité

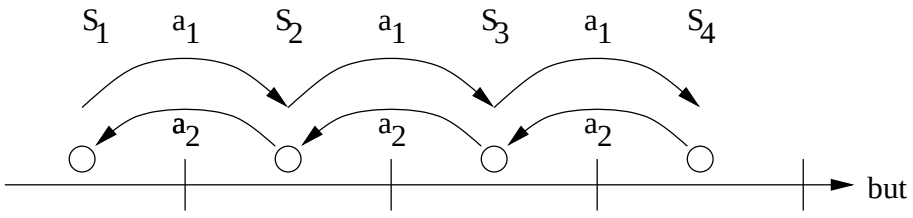

FIG. 6.7 – *Problème à une dimension, discrétisée en trois états.*

1. W. Schultz, P. Dayan and R.R. Montague: *A neural substrate for prediction and reward*, Science 257:1593-1599 (1997)

Considérons le problème illustré par la figure 6.7. On y retrouve un problème mono-dimensionel avec une zone de récompense à droite. Tant que nous n'avons pas atteint le but final, nous n'obtenons pas de récompense puisque, à priori, nous ne savons pas où se trouve le but, et ne savons donc pas si nous nous en approchons ou nous en éloignons. Regardons le comportement de l'algorithme SARSA. Supposons que nous avons initialisé toutes les valeurs Q à une même valeur, par exemple $Q(s,a) = 0$ pour tout s,a. Si nous appliquons la formule 6.14, la mise-à-jour des Q ne changera donc pas leurs valeurs – sauf pour la dernière action avant l'arrivée au but.

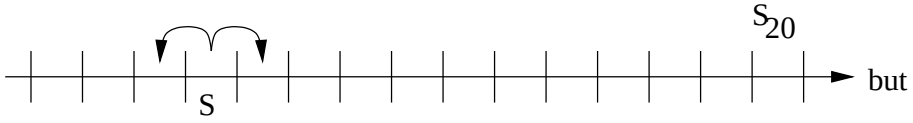

FIG. 6.8 – *Même problème, discrétisé en 20 états.*

Considérons d'abord une discrétisation à 3 états, comme sur la figure 6.7.

- Au 1^{er} essai, mise à jour de $Q(s_3,a_1)$.
- Au 2^e essai, mise à jour de $Q(s_2,a_1)$.
- Au 3^e essai, mise à jour de $Q(s_1,a_1)$.

L'information diffuse rapidement en 3 essais sur toute la distance.

Considérons maintenant une discrétisation à 20 états, comme sur la figure 6.8.

- Au 1^{er} essai, mise à jour de $Q(s_{20},a_1)$.
- Au 2^e essai, mise à jour de $Q(s_{19},a_1)$.
- ...

La diffusion de l'information est très lente. En plus, elle dépend de notre choix de discrétisation. Avec une discrétisation plus fine, le problème deviendra encore plus long à résoudre.

On aimerait bien que la diffusion de l'information sur le but soit plus rapide et ne dépende pas de notre choix de discrétisation. La solution à ce problème consiste en l'utilisation d'une "mémoire de trajectoire", ou une trace d'éligibilité: $e_k^{a_i}$. Lors d'une itération, nous avons alors:

- l'état actuel s.
- l'action actuelle a_1.
- et nous mettons les mémoires à jour:
  - Pour $a_j = a_1$: $e_k^{a_1}(t) = \lambda\, e_k^{a_1}(t-1) + \Phi_k(s)$
  - Pour $a_j \neq a_1$: $e_k^{a_j}(t) = \lambda\, e_k^{a_j}(t-1)$
  - Mise-à-jour des poids: $\Delta w_k^{a_j} = \eta \cdot \rho \cdot e_k^{a_j}(t)$

Remarques. (i) On peut noter que pour $\lambda = 0$, nous retombons sur l'algorithme de SARSA du problème continu.
(ii) Si l'on mets Φ égale à la fonction indicateur mentionné dans la section précedente, on trouve une version de SARSA avec trace d'éligibilité pour un système avec des états discrèts.

Une illustration de cette trace d'éligibilité se trouve dans la figure 6.9. Il s'agit d'un problème en deux dimension avec une représentation avec des états discrets.

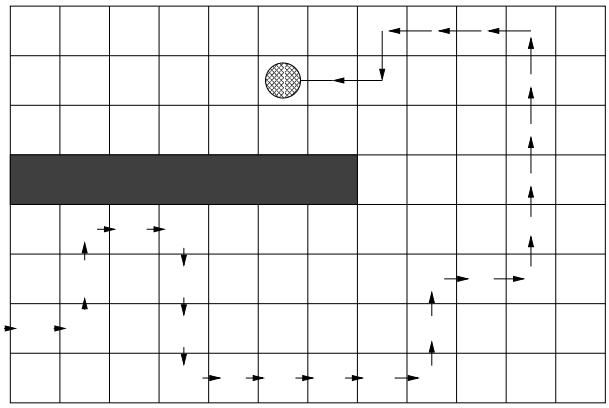

FIG. 6.9 – *La trace d'éligibilité. Chaque action prise dans le passé est notée par une flèche avec une longeur proportionelle à* e_k^a. *Un obstacle est marqué en noir, le but marqué par un cercle est le seul point de récompense. Au moment où l'agent atteint le but, toutes les valeurs* $Q(s,a)$ *des actions prises dans le passé sont mises à jour proportionellement à la valeur d'*e_k^a.

6.5.1 Applications

Il y a plusieurs applications phare des algorithmes de l'apprentissage par renforcement.

La première est le système de l'acrobot indiqué dans la figure 6.10. Le système a deux degrés de liberté noté par α et β. Le problème consiste à mettre le bout de l'acrobot au-dessus la ligne horizontale en appliquant des actions a qui mettent un moment de rotation sur l'angle β. Le problème peut être résolu à l'aide d'un algorithme Sarsa avec trace de mémoire. L'apprentissage se fait pendant 200-400 épisodes environ. Chaque épisode se terminent lorsque le but est atteint. Les premièrs episodes prennent plus de mille pas d'itérations. Après l'apprentissage les épisodes se terminent déjà après moins de 100 pas d'itérations. Voir [SB98] pour les détails.

A **B**

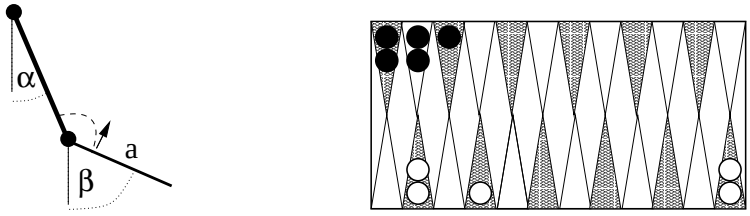

FIG. 6.10 – **A**. *Schéma de l'acrobot et* **B** *le jeu de Backgammon*

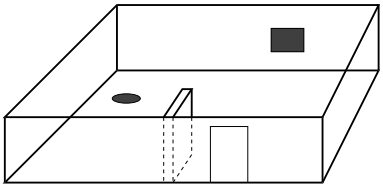

FIG. 6.11 – *L'environnement pour des expériences avec un rat. Le rat apprend à s'approcher du but (cercle) grâce au conditionnement.*

Les algorithmes d'apprentissage par renforcement s'adaptent bien pour des jeux. Pour le jeu du backgammon, le défi principal consiste à coder les états actuels du jeu. Si on modélise la qualité d'états $V(s)$ comme fonction d'états par un réseau de neurones, on peut appliquer un algorithme de type $TD(\lambda)$. Deux implémentations de cette idée peuvent jouer l'une contre l'autre en s'améliorant mutuellement. Une telle approche s'avère très puissante et bat les meilleurs joueurs du monde.

L'apprentissage par renforcement est probablement aussi à la base de l'apprentissage chez l'animal. Pour illustrer l'idée de base considérons un rat qui se promène pour la première fois à l'intérieur d'un environnement où il trouve un morceau de chocolat caché sous le sol. Si l'expérience se repète tous les jours, le rat apprend rapidement à retrouver l'endroit exact. Le chocolat sert de renforcement. Le processus d'apprentissage par renforcement chez l'animal est aussi appelé 'condionnement'.