Apprentissage par Renforcement sans Modèle et avec Action Continue ⋆

Thomas Degris[1], Patrick M. Pilarski[2], Richard S. Sutton[2]

[1] INRIA Bordeaux Sud-Ouest
351 cours de la libération - 33405 Talence Cedex
thomas.degris@inria.fr

[2] University of Alberta, Edmonton, T6G 2E8, AB, Canada

Résumé : L'apprentissage par renforcement est souvent considéré comme une solution potentielle pour permettre à un robot de s'adapter en temps réel aux changements imprédictibles d'un environnement ; mais avec des actions continues, peu d'algorithmes existants sont utilisables pour un tel apprentissage temps réel. Les méthodes les plus efficaces utilisent une politique paramétrée, souvent en combinaison avec une estimation, elle aussi paramétrée, de la fonction de valeur de cette politique. Le but de cet article est d'étudier de telles méthodes acteur–critique afin de constituer un algorithme complètement spécifié et utilisable en pratique. Nos contributions incluent 1) le développement d'une extension des algorithmes d'optimisation de politique par gradient pour l'utilisation des traces d'éligibilité, 2) une comparaison empirique des algorithmes résultants pour des actions continues, 3) l'évaluation d'une technique de mise à l'échelle du gradient qui peut améliorer les performances significativement. Finalement, nous appliquerons l'un de ces algorithmes sur un robot avec une boucle sensori-motrice rapide (10ms). L'ensemble de ces résultats constitue une étape importante pour la conception d'algorithmes de contrôle avec des actions continues et facilement utilisable en pratique.
Mots-clés : Apprentissage par renforcement, actions continues, robotique, acteur-critique

1 Apprentissage par Renforcement en Temps Réel sur des Robots

Il est souvent souhaitable en robotique de pouvoir adapter un comportement en temps réel en réaction à des changements de l'environnement. Un exemple typique est celui d'un robot aspirateur qui adapterait les paramètres de ses servo-moteurs en fonction de la texture du sol afin d'améliorer sa performance et économiser de l'énergie. Anticiper tous les types possibles de sol, ajouter des capteurs afin de détecter certaines des caractéristiques du sol, puis déterminer les meilleurs paramètres à utiliser seraient à la fois difficile, voir impossible, et couteux. Une solution alternative est d'utiliser un apprentissage en ligne afin de pouvoir observer les changements de l'environnement pour s'y adapter, comme cela a été montré par Sutton *et al.* (2007). Si l'apprentissage est en temps réel et continu, alors le robot peut en permanence régler ses paramètres en fonction de l'expérience vécue pendant le nettoyage d'une pièce et donc, s'adapter en permanence aux changements de caractéristique du sol.

Étonnamment, il y a assez peu d'algorithmes d'apprentissage par renforcement qui ont été utilisés pour un apprentissage temps réel sur des robots. Par exemple, Kohl & Stone (2004) ont utilisé des méthodes d'apprentissage par renforcement pour améliorer la marche d'un chien robot, mais la politique était mise à jour hors-ligne, de façon incrémentale certes. De la même manière, un robot capable de marcher avec une démarche dynamique, présenté par Tedrake *et al.* (2005), utilisait l'apprentissage par renforcement pour améliorer sa politique, mais les mises-à-jour étaient faites seulement à la fin de un ou plusieurs cycle de pas. Peters & Schaal (2008) montre un exemple d'apprentissage du contrôle d'une batte de base-ball, mais la politique était changée seulement après un ou plusieurs épisodes. Un dernier exemple est celui de Abbeel *et al.* (2010), qui apprend le contrôle d'un hélicoptère hors-ligne. Notons que, en plus du fait de ne pas utiliser un apprentissage en ligne et en temps réel, tous ces exemples apprenaient à partir d'une politique déjà bien adaptée au problème : soit des politiques de type génération de trajectoire (Kohl & Stone, 2004;

⋆. La version anglaise de cet article a été publiée à l'occasion de la conférence: American Control Conference en 2012.

Tedrake *et al.*, 2005), soit à partir d'imitations (Peters & Schaal, 2008; Abbeel *et al.*, 2010). Suivant le problème, de telles connaissances a priori ne sont pas toujours disponibles.

Le premier exemple d'apprentissage par renforcement en temps réel sur un robot que nous connaissons est celui de Benbrahim *et al.* (1992), qui ont appliqué une méthode acteur–critique avec des actions discrètes sur une implémentation physique d'une tâche de maintien en équilibre d'une balle sur un plateau ; le système apprenait avec un temps réel d'approximativement 55ms par cycle. Il semblerait que les premiers travaux d'apprentissage par renforcement temps réel sur un robot mobile conventionnelle soient ceux de Bowling & Veloso (2003). Leur système utilisait aussi des actions discrètes et un cycle d'environ 100ms.

En dépit de ces deux derniers exemples, en pratique, il peut être difficile d'appliquer des techniques d'apprentissage par renforcement pour de l'apprentissage en temps réel sur des robots. En robotique, les actions sont souvent continues alors que la majorité des travaux en apprentissage par renforcement considèrent des actions discrètes, y compris les deux exemples cités ci-dessus. De plus, afin de pouvoir apprendre en temps réel, un algorithme d'apprentissage par renforcement doit satisfaire deux exigences clés. La première exigence est que la complexité du calcul doit être linéaire avec le nombre de paramètres de la politique à apprendre. Par exemple, l'algorithme acteur–critique naturel (Peters & Schaal, 2008) n'est pas adapté pour une utilisation en ligne en temps réel ; en effet, bien que sa complexité en terme de nombre d'échantillon pour l'apprentissage soit souvent inférieur à un acteur–critique ordinaire, sa complexité de calcul quadratique est problématique lorsque le nombre de poids est trop important ou que le problème requiert un cycle de mise à jour très rapide. La deuxième exigence est que l'algorithme doit être strictement incrémental, c'est-à-dire que sa complexité de calcul ne doit pas augmenter au cours du temps (Sutton & Whitehead, 1993).

Dans cet article, nous regroupons plusieurs contributions afin de former un algorithme d'apprentissage par renforcement complet et utilisable en pratique. Plus particulièrement, nous partons du travail théorique de Bhatnagar *et al.* (2009) pour l'étendre, d'une part, aux actions continues, comme déjà exploré par Williams (1992), et d'autre part, aux traces d'éligibilité de façon similaire à Kimura *et al.* (2002).

L'article est structuré de la façon suivante : en premier lieu, nous introduisons le formalisme théorique. Dans ce formalisme, nous décrivons un ensemble d'algorithmes faciles à implémenter et avec une complexité linéaire. Nous utilisons ensuite une architecture basée sur un codage en grille [1] (Sutton & Barto, 1998) afin d'analyser leur performance sur deux études empiriques. Enfin, nous démontrons que cette architecture est utilisable pour un apprentissage en temps réel du contrôle d'un robot et que, grâce à une exploration adaptative, le système est capable d'adapter sa politique à un environnement bruité et non-stationnaire avec un cycle rapide de 10ms et une connaissance a priori limitée.

2 Formalisme d'Optimisation d'une Politique par Gradient

Nous considérons un problème classique d'apprentissage par renforcement (Sutton & Barto, 1998), à l'exception près d'un espace d'actions $\mathcal{A}$ continu. Nous supposons que l'espace d'état $\mathcal{S}$ est discret pour simplifier la présentation de la théorie ; bien sûr les problèmes utilisés pour la validation expérimentale auront un espace d'état continu. Nous utilisons le formalisme d'optimisation d'une politique par gradient dans lequel une politique stochastique π est (implicitement) paramétrée par un vecteur de poids $\mathbf{u} \in \Re^N$, avec $\pi(a|s)$ représentant la densité de probabilité de pendre l'action a dans l'état s.

Une fonction objectif $J(\pi)$ associe une politique à un nombre scalaire représentant la performance de cette politique. L'idée principale des méthodes d'optimisation d'une politique par gradient est d'améliorer une politique en mettant à jour son vecteur de poids approximativement proportionnellement au gradient de la fonction objectif :

$$\mathbf{u}_{t+1} - \mathbf{u}_t \approx \alpha_u \nabla_{\mathbf{u}} J(\pi), \tag{1}$$

où $\alpha_u \in \Re$ est un paramètre de pas d'apprentissage positif et où $\nabla_{\mathbf{u}} J(\pi) \in \Re^N$ est le gradient de la fonction objective par rapport aux poids $\mathbf{u}$ de la politique à améliorer.

2.1 Définition de la Fonction Objectif

Suivant la nature du problème, deux définitions différentes peuvent être considérées pour définir la fonction objectif $J(\pi)$. Le premier cas est lorsque l'interaction entre un agent et son environnement est continue sans interruption et sans fin d'épisode : on utilise alors le critère de récompense moyenne. Dans

1. *Tile-coding* en anglais

ce cas, les politiques sont évaluées suivant l'espérance de la récompense moyenne par pas de temps : $J(\pi) = \lim_{t\to\infty} \frac{1}{t}\mathrm{E}_\pi [r_1 + r_2 + \ldots + r_t]$.

Le deuxième cas est celui du critère de l'état de départ : un agent doit maximiser la récompense totale accumulée à partir d'un état de départ s_0 connu et jusqu'à ce qu'un évènement spécial termine l'épisode. Une politique est évaluée par l'espérance de la somme pondérée des récompenses en partant de s_0 et en terminant au pas de temps T_{ep} : $J(\pi) = \mathrm{E}_\pi \left[\sum_{t=1}^{T_{ep}} \gamma^{t-1} r_t \middle| s_0\right]$ où $\gamma \in [0, 1]$ est le paramètre de pondération.

Pour les deux cas, nous recherchons des algorithmes satisfaisant l'équation 1 à chaque pas de temps. Le reste de cette section présente une analyse théorique de la vue-vers-l'avant [2] des traces d'éligibilité. La prochaine section convertira cette vue-vers-l'avant en une vue-vers-l'arrière [3] afin d'arriver à un algorithme en ligne purement incrémental et utilisant les traces d'éligibilité.

2.2 Vue-vers-l'avant

Le théorème du gradient de la politique (Sutton *et al.*, 2000) peut être étendu aux actions continues de la façon suivante :

$$\nabla_{\mathbf{u}} J(\pi) = \sum_{s\in\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \nabla_{\mathbf{u}} \pi(a|s) Q^\pi(s, a) da, \tag{2}$$

où, pour le critère de la récompense moyenne, $d^\pi(s) = \lim_{t\to\infty} P(s_t = s|s_0, \pi)$ est une distribution des états pour la politique π, avec $P(s_t = s|s_0, \pi)$ étant la probabilité que $s_t = s$ partant de l'état s_0 puis exécutant π. Pour le critère de l'état de départ, $d^\pi(s) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s|s_0, \pi)$ est une distribution pondérée des états pour π.

Nous définissons la fonction de valeur d'action $Q^\pi(s, a) = \mathrm{E}_\pi [\sum_{t=0}^{\infty} \gamma^t r_t - \bar{r}(\pi)|s_0 = s, a_0 = a]$ avec, pour le critère de récompense moyenne, $\gamma = 1$ et $\bar{r}(\pi) = J(\pi)$ est la moyenne des récompenses pour la politique π et, pour le critère de l'état de départ, $\bar{r}(\pi)$ toujours égal à 0.

Par ailleurs, l'équation 2 peut être généralisée pour inclure une fonction de base arbitraire ne dépendant que des états et que nous notons $b : \mathcal{S} \to \Re$:

$$\nabla_{\mathbf{u}} J(\pi) = \sum_{s\in\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \nabla_{\mathbf{u}} \pi(a|s) [Q^\pi(s, a) - b(s)] da, \tag{3}$$

puisque $\int_{\mathcal{A}} \nabla_{\mathbf{u}} \pi(a|s) da = 0$ (Bhatnagar *et al.*, 2009). Alors que l'utilisation de cette fonction de base ne change pas l'égalité, elle peut souvent diminuer la variance de l'estimation du gradient.

Notons que lorsqu'une politique π est exécutée, la distribution des états observée suit $d^\pi(s)$ et que les actions sont sélectionnées suivant π. Par conséquent, l'équation 3 peut être écrite sous la forme d'une espérance :

$$\nabla_{\mathbf{u}} J(\pi) = \mathrm{E}_\pi \left[\frac{\nabla_{\mathbf{u}} \pi(a_t|s_t)}{\pi(a_t|s_t)} (Q^\pi(s_t, a_t) - b(s_t)) \right], \tag{4}$$

définie par la somme pondérée sur les états s et les actions a échantillonnés d'après leur distribution respective, noté $s \sim d^\pi(\cdot)$ et $a \sim \pi(\cdot|s)$.

Le vecteur $\frac{\nabla_{\mathbf{u}} \pi(a|s)}{\pi(a|s)}$ est appelé le vecteur des fonctions compatibles [4] (Bhatnagar *et al.*, 2009), c'est-à-dire le vecteur gradient compatible avec le vecteur de fonctions utilisé pour estimer la politique. La prochaine étape est d'écrire l'équation 4 comme une espérance du gain [5] (Sutton & Barto, 1998) :

$$\nabla_{\mathbf{u}} J(\pi) = \mathrm{E}_\pi \left[\frac{\nabla_{\mathbf{u}} \pi(a_t|s_t)}{\pi(a_t|s_t)} (R_t^\pi - b(s_t)) \right], \tag{5}$$

avec $s_t \sim d^\pi(\cdot)$, $a_t \sim \pi(\cdot|s_t)$, et le gain $R_t^\pi = r_t - \bar{r}(\pi) + \gamma r_{t+1} - \bar{r}(\pi) + \gamma^2 r_{t+2} - \bar{r}(\pi) + \ldots$. La partie gauche de l'équation 5 ne dépend maintenant que d'observations directement accessibles en exécutant la politique π.

2. *Forward-view* en anglais
3. *Backward-view* en anglais
4. *compatible features* en anglais
5. *return* en anglais

3 Algorithmes

Le gain R_t^π n'est souvent pas connu au pas de temps t puisqu'il dépend des récompenses reçues dans le future, mais il peut être approché par $R_t^{\pi,\lambda}$, nommé le gain-λ [6] (Sutton & Barto, 1998), et défini tel que : $R_t^{\pi,\lambda} = r_{t+1} - \overline{r} + \gamma(1-\lambda)v(s_{t+1}) + \gamma\lambda R_{t+1}^{\pi,\lambda}$ avec $\lambda \in [0,1]$, où $\overline{r}$ est une estimation de la récompense moyenne (lors de l'exécution de π) pour le critère de la récompense moyenne et égal à 0 pour le critère de l'état de départ, et où $v(s_{t+1})$ est une estimation de la fonction de valeur de la politique π. À partir de l'équation 5, nous pouvons maintenant définir un algorithme de vue-vers-l'avant en définissant la mise à jour des paramètres de la politique à l'instant t de la façon suivante :

$$\begin{aligned}\mathbf{u}_{t+1} - \mathbf{u}_t &= \alpha_u \left(R_t^{\pi,\lambda} - b(s_t)\right) \frac{\nabla_{\mathbf{u}} \pi(a_t|s_t)}{\pi(a_t|s_t)} \\ &= \alpha_u \delta_t^{\pi,\lambda} \frac{\nabla_{\mathbf{u}} \pi(a_t|s_t)}{\pi(a_t|s_t)} \end{aligned} \tag{6}$$

où s_t et a_t sont l'état et l'action à l'instant t, et $\delta_t^{\pi,\lambda} = R_t^{\pi,\lambda} - b(s_t)$. Alors que n'importe quelle fonction dépendant seulement de l'état (et non de l'action) peut être utilisée comme fonction de base $b(s_t)$, un choix naturel est l'estimation de la fonction de valeur $v(s)$ de la politique, apprise par un critique dans une architecture acteur-critique. Remarquons que pour le critère de l'état de départ et lorsque $\lambda = 1$, l'équation 6 est équivalente à une mise-à-jour de Monte Carlo. Cependant, un tel algorithme n'est pas utilisable pour un apprentissage en ligne puisqu'il nécessite de connaître les valeurs des récompenses à venir, et donc de se souvenir de toutes les valeurs individuelles passées de $\frac{\nabla_{\mathbf{u}} \pi(a_t|s_t)}{\pi(a_t|s_t)}$ afin de mettre à jour les paramètres de la politique à la fin d'un épisode.

3.1 Vue-vers-l'arrière

L'utilisation des traces d'éligibilité permet de résoudre le problème d'avoir à se souvenir des valeurs passées de $\frac{\nabla_{\mathbf{u}} \pi(a_t|s_t)}{\pi(a_t|s_t)}$; pour cela, un vecteur de trace d'éligibilité est maintenu. En passant de la vue-vers-l'avant définie par l'équation 6 vers une vue-vers-l'arrière, on définit la mise-à-jour du vecteur de trace d'éligibilité :

$$\mathbf{u}_{t+1} - \mathbf{u}_t = \alpha_u \delta_t^{\pi,\lambda} \frac{\nabla_{\mathbf{u}} \pi(a_t|s_t)}{\pi(a_t|s_t)} = \alpha_u \delta_t \mathbf{e}_{\mathbf{u}} \tag{7}$$

où $\mathbf{e}_{\mathbf{u}}$ est le vecteur de trace d'éligibilité des fonctions compatibles $\frac{\nabla_{\mathbf{u}} \pi(a_t|s_t)}{\pi(a_t|s_t)}$ avec :

$$\mathbf{e}_{\mathbf{u}} \leftarrow \lambda \mathbf{e}_{\mathbf{u}} + \frac{\nabla_{\mathbf{u}} \pi(a_t|s_t)}{\pi(a_t|s_t)},$$

et où δ_t est l'erreur TD définie telle que $\delta_t = r_{t+1} - \overline{r} + \gamma v(s_{t+1}) - v(s_t)$.

3.2 Algorithmes d'apprentissage en ligne

Nous présentons maintenant deux algorithmes en utilisant la convention suivante. Pour le critère de la récompense moyenne, $\gamma = 1$ et $0 < \alpha_r < 1$ est le pas d'apprentissage pour l'estimation de la moyenne de la récompense. Pour le critère de l'état de départ, $\gamma \in [0;1]$ et $\alpha_r = 0,0$.

D'après l'équation 7 de la vue-vers-l'arrière, on peut définir directement un algorithme acteur–critique avec traces d'éligibilité et noté AC. AC est décrit ci-dessous. AC met tout d'abord à jour une estimation de la moyenne de la récompense $\overline{r}$ (qui peut être comme considéré une trace de la récompense immédiate r). La fonction de valeur est ensuite estimée par la combinaison linéaire $v(s) = \mathbf{v}^\mathsf{T} \mathbf{x}_{\mathbf{v}}(s)$; où $\mathbf{v}$ est un vecteur de poids et $\mathbf{x}_{\mathbf{v}}(s_t)$ est un vecteur de fonction [7] correspondant à l'état s. Ensuite, le critique met à jour le vecteur de poids $\mathbf{v}$ en utilisant l'algorithme TD(λ) (Sutton, 1988), avec $\alpha_v \geq 0$ le pas d'apprentissage du critique. Puis les paramètres de la politique $\mathbf{u}$ de l'acteur sont mis à jour à partir de l'erreur TD δ et le vecteur de trace des fonctions compatibles $\mathbf{e}_{\mathbf{u}}$.

6. *λ-return* en anglais
7. *Feature vector* en anglais

Algorithme Acteur-critique (noté AC, ou noté A lorsque $\alpha_v = 0$)

Choisir l'action a d'après $\pi(a|s)$
Exécuter l'action a dans l'état s, observer s' et r
$\overline{r} \leftarrow \overline{r}(1-\alpha_r) + \alpha_r r$
$\delta \leftarrow r - \overline{r} + \gamma \mathbf{v}^\mathsf{T} \mathbf{x}_\mathbf{v}(s') - \mathbf{v}^\mathsf{T} \mathbf{x}_\mathbf{v}(s)$
$\mathbf{e}_\mathbf{v} \leftarrow \lambda \gamma \mathbf{e}_\mathbf{v} + \mathbf{x}_\mathbf{v}(s)$
$\mathbf{v} \leftarrow \mathbf{v} + \alpha_v \delta \mathbf{e}_\mathbf{v}$
$\mathbf{e}_\mathbf{u} \leftarrow \lambda \mathbf{e}_\mathbf{u} + \frac{\nabla_\mathbf{u} \pi(a|s)}{\pi(a|s)}$
$\mathbf{u} \leftarrow \mathbf{u} + \alpha_u \delta \mathbf{e}_\mathbf{u}$

L'algorithme AC est similaire à l'algorithme introduit par Kimura *et al.* (2002), bien que ce dernier n'avait pas la justification théorique présentée dans cet article. L'algorithme AC sans le critique ($\alpha_v = 0$) est similaire à REINFORCE et OLPOMDP (Baxter & Bartlett, 2002), que nous notons A.

L'algorithme acteur–critique incrémental du gradient naturel est un nouvel algorithme. Il est noté INAC et est décrit ci-dessous. INAC étend l'algorithme proposé par Bhatnagar *et al.* (2009) afin d'utiliser les traces d'éligibilité. INAC utilise le gradient naturel $\tilde{\nabla}_\mathbf{u} J(\pi) = G(\mathbf{u})^{-1} \nabla_\mathbf{u} J(\pi)$, où $G(\mathbf{u})$ est la matrice d'information de Fisher $G(\mathbf{u}) = \sum_{s \in \mathcal{S}} d^\pi(s) \int_\mathcal{A} \pi(a|s) \frac{\nabla_\mathbf{u} \pi(a_t|s_t)}{\pi(a_t|s_t)} \frac{\nabla_\mathbf{u} \pi(a_t|s_t)}{\pi(a_t|s_t)}^\mathsf{T}$. Comme l'algorithme précédent, le vecteur de poids du critique est mis à jour par TD(λ). Ensuite, le vecteur $\mathbf{w}$ est mis à jour puis utilisé comme une estimation du gradient naturel pour mettre à jour le vecteur de poids de l'acteur.

Algorithme incrémental acteur–critique du gradient naturel (noté INAC)

Choisir l'action a d'après $\pi(a|s)$
Exécuter l'action a dans l'état s, observer s' et r
$\overline{r} \leftarrow \overline{r}(1-\alpha_r) + \alpha_r r$
$\delta \leftarrow r - \overline{r} + \gamma \mathbf{v}^\mathsf{T} \mathbf{x}_\mathbf{v}(s') - \mathbf{v}^\mathsf{T} \mathbf{x}_\mathbf{v}(s)$
$\mathbf{e}_\mathbf{v} \leftarrow \lambda \gamma \mathbf{e}_\mathbf{v} + \mathbf{x}_\mathbf{v}(s)$
$\mathbf{v} \leftarrow \mathbf{v} + \alpha_v \delta \mathbf{e}_\mathbf{v}$
$\mathbf{e}_\mathbf{u} \leftarrow \lambda \mathbf{e}_\mathbf{u} + \frac{\nabla_\mathbf{u} \pi(a|s)}{\pi(a|s)}$
$\mathbf{w} \leftarrow \mathbf{w} - \alpha_v \frac{\nabla_\mathbf{u} \pi(a|s)}{\pi(a|s)} \frac{\nabla_\mathbf{u} \pi(a|s)}{\pi(a|s)}^\mathsf{T} \mathbf{w} + \alpha_v \delta \mathbf{e}_\mathbf{u}$
$\mathbf{u} \leftarrow \mathbf{u} + \alpha_u \mathbf{w}$

Les deux algorithmes AC et INAC ont une preuve de convergence lorsque $\lambda = 0$, étant donné plusieurs restrictions concernant le problème et la valeur des paramètres (Bhatnagar *et al.*, 2009). Notons que ces deux algorithmes utilisent des représentations approchées de la fonction de valeur et de la politique, c'est-à-dire qu'ils utilisent le vecteur $\mathbf{x}_\mathbf{v}(s)$ pour représenter la fonction de valeur dans le critique, et le vecteur $\mathbf{x}_u(s)$ pour représenter la politique de l'acteur (voir la section suivante). Les preuves proposées par Bhatnagar *et al.* (2009) s'appliquent donc aussi lorsque l'hypothèse de Markov n'est pas satisfaite pour l'état de l'agent, c'est-à-dire lorsqu'elle n'est pas satisfaite pour $\mathbf{x}_\mathbf{v}(s)$ et $\mathbf{x}_u(s)$. Enfin, une preuve de convergence pour $\lambda \neq 0$ est en dehors du propos de cet article ; Bhatnagar *et al.* (2009) mentionne le fait que leur preuve devrait s'étendre à ce cas.

4 Structure de la politique pour des actions continues

Les algorithmes décrits lors de la section précédente sont indépendants de la structure de politique utilisée. Pour des actions discrètes, une distribution de Gibbs est souvent bien adaptée. Dans cet article, pour des actions continues, comme suggéré par Williams (1992), nous choisissons une politique telle que les actions sont décidées suivant une distribution normale dont la densité de probabilité est : $\mathcal{N}(s,a) = \frac{1}{\sqrt{2\pi\sigma^2(s)}} \exp\left(-\frac{(a-\mu(s))^2}{2\sigma^2(s)}\right)$ où $\mu(s)$ et $\sigma(s)$ sont respectivement la moyenne et l'écart type de la distribution $\pi(\cdot|s)$.

Nous paramétrons cette politique en définissant les scalaires $\mu(s) = \mathbf{u}_\mu{}^\mathsf{T} \mathbf{x}_\mu(s)$ et $\sigma(s) = \exp\left(\mathbf{u}_\sigma{}^\mathsf{T} \mathbf{x}_\sigma(s)\right)$, où les paramètres de la politique sont $\mathbf{u} = \left(\mathbf{u}_\mu{}^\mathsf{T}, \mathbf{u}_\sigma{}^\mathsf{T}\right)^\mathsf{T}$, et le vecteur de fonctions pour un état s pour l'acteur est $\mathbf{x}_u(s) = \left(\mathbf{x}_\mu(s)^\mathsf{T}, \mathbf{x}_\sigma(s)^\mathsf{T}\right)^\mathsf{T}$.

Les fonctions compatibles $\frac{\nabla_{\mathbf{u}}\pi(a|s)}{\pi(a|s)}$ dépendent des fonctions de probabilité de densité de la politique. Comme nous avons choisi une distribution normale, les fonctions compatibles pour la moyenne et l'écart type sont (Williams, 1992) :

$$\frac{\nabla_{\mathbf{u}_\mu}\pi(a|s)}{\pi(a|s)} = \frac{1}{\sigma(s)^2}\left(a-\mu(s)\right)\mathbf{x}_\mu(s) \quad (8)$$

$$\frac{\nabla_{\mathbf{u}_\sigma}\pi(a|s)}{\pi(a|s)} = \left(\frac{\left(a-\mu(s)\right)^2}{\sigma(s)^2}-1\right)\mathbf{x}_\sigma(s) \quad (9)$$

où $\frac{\nabla_{\mathbf{u}}\pi(a|s)}{\pi(a|s)} = (\frac{\nabla_{\mathbf{u}_\mu}\pi(a|s)}{\pi(a|s)}^\mathsf{T}, \frac{\nabla_{\mathbf{u}_\sigma}\pi(a|s)}{\pi(a|s)}^\mathsf{T})^\mathsf{T}$.

Le vecteur de fonctions compatibles défini par l'équation 8, utilisé pour mettre à jour les paramètres $\mathbf{u}_\mu$ de la politique, a un facteur $\frac{1}{\sigma(s)^2}$. En pratique, cela signifie que plus l'écart type sera proche de zéro, plus grande sera la norme de $\frac{\nabla_{\mathbf{u}_\mu}\pi(a_t|s_t)}{\pi(a_t|s_t)}$ et vice-versa. Nous avons observé qu'un tel effet peut causer des instabilités, notamment parce que $\lim_{\sigma\to 0}\frac{(a-\mu(s))}{\sigma(s)^2}=\infty$.

Williams (1992) suggère d'utiliser un pas d'apprentissage de la forme $\alpha_u\sigma^2$ pour une distribution normale, changeant ainsi l'échelle du gradient en fonction de la variance de la distribution. Nous notons respectivement les algorithmes acteur–critique et acteur–critique incrémental du gradient naturel avec une telle mise-à-l'échelle AC-S and INAC-S.

5 Étude empirique

Nous présentons maintenant une étude empirique pour évaluer comment les idées décrites précédemment, c'est-à-dire l'utilisation d'un critique, l'utilisation du gradient naturel, et la mise à l'échelle du gradient de la politique, influencent la performance de l'apprentissage.

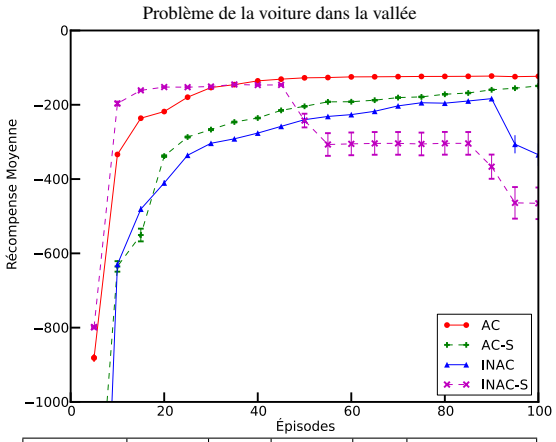
Dans cet article, nous avons utilisé une architecture basée sur la technique de codage en grille (Sutton, 1988) pour convertir un espace d'état continu en un vecteur de fonctions. Le codage en grille prend une observation de l'environnement comme entrée et la projette de façon non-linéaire dans un espace plus grand représenté par un vecteur de fonctions creux de grande dimension. Ce vecteur est ensuite linéairement combiné avec un vecteur de poids pour représenter des fonctions non linéaires. Le but de cet article n'est pas de comparer différentes architectures d'approximation de fonction, cependant le codage en grille a plusieurs avantages pratiques importants pour l'apprentissage en temps réel. Premièrement, le codage en grille est efficace d'un point de vue de complexité de calcul par pas de temps puisque le calcul du vecteur $\mathbf{x}$ de fonctions ne dépend pas de la taille du vecteur $\mathbf{x}$ lui-même, mais seulement du nombre de grilles utilisé. Deuxièmement, la norme du vecteur $\mathbf{x}$ est constante : elle est égale au nombre de grilles. Finalement, comme nous allons le montré dans la section suivante, le codage en grille est robuste au bruit.

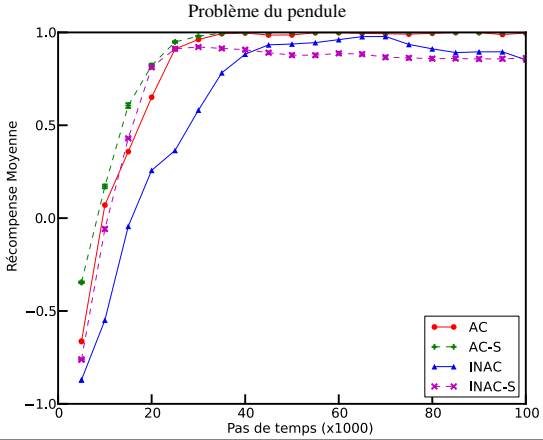
Le premier problème de l'étude est celui du problème de la voiture dans la vallée [8] (Sutton & Barto, 1998), dont le but est de conduire du fond d'une vallée vers le haut d'une colline une voiture sous-motorisée. Nous utilisons le critère de l'état de départ (qui correspond à démarrer au fond de la vallée). Les actions sont continues et bornée dans l'intervalle $[-1;1]$. La récompense à chaque pas de temps est -1 et un épisode se termine lorsque la voiture a atteint le haut de la colline sur la droite ou après 5.000 pas de temps. Les variables d'état observées par l'agent sont la position de la voiture (dans l'intervalle $[-1,2;0,6]$) et la vitesse de la voiture (dans l'intervalle $[-0,07;0,07]$). La voiture est initialisée dans la position -0,5 avec une vitesse de 0. Nous utilisons $\gamma=1$.

Le deuxième problème est le problème du pendule (Doya, 2000), pour lequel le but est de balancer un pendule et de le maintenir en position verticale. Nous utilisons le critère de la récompense moyenne. La récompense à chaque pas de temps est le cosinus de l'angle du pendule par rapport à sa base. Les actions— le couple appliqué à la base— sont restreintes à l'intervalle $[-2;2]$. Les variables d'état observées par l'agent sont l'angle du pendule avec sa base en radians et sa vitesse angulaire (dans l'intervalle $[-78,54;78,54]$). Le pendule est initialisé dans une position horizontale avec une vitesse angulaire de 0 puis réinitialisé tous les 1.000 pas de temps.

Pour les deux problèmes, nous avons utilisé dix grilles de 10×10 sur l'espace joint des deux variables d'état, plus une fonction toujours égale à 1. Nous avons utilisé le même vecteur de fonction $\mathbf{x}(s)$ pour

8. *Moutain car problem* en anglais





	α_u	α_v	α_n	τ	Reward
A	.05	na	na	1	-3575 ± 42
	.005	na	na	32	-3297 ± 47
AC	.005	1.0	na	1	-340 ± 2
	.01	1.0	na	2	$-189\pm.5$
AC-S	.1	.5	na	1	-567 ± 40
	.01	1.0	na	2	-308 ± 1
INAC	.1	1.0	.0005	1	-435 ± 33
	.001	.5	.01	4	-386 ± 5
INAC-S	.01	1.0	.05	1	-286 ± 16
	.0005	1.0	.05	2	-518 ± 30

	α_r	α_u	α_v	α_n	τ	Reward
A	.5	.05	na	na	1	$-.46\pm.006$
	.01	.005	na	na	16	$.82\pm.003$
AC	.01	.005	1.0	na	1	$.63\pm.003$
	.005	.005	.001	na	8	$.81\pm.001$
AC-S	.0005	.1	.5	na	1	$.79\pm.006$
	.001	0.05	.1	na	4	$.85\pm.002$
INAC	.001	.01	1.0	.001	1	$.52\pm.004$
	.01	.1	.005	.0001	16	$.63\pm.004$
INAC-S	.0001	.001	1.0	.05	1	$.52\pm.006$
	.01	.005	.01	.05	18	$.73\pm.006$

FIGURE 1 – Figure du dessus : *courbe d'apprentissage des algorithmes exécutés avec leur meilleurs paramètres trouvés*. Figure du dessous : *meilleurs valeurs des paramètres et moyenne des récompenses par exécution indépendante avec l'erreur standard. Les algorithmes avec la meilleur performance sont* AC *et* AC-S *avec les traces d'éligibilité. La mise à l'échelle du gradient (cf partie 4) a souvent amélioré la performance. L'algorithme* INAC *n'a pas obtenu un meilleur score que l'algorithme* AC.

le critique, et pour la moyenne et l'écart type de la politique de l'acteur, c'est-à-dire $\mathbf{x}_{\mathbf{v}}(s) = \mathbf{x}_{\mu}(s) = \mathbf{x}_{\sigma}(s)$. Nous avons évalué 30 exécutions indépendantes de toutes les combinaisons possibles des paramètres pour les algorithmes décrits ci-dessus (A, AC, AC-S, INAC, et INAC-S). Nous avons utilisé les neuf valeurs suivantes : $\{10^{-4}; 5 \cdot 10^{-4}; 10^{-3}; \ldots; 0,5; 1\}$ divisées par le nombre de fonction active ($10+1=11$) dans le vecteur d'état $\mathbf{x}(s)$ pour les paramètres α_r, α_v et α_u.

Le paramètre λ représente le taux auquel les traces d'éligibilité sont diminuées. Souvent, il est plus intuitif de penser à ce paramètre comme un nombre de pas de temps pendant lequel la trace persistera. Nous utilisons donc $\tau = \frac{1}{1-\lambda}$ pour noter le taux de diminution d'une trace. Nous avons utilisé les valeurs $\{1; 2; 4; 8; 16; 32\}$ pour le paramètre τ.

Tous les vecteurs ainsi que la récompense moyenne sont initialisés à 0. Pour le critère de l'état de départ, les traces d'éligibilité sont réinitialisées à 0 au début de chaque épisode.

La Figure 1 montre un résumé des résultats obtenus. Pour des questions de lisibilité, chaque point est une moyenne des 10 épisodes précédents pour le problème de la voiture dans la vallée et de 10.000 pas de temps pour le problème du pendule. Premièrement, nous observons qu'un critique améliore considérablement la performance. Deuxièmement, la meilleur performance pour chaque algorithme est quasiment toujours avec les traces d'éligibilité ($\tau > 1$), avec pour seule exception l'algorithme INAC-S sur le problème de la voiture dans la vallée. Troisièmement, la mise à l'échelle du gradient de la politique améliore souvent la performance, en particulier sur le problème du pendule. Enfin, l'utilisation du gradient naturel par les algorithmes INAC and INAC-S n'a pas permis d'améliorer leur performance comparé aux algorithmes AC et AC-S, en dépit d'un paramètre de pas d'apprentissage à définir en plus.

6 Économie d'énergie en temps réel sur un robot mobile

Pour démontrer l'utilité de ces algorithmes en pratique, nous avons évalué la performance de l'algorithme AC-S sur un robot. Nous considérons une tâche d'accélération d'un robot mobile tout en minimisant l'éner-

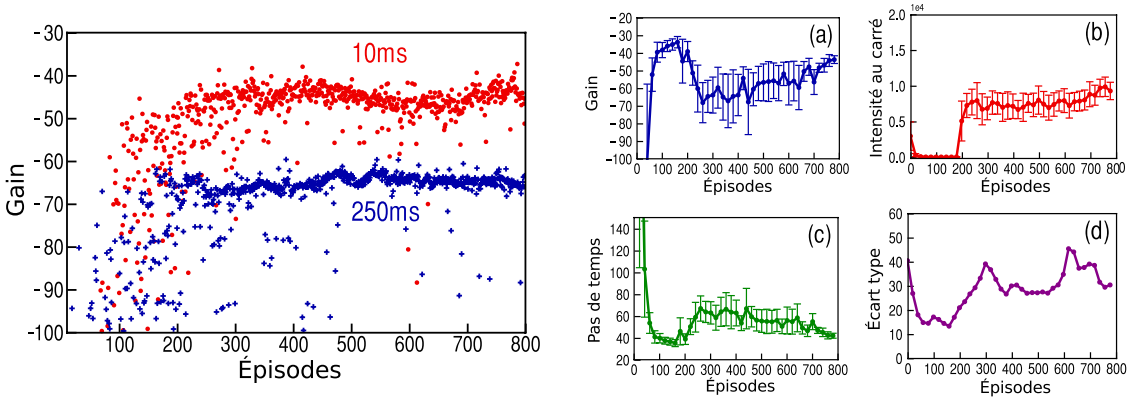


FIGURE 2 – Gauche : *Comparaison de la performance (somme des récompenses par épisode) de l'algorithme* AC-S *sur la tâche d'accélération du robot pour deux cycles de mise-à-jour différents : 10ms et 250ms. Les résultats sont moyennés sur six exécutions indépendantes pour chaque cycle. Le temps moyen d'exécution pour 800 épisodes est de 14,3 minutes.* Droite : *Apprentissage lorsqu'une transition, effectuée à l'épisode 200, entre un état où le robot ne touche pas le sol et un état où le robot bouge sur le sol : (a) gain par épisode, (b) somme de l'intensité électrique au carré par épisode, et (c) nombre de pas de temps jusqu'à que l'épisode se termine. Ces données sont les moyennes de deux exécutions indépendantes. Comme montré par la figure (d) pour une exécution, la valeur de apprise de l'écart type σ, et pouvant être considéré comme une indication du niveau d'exploration, augmente significativement après la transition. Le temps moyen d'exécution pour 800 épisodes est de 19,7 minutes.*

gie consommée. Comme exemples de cas pratique, nous pouvons citer les robots aspirateurs ou bien des robots de recherche contrôlés à distance : une utilisation efficace de l'énergie de leur batterie peut avoir un effet important sur leur capacité à remplir leur mission. Pendant les périodes d'accélération, des compromis doivent être réalisés entre la puissance de l'accélération et l'intensité électrique utilisée pour atteindre la vitesse désirée. Des environnements différents peuvent nécessiter des équilibres intensité/accélération différents, rendant pertinent des systèmes de contrôle des moteurs capable de s'adapter à leur environnement courant, y compris les environnements inconnus et non anticipés lors de la conception du robot.

Le robot utilisé dans cet article était un robot mobile avec trois roues omnidirectionnelles. La tâche d'apprentissage est formalisée en utilisant le critère de l'état de départ : lorsque l'épisode commençait, le robot avait une vitesse nulle, l'épisode se terminait lorsque la vitesse mesurée $\dot{x}_t$ de l'une des trois roues atteignaient ou dépassaient une vitesse cible $\dot{x}_t \geq \dot{x}^*$. L'agent contrôlait la vitesse de rotation des moteurs envoyée au robot, lui permettant de tourner vers la gauche, vers la droite, ou bien de rester immobile.

La fonction de récompense pour ce problème était définie telle que $r_t = -\left(\mathbf{1}_{\{\dot{x}_t<\dot{x}^*\}} + 0,5|i_t^2|\right)$ où $\mathbf{1}_{\{\dot{x}_t<\dot{x}^*\}}$ est une fonction indicateur et i_t est l'intensité de courant utilisée à l'instant t. La tâche est donc d'équilibrer deux contraintes conflictuelles en temps réel : atteindre la vitesse $\dot{x}^*$ rapidement pour éviter la récompense négative à chaque pas de temps et en même temps minimiser le courant utilisé par les roues. La vitesse et l'intensité de courant, respectivement $\dot{x}_t$ et i_t, constituent les observations disponibles à chaque pas de temps.

Le vecteur de fonctions était construit à partir d'une représentation en grille combinant les valeurs récentes des observations et l'action réalisée. À chaque pas de temps, dix grilles d'une résolution de $10\times10\times10$ combinent les variables $\langle \dot{x}_t, i_t, a_{t-1}\rangle$, dix grilles de même résolution combinent les variables $\langle \dot{x}_{t-1}, i_{t-1}, a_{t-2}\rangle$ et ce, jusqu'à $\langle \dot{x}_{t-4}, i_{t-4}, a_{t-5}\rangle$ pour un total de 50 grilles chacune d'une résolution de $10 \times 10 \times 10$, c'est-à-dire d'un vecteur de fonctions de 50.000 éléments auquel est rajouté une fonction toujours égale à 1. L'état de l'agent est donc un vecteur de 50.001 fonctions binaires, dont exactement $m = 51$ sont actives à chaque pas de temps. Nous avons utilisé $\alpha_v = 1,0/m; \alpha_u = 0,1/m; \alpha_r = \overline{r}_0 = 0; \gamma = 0,99$ et $\lambda = 0,7$. Pour l'acteur, seul la mise à jour de la moyenne était multipliée par la variance. Les vecteurs de poids étaient initialisés à 0. L'écart type $\mathbf{u}_\sigma$ était initialisé pour que $\sigma = 40$ (en définissant la valeur du poids correspondant à la fonction toujours égale à 1), et borné par $\sigma \geq 1$.

6.1 Expérience 1 : apprentissage d'une politique de contrôle en temps réel

Afin d'explorer la performance d'apprentissage de l'algorithme AC-S à différentes échelles de temps réel, la tâche d'accélération décrite ci-dessus fût exécutée en utilisant deux cycles différents de mise à jour : 10ms

et 250ms, où un cycle de temps correspond à la période avec laquelle les actions sont choisies et les vecteurs de poids mises à jour. Chaque exécution dure 800 épisodes. Six exécutions indépendantes ont été réalisées pour chaque cycle de mise-à-jour. Le but de cette expérience est d'illustrer la capacité de l'algorithme AC-S à optimiser une politique de contrôle des moteurs lorsqu'à la fois le choix des actions et l'apprentissage doivent être effectués en temps réel.

La figure 2 (gauche) montre les résultats obtenus. Pour les deux cycles de mise-à-jour, en moins de 15 minutes (en moyenne), AC-S a appris une politique de contrôle prenant en compte les contraintes imposées par la fonction de récompense et les contraintes physiques imposées par la période du cycle de mise-à-jour. Nous avons observé une différence statistiquement significative entre la performance asymptotique à 10ms comparé à celle de 250ms. Un apprentissage et un contrôle à 10ms conduisent à une récompense sur le long terme plus importante : le robot utilisa moins de courant avec des périodes d'accélération plus courtes, exploitant ainsi plus efficacement l'énergie disponible de la batterie pendant les phases d'accélération.

6.2 Expérience 2 : adaptation en ligne de la politique de contrôle

L'un des avantages d'une politique basée sur une distribution normale est le rôle de l'écart type σ dans la régulation de l'exploration. Cette expérience illustre comment AC-S a adapté sa politique en réponse à un changement inattendu de l'environnement avec un contrôle et un apprentissage en ligne et en temps réel avec un cycle de mise à jour de 10ms. Cette expérience est similaire à l'expérience précédente, excepté qu'au début de l'exécution, le robot est suspendu en l'air afin que ces roues ne touchent pas le sol. Après 200 épisodes, le robot est déplacé pour être déposé sur le sol. Dans ces nouvelles conditions, la masse du robot et les frottements entre les roues et le sol rendent l'équilibre entre la consommation de courant et la longueur de l'épisode différent, incitant l'algorithme à changer sa politique.

La figure 2 (droite) montre comment l'algorithme AC-S a été capable d'adapter en temps réel la valeur de σ utilisée dans la sélection des actions, modulant ainsi le degrés d'exploration suite au changement de l'environnement. Les résultats sont la moyenne de deux exécutions indépendantes, chaque point montrant la moyenne d'un regroupement des vingt valeurs voisines. Comme montré par les figures 2a–c, AC-S converge vers une politique adaptée à l'environnement initiale (200 premiers épisodes). Après le changement d'environnement (épisode 200), la somme des récompenses par épisodes décroît rapidement, pour ensuite s'améliorer continuellement lors de l'apprentissage d'une politique plus adaptée à un mouvement sur le sol (épisodes 201–800).

Lors du changement des conditions de l'environnement, c'est-à-dire lors de la transition entre le robot suspendu et le robot posé sur le sol, on observe une augmentation importante de l'écart type provoquant un changement dans la politique exécutée. Pendant qu'AC-S apprenait à maximiser le gain dans ces nouvelles conditions (figure 2a), la politique était graduellement ajustée afin de diminuer le nombre de pas de temps par épisode (figure 2c), au prix d'une augmentation de la quantité de courant par épisode (figure 2b). Pour l'une des deux exécutions, nous avons observé une augmentation de l'écart type une seconde fois lors de l'apprentissage d'une politique plus performante vers l'épisode 600.

7 Discussion

Les algorithmes présentés dans cet article ont tous été testés sur un ordinateur portable standard. Toutefois, le temps de réponse, comprenant à la fois la décision de l'action et l'apprentissage, était suffisamment court, et de loin, pour satisfaire les contraintes temps réels requises pour notre problème sur le robot. Nous pensons que la complexité linéaire de cette famille d'algorithmes acteur–critique est un avantage clé. De plus, aucun modèle de l'environnement n'a été nécessaire. Aucune réduction de bruit n'a été requise pour obtenir les résultats présentés dans cet article.

Une politique basée sur une distribution normale a permis au robot d'adapter automatiquement sa politique en réponse à un changement inattendu de l'environnement. Pour nos deux expériences sur le robot, nous avons observé que la valeur de l'écart type σ diminuait lorsque AC-S convergeait vers une politique, et augmentait en réponse à une nouvelle dynamique ou dynamique modifiée de l'environnement.

Un problème avec les méthodes présentées est qu'il est nécessaire de définir la valeur des paramètres des pas d'apprentissage. En pratique, la procédure suivante peut être utilisée pour les algorithmes AC et AC-S. Tout d'abord, pour l'estimation de la récompense moyenne, α_r doit être bas (0,001 par exemple). Puis, la règle classique consistant à utiliser 10% de la norme du vecteur de fonction $\mathbf{x}$ peut être utilisée pour définir

le pas d'apprentissage α_v du critique (tout en gardant le pas d'apprentissage de l'acteur α_u à 0). Une fois le critique stable et convergeant vers une estimation de la fonction de valeur, le paramètre α_u peut enfin être augmenté progressivement. Notons que la fonction de récompense et la norme du vecteur de fonctions doivent être bornées et normalisées. Un avantage supplémentaire à utiliser un codage en grille est d'obtenir un vecteur de fonctions satisfaisant ces deux critères.

Finalement, il est souvent nécessaire de contrôler plus d'un degrés de liberté. Bien que ce travail a été étendu à un problème avec un espace d'action à deux dimensions par Pilarski *et al.* (2011), nous ne pensons pas que ces algorithmes en tant que tel soient utilisables sur des problèmes possédant un espace d'action de grande dimension. Cependant, nous pensons que ces travaux constituent un premier pas vers une solution générique pour apprendre et adapter des politiques de contrôle dans des environnements complexes non stationnaires.

8 Conclusion

Dans cet article, nous avons étendu des algorithmes de gradient de la politique pour utiliser des traces d'éligibilité et une technique de mise à l'échelle du gradient. Nous avons ensuite testé ces extensions par une étude empirique, utilisant systématiquement des actions continues. Nous avons observé que l'introduction des traces d'éligibilité et la technique de mise à l'échelle du gradient amélioraient souvent la performance, alors que ce n'était pas le cas pour l'utilisation du gradient naturel. Finalement, cet article démontre que les algorithmes acteur-critique avec le codage par grille sont utilisables pour un apprentissage en ligne, en temps réel, avec des espaces d'état et d'action continus et qu'ils sont capables de s'adapter à des changements inattendus de l'environnement.

9 Remerciements

Ce travail a été financé par MPrime, Alberta Innovates Centre for Machine Learning, Glenrose Rehabilitation Hospital Fundation, Alberta Innovates—Technology Futures, NSERC, INRIA Bordeaux Sud-Ouest, et Westgrid, un partenaire de Compute Canada. Les auteurs remercient les relecteurs pour leurs commentaires utiles, Phillip Thomas pour des discussions informatives, et Mai Nguyen pour sa relecture de la version française.

Références

ABBEEL P., COATES A. & NG A. (2010). Autonomous helicopter aerobatics through apprenticeship learning. In *International Journal of Robotics Research*, volume 29, p. 1608–1639.

BAXTER J. & BARTLETT P. (2002). Direct gradient-based reinforcement learning. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 3, p. 271–274.

BENBRAHIM H., DOLEAC J., FRANKLIN J. & SELFRIDGE O. (1992). Real-time learning : A ball on a beam. In *International Joint Conference on Neural Networks*.

BHATNAGAR S., SUTTON R., GHAVAMZADEH M. & LEE M. (2009). Natural actor-critic algorithms. In *Automatica*, volume 45, p. 2471–2482.

BOWLING M. & VELOSO M. (2003). Simultaneous adversarial multi-robot learning. In *International Joint Conference on Artificial Intelligence*, volume 45, p. 2471–2482.

DOYA K. (2000). Reinforcement learning in continuous time and space. In *Neural computation*, volume 12, p. 219–245.

KIMURA H., YAMASHITA T. & KOBAYASHI S. (2002). Reinforcement learning of walking behavior for a four-legged robot. In *Proceedings of the 40th IEEE Conference on Decision and Control*, volume 1, p. 411–416.

KOHL N. & STONE P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of the International Conference on Robotics and Automation*, volume 3, p. 2619–2624.

PETERS J. & SCHAAL S. (2008). Natural actor-critic. In *Neurocomputing*, volume 71, p. 1180–1190.

PILARSKI P., DAWSON M., DEGRIS T., FAHIMI F., CAREY J. & R.S. S. (2011). Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. In *Proceeding of the IEEE International Conference on Rehabilitation Robotics*, p. 134–140.

SUTTON R. (1988). Learning to predict by the methods of temporal differences. In *Machine Learning*, volume 3, p. 9–44.

SUTTON R. & BARTO A. (1998). *Reinforcement Learning : an Introduction*. MIT press.

SUTTON R., KOOP A. & SILVER D. (2007). On the role of tracking in stationary environments. In *Proceedings of the 24th international conference on Machine learning*, p. 871–878.

SUTTON R., MCALLESTER D., SINGH S. & MANSOUR Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, p. 1057–1063.

SUTTON R. & WHITEHEAD S. (1993). Online learning with random representations. In *Proceedings of the Tenth International Conference on Machine Learning*, p. 314–321.

TEDRAKE R., ZHANG T. & SEUNG H. (2005). Stochastic policy gradient reinforcement learning on a simple 3d biped. In IEEE, Ed., *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, p. 2849–2854.

WILLIAMS R. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, volume 8, p. 229–256.