Méthodes d'apprentissage par renforcement pour espaces continus : les méthodes à gradient de politique et leurs améliorations (TRPO et PPO). v1.01

Franck Vandewiele, Samuel Delepoulle

▶ **To cite this version:**

Franck Vandewiele, Samuel Delepoulle. Méthodes d'apprentissage par renforcement pour espaces continus : les méthodes à gradient de politique et leurs améliorations (TRPO et PPO). v1.01. Laboratoire d'Informatique Signal et Image de la Côte d'Opale. 2023. hal-04115352

HAL Id: hal-04115352
https://hal.science/hal-04115352

Submitted on 2 Jun 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Public Domain

Méthodes d'apprentissage par renforcement pour espaces continus : les méthodes à gradient de politique et leurs améliorations (TRPO et PPO).

v1.01

Franck Vandewiele & Samuel Delepoulle
LISIC
Groupe projet Artisis – Artificial Intelligence for Image Synthesis

2 juin 2023

Table des matières

Chapitre 1

Introduction

1.1 Rappel RL

L'objectif de ce rapport est de démontrer l'utilisation de l'apprentissage par renforcement en apprentissage automatique dans le contexte de problèmes impliquant des facteurs continus. Les algorithmes d'apprentissage par renforcement (RL pour « reinforcement learning ») ont été initialement développés pour résoudre des problèmes relativement simples, où les états et les actions sont définis par des variables discrètes prenant un nombre limité de valeurs. L'extension aux espaces continus présente de nouveaux défis qui sont abordés dans ce rapport. La figure 1.1 présente succinctement la situation dans laquelle un agent est placé dans un environnement. Celui-ci choisit des *actions* et observe les transition d'*état* d'une part et un signal de son environnement appelé *récompense*.

Remarque : Ce rapport ne présente pas en détail le cadre de l'apprentissage par renforcement. Pour une compréhension approfondie de ce domaine, le lecteur est invité à se référer des ouvrages de référence tels Richard S. SUTTON et BARTO (2018) et à WIERING et van OTTERLO (2012)

1.2 RL pour problèmes continus

La formalisation du problème d'apprentissage par renforcement dans le cadre des chaînes de Markov a permis de développer une famille de méthodes basées sur l'estimation de tables de valeurs pour les états-actions, généralement appelées Q-values. Des algorithmes tels que le Q-Learning et SARSA utilisent cette approche pour apprendre les meilleures actions à entreprendre dans chaque état. Dans ces méthodes, l'apprentissage se fait souvent par différence temporelle (TD learning), technique qui estime la table des valeurs d'état-action

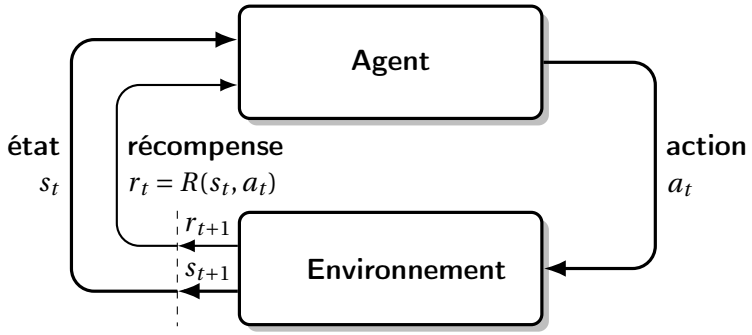

FIGURE 1.1 – Diagramme de principe de l'apprentissage par renforcement. Au temps t, dans l'état s_t, l'agent choisit une action a_t. Celle-ci produit une récompense r_{t+1} et produit une transition vers l'état s_{t+1}.

de façon itérative à partir des récompenses observées.

Cependant, ces algorithmes sont initialement conçus pour des espaces d'états et d'actions discrets. Lorsque les espaces sont continus, il est courant d'adapter les algorithmes TD en apprenant une fonction d'approximation, qui est utilisée pour estimer la fonction de valeur.

Les réseaux de neurones sont souvent choisis comme fonctions d'approximation dans l'apprentissage par renforcement continu. Ils sont considérés comme des approximateurs universels en raison de leur capacité à représenter et à apprendre des fonctions complexes (CYBENKO 1989). La structure en couches des réseaux de neurones permet une modélisation hiérarchique et la représentation de relations non linéaires entre les variables. Les fonctions d'activation non linéaires utilisées dans les réseaux de neurones, telles que la fonction sigmoïde, la fonction tanh ou la fonction ReLU, leur permettent de capturer des comportements complexes.

L'apprentissage par approximation de fonctions avec des réseaux de neurones dans le cadre de l'apprentissage par renforcement continu implique généralement l'utilisation de l'algorithme de descente de gradient pour ajuster les poids du réseau. Les réseaux de neurones sont entraînés à minimiser l'erreur entre les estimations de la fonction de valeur et les valeurs cibles calculées à l'aide de la méthode TD.

L'inconvénient principal de l'approche d'apprentissage par renforcement avec des réseaux de neurones comme fonctions d'approximation est la complexité et l'instabilité potentielle de l'apprentissage (MNIH et al. 2015).

Chapitre 2

Méthodes à gradient de politique

2.1 Principe des méthodes à gradient de politique

Comme nous l'avons vu précédemment, les approches de type Q-learning ne s'expriment pas naturellement lorsqu'un agent apprend dans un espace d'états ou d'actions continus. Les méthodes à gradient de politique permettent de résoudre cette difficulté, en travaillant directement dans l'espace des politiques. Une politique initiale étant paramétrée par un vecteur de paramètres θ, les méthodes à gradient de politique corrigent θ dans la direction de *retours* plus favorables afin d'obtenir une politique plus performante. Les approches Q-learning et à gradient de politique sont en cela totalement différentes tout en ayant le même but : quand Q-learning apprend une fonction de qualité dont découle indirectement une politique optimale, sans hypothèse d'une politique préalable, les méthodes à gradient de politique recherchent une politique maximisant une fonction objectif à partir d'une politique initiale en explorant l'espace des politiques.

Pour mettre sur pied une telle méthode, il convient donc :

— d'expliciter ce qu'est le retour d'une politique ;

— de déterminer, une politique paramétrée par θ étant donnée, dans quelle direction autour de θ ce retour est maximisé.

2.2 Cadre formel

Notations. Dans la suite, S désigne l'espace des états, A l'espace des actions, P le modèle de transition et R la fonction de récompense d'un processus de décision markovien $\mathcal{M} =< S, A, P, R >$. a désigne une action ; s un état. $P(s', a, s)$ représente la probabilité qu'un agent se trouve dans l'état s' après avoir entrepris l'action a depuis l'état s. On notera $P(s'|a, s) = P(s', a, s)$. R est une fonction de

$S \times A$, c'est-à-dire une récompense liée à l'action entreprise dans un état, sans tenir compte de l'état conséquent.

π_θ désigne une politique stochastique de $\mathscr{M}$ paramétrée par un vecteur $\theta \in \mathbb{R}^n$. C'est une fonction de $A \times S$ à valeurs dans $[0,1]$. $\pi_\theta(a,s)$ représente la probabilité que l'agent qui suit cette politique entreprenne l'action a lorsqu'il perçoit l'état s. On note $\pi_\theta(a|s) = \pi_\theta(a,s)$.

Dans la suite, on considérera que π_θ est différentiable en θ. En pratique, π_θ pourra être déterminé par un réseau de neurones dont θ est le vecteur de paramètres.

Definition 2.2.1 (trajectoire)**.** Une trajectoire τ est une suite d'états-actions :

$$\tau = (s_0, a_0, \dots s_H, a_H, s_{H+1})$$

où H est un horizon, soit entier, soit infini.

Definition 2.2.2 (récompense d'une trajectoire)**.** La récompense d'une trajectoire τ est définie par :

$$R(\tau) = \sum_{t=0}^{H} R(s_t, a_t)$$

Remarque. On peut adjoindre à la définition de la récompense un facteur d'actualisation γ, avec $0 < \gamma < 1$:

$$R_\gamma(\tau) = \sum_{t=0}^{H} \gamma^t R(s_t, a_t)$$

L'introduction d'un facteur d'actualisation permet de diminuer le poids relatif des récompenses éloignées. Lorsque H est infini, un facteur d'actualisation permet de garantir la convergence de la somme R.

Dans la suite, s'il n'y a pas d'ambiguïté, on notera $R(\tau)$ plutôt que $R_\gamma(\tau)$.

Definition 2.2.3 (utilité d'une politique)**.** L'utilité d'une politique π_θ est la récompense espérée par un agent suivant cette politique. Notée $J(\theta)$, elle est définie comme suit :

$$J(\theta) = \mathbb{E}_{\pi_\theta} R(\tau)$$

Dans le cas où H est fini, J s'exprime comme suit :

$$J(\theta) = \sum_{\tau} \pi_\theta(\tau) R(\tau)$$

avec :

$$\pi_\theta(\tau) = \prod_{t=0}^{H} P(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t)$$

par application immédiate de la propriété de Markov de $\mathscr{M}$.

Definition 2.2.4 (fonctions de valeur)**.** La fonction de valeur d'état sous politique π_θ, notée $V_{\pi_\theta}(s)$, est définie par :

$$V_{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}\left(R(\tau)|s_0 = s\right)$$

La fonction de valeur d'action sous politique π_θ, notée $Q_{\pi_\theta}(s,a)$, est définie par :

$$Q_{\pi_\theta}(s,a) = \mathbb{E}_{\pi_\theta}\left(R(\tau)|s_0 = s, a_0 = a\right)$$

La fonction d'avantage sous politique π_θ, notée A_{π_θ}, est définie par :

$$A_{\pi_\theta}(s,a) = Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s)$$

2.3 Gradient de la fonction d'utilité

Les méthodes à gradient de politique reposent sur la maximisation de la fonction d'utilité via une technique de montée de gradient sur J.

Lorsque H est fini, le gradient de J s'exprime comme suit :

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_\tau \pi_\theta(\tau) R(\tau)$$
$$\nabla_\theta J(\theta) = \sum_\tau R(\tau) \nabla_\theta \pi_\theta(\tau)$$

En multipliant artificiellement chaque quantité sommée par $\frac{\pi_\theta(\tau)}{\pi_\theta(\tau)}$, il est possible de faire apparaître une intéressante transformation de cette écriture :

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_\tau \pi_\theta(\tau) R(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} \\ &= \sum_\tau \pi_\theta(\tau) R(\tau) \nabla_\theta \log \pi_\theta(\tau) \\ \nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left(R(\tau) \nabla_\theta \log \pi_\theta(\tau)\right)\end{aligned}$$

Cette identité est connue sur le nom de *théorème du gradient de politique*. Elle reste vraie dans le cas où H est infini et qu'un facteur d'actualisation est intégré au calcul de la récompense (Richard S SUTTON et al. 1999).

Théorème du gradient de politique 2.3.1.

$$\nabla_\theta \mathbb{E}_{\pi_\theta} R(\tau) = \mathbb{E}_{\pi_\theta}\left(R(\tau) \nabla_\theta \log \pi_\theta(\tau)\right)$$

Le gradient de la fonction d'utilité peut ainsi s'exprimer comme une espérance, ce qui le rend accessible à une estimation par méthode de Monte-Carlo. En échantillonnant des trajectoires selon la politique π_θ, l'estimation du gradient de la fonction d'utilité devient :

$$\nabla_\theta J(\theta) \simeq \hat{g} = \frac{1}{m} \sum_{i=1}^{m} R(\tau^{(i)}) \nabla_\theta \log \pi_\theta(\tau^{(i)})$$

Cette estimation n'est cependant pas exploitable directement : $\nabla_\theta \log \pi_\theta(\tau^{(i)})$ est une quantité qui dépend a priori du modèle de transition P, c'est à-dire directement de l'environnement. En pratique, ce modèle de l'environnement est difficile à caractériser.

Un examen plus approfondi nous permet cependant de nous en affranchir :

$$\begin{aligned}
\nabla_\theta \log \pi_\theta(\tau^{(i)}) &= \nabla_\theta \log \left(\prod_{t=0}^{H} P(s_{t+1}^{(i)} | s_t^{(i)}, a_t^{(i)}) \pi_\theta(a_t^{(i)} | s_t^{(i)}) \right) \\
&= \nabla_\theta \left(\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, a_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \right)
\end{aligned}$$

Dans cette expression, la quantité $\log P(s_{t+1}^{(i)} | s_t^{(i)}, a_t^{(i)})$ ne dépend pas de θ et sa contribution au gradient est donc nulle. On a ainsi :

$$\begin{aligned}
\nabla_\theta \log \pi_\theta(\tau^{(i)}) &= \nabla_\theta \left(\cancel{\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, a_t^{(i)})} + \sum_{t=0}^{H} \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \right) \\
&= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \\
\nabla_\theta \log \pi_\theta(\tau^{(i)}) &= \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})
\end{aligned}$$

Le gradient de la fonction objectif s'estime finalement comme suit :

$$\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \left(R(\tau^{(i)}) \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \right)$$

Cette estimation du gradient ne dépend que de la récompense des trajectoires. Elle sert ainsi de base à une famille d'algorithmes d'apprentissage dits *model-free* en ce sens qu'ils n'essayent pas d'estimer le modèle de transition P.

Remarque. Le théorème du gradient de politique peut être formulé de différentes façons. En plus de la version que nous avons démontrée, il en existe

d'autres, qui utilisent les fonctions de valeur. On en trouve une synthèse dans SCHULMAN et al. (2016), parmi lesquelles :

$$\nabla_\theta \mathbb{E}_{\pi_\theta} R(\tau) = \mathbb{E}_{\pi_\theta} \left(R(\tau) \nabla_\theta \log \pi_\theta(\tau) \right)$$

$$\nabla_\theta \mathbb{E}_{\pi_\theta} R(\tau) = \mathbb{E}_{\pi_\theta} \sum_{t=0}^{\infty} \left(Q_{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(\tau) \right)$$

$$\nabla_\theta \mathbb{E}_{\pi_\theta} R(\tau) = \mathbb{E}_{\pi_\theta} \sum_{t=0}^{\infty} \left(A_{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(\tau) \right)$$

2.4 REINFORCE

2.4.1 Principe et caractéristiques de REINFORCE

REINFORCE (WILLIAMS 1992) est l'algorithme d'apprentissage par gradient de politique historique. Il repose directement sur une optimisation de politique par montée de gradient obtenu par méthode de Monte-Carlo.

Initialiser θ aléatoirement
foreach $\tau = (s_0, a_0, \dots s_H, a_H, s_{H+1}) \sim \pi_\theta$ **do**
  for $t \leftarrow 0$ **to** H **do**
    $\theta \leftarrow \theta + \alpha R(\tau) \nabla_\theta \pi_\theta(s_t, a_t)$
  end
end

Algorithme 1 : REINFORCE (WILLIAMS 1992) est paramétré par un pas de descente de gradient α.

REINFORCE donne des résultats intéressants pour des problèmes continus que d'autres approches comme Q-learning ont des difficultés à aborder. Il souffre néanmoins de certains défauts :

variance importante L'échantillonnage des trajectoires est intrinsèquement sujette à une variance importante, ce qui ralentit la convergence de l'algorithme.

contrôle du pas de gradient Comme toutes les méthodes basées sur le gradient, le processus de convergence de REINFORCE est sensible à l'amplitude du pas fait à chaque itération dans la direction du gradient. Ce pas est déterminant dans le processus de convergence : choisi trop petit, la convergence de l'algorithme sera très lente; choisi trop grand, l'algorithme risque de ne pas converger en faisant des sauts trop importants.

2.4.2 Baseline

Dans l'expression du théorème du gradient de politique 2.3.1, les contributions au gradient, qui sont sujettes à une variance importante, sont pondérées par $R(\tau)$, récompense dont la dispersion peut être grande, aggravant d'autant le problème. Dès l'introduction de REINFORCE, une première approche est proposée pour amortir cette variance : corriger la récompense $R(\tau)$ par une valeur de base (*baseline*), éventuellement variable sans dépendre de θ, dont l'espérance serait nulle, et dont l'introduction aurait un effet positif sur la dispersion des contributions au gradient. L'article de WILLIAMS (1992) laisse cependant la question du choix d'une baseline largement ouverte.

Initialiser θ aléatoirement
Choisir une baseline $b(s)$
foreach $\tau = (s_0, a_0, \dots s_H, a_H, s_{H+1}) \sim \pi_\theta$ **do**
  for $t \leftarrow 0$ **to** H **do**
    $\theta \leftarrow \theta + \alpha(R(\tau) - b(s_t))\nabla_\theta \pi_\theta(s_t, a_t)$
  end
end

Algorithme 2 : REINFORCE avec baseline (WILLIAMS 1992). b est une baseline qui peut dépendre de l'état observé ; b ne dépend en revanche pas du paramètre θ.

2.5 Méthodes acteur-critique

La grande variance de REINFORCE est notamment liée à l'estimation du gradient à partir de la récompense de trajectoires individuelles. Cette approche épisodique nuit à la convergence de cet algorithme.

Comme nous l'avons vu, le théorème 2.3.1, qui quantifie le gradient de politique comme une fonction du retour, peut être reformulé en utilisant des fonctions de valeur. Les fonctions de valeur quantifient le comportement d'une politique de façon plus globale et sont moins sujettes à une variance élevée que les approches épisodiques reposant sur l'évaluation de récompenses de trajectoires individuelles (CHADI et MOUSANNIF 2023). La fonction d'avantage $A_{\pi_\theta}(s, a)$, en particulier, quantifie la contribution d'une action à la récompense espérée des trajectoires produites sous une politique donnée. C'est un indicateur à la fois plus précis et plus stable que la récompense accordée à une trajectoire donnée.

Une estimation du gradient de politique à partir d'une fonction de valeur est susceptible d'être plus stable qu'une approche épisodique. Calculer une fonc-

tion de valeur n'est cependant pas immédiat et nécessite son propre processus d'estimation (SCHULMAN et al. 2016). Les approches qui procèdent à ce type d'évaluation en parallèle du calcul d'une politique ont été développées dans Richard S SUTTON et al. (1999). Elles sont connues sous le nom de méthodes acteur-critique.

Les méthodes acteur-critique sont des algorithmes qui procèdent en parallèle à deux types de traitements :

— l'acteur est chargé de produire le traitement attendu en sortie, c'est-à-dire une estimation du paramètre θ d'une politique réalisant un maximum local ;
— le critique est chargé de produire un traitement permettant d'évaluer et de guider l'acteur en estimant une fonction de valeur.

Dans le cadre de l'apprentissage profond, acteur et critique sont des réseaux de neurones ayant chacun leur propre vecteur de paramètres.

L'approche acteur-critique est devenue une pratique standard pour les algorithmes à politique de gradient.

L'algorithme 3 est une version minimale des algorithmes acteur-critique à gradient de politique, connue sous le nom générique de *Vanilla Policy Gradient*[1]. Cet algorithme est donné à titre d'illustration et n'a pas la prétention d'être performant. Nous avons fait le choix de la simplicité. Nous y estimons la fonction de valeur d'état et la fonction d'avantage par différence temporelle (Richard S. SUTTON et BARTO 2018). Des estimations plus efficaces et non biaisées peuvent également être utilisées (SCHULMAN et al. 2016).

La plupart des algorithmes à gradient de politique développés ces dernières années sont basés sur la même structure.

1. Cette dénomination a été largement adoptée a posteriori par la communauté ; elle n'est pas du fait de Sutton.

Input : γ, facteur d'actualisation
Input : α, taux d'apprentissage du critique
Input : β taux d'apprentissage de l'acteur
Initialiser les paramètres de l'acteur θ aléatoirement
Initialiser les paramètres du critique ω aléatoirement
for $k \leftarrow 0$ **to** N **do**
  choisir un état initial s
  for $t \leftarrow 0$ **to** H **do**
    choisir une action a en suivant la politique π_θ
    entreprendre l'action a
    recevoir la récompense r et observer l'état suivant s'
    `/* Mettre à jour le critique */`
    $\omega \leftarrow \omega + \alpha * (r + \gamma V_\omega(s') - V_\omega(s)) * \nabla_\omega V_\omega(s)$
    `/* Estimer l'avantage */`
    $A \leftarrow r + \gamma V_\omega(s') - V_\omega(s)$
    `/* Mettre à jour l'acteur */`
    $\theta \leftarrow \theta + \beta * A * \nabla_\theta \log(\pi_\theta(a|s))$
    $s \leftarrow s'$
  end
end
Output : politique π_θ

Algorithme 3 : Vanilla Policy Gradient, avec estimation des fonctions de valeur d'état et d'avantage par différence temporelle.

Chapitre 3

Méthodes basées sur des régions de confiance

3.1 Problème des méthodes à gradient de politique

Dans une approche linéaire de la recherche d'optimum, les méthodes de descente de gradient sont confrontées à deux difficultés :

— choisi trop petit, le pas d'itération conduit à une convergence extrêmement lente ;

— choisi trop grand, le pas d'itération ne permet pas de converger vers un optimum (*overshoot*.)

Cette tension existe pour toutes les méthodes d'optimisation basées sur le gradient ; elle est plus aiguë pour les méthodes à gradient de politique, qui apprennent à partir des trajectoires générées en utilisant la politique apprise. Si la politique apprise vient à s'éloigner d'un optimum, la qualité des données d'apprentissage va se dégrader, ce qui risque de compromettre le processus d'apprentissage (WAGNER 2014).

3.2 Trust Region Policy Optimization

Pour répondre à ces difficultés, TRPO (SCHULMAN et al. 2015) maximise la fonction objectif $J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$ en imposant la contrainte de la non-décroissance de $J(\theta)$. Parce que les politiques apprises s'amélioreront à chaque pas d'itération, les problèmes liés à l'overshoot sont éliminés.

Pour cela, l'algorithme maximise une fonction objectif de substitution, qui a les propriétés suivantes :

— elle coïncide localement avec la fonction objectif ;
— elle minore la fonction objectif ;
— elle permet une montée de gradient plus facile à contrôler que sur la fonction objectif.

L'objectif de substitution est ainsi en quelque sorte une fonction objectif *pessimiste*, dont le comportement est illustré à la figure 3.1. Sa non décroissance est garantie par une contrainte sur la divergence de Kullback-Leibler de la politique apprise au rang précédent par rapport à celle apprise au rang suivant, qui doit rester petite.

Lors d'un pas d'itération mettant à jour le paramètre θ_{old}, l'algorithme recherche ainsi le paramètre θ maximisant le minorant :

$$L(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right]$$

en respectant la contrainte :

$$\hat{\mathbb{E}}_t \left[\text{KL} \left[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t) \right] \right] \leq \delta$$

TRPO peut être ainsi interprété comme une variante des méthodes d'optimisation reposant sur des régions de confiance (NOCEDAL et WRIGHT 2006) : la contrainte sur la divergence de Kullback-Leibler apparaît comme une caractérisation d'une région de confiance garantissant que la politique apprise s'améliore.

Bien qu'élégant sur le plan théorique, le problème d'optimisation sous contrainte que TRPO essaye de résoudre est difficile à implémenter directement. Dans sa version pratique, il nécessite de nombreux compromis. Ses performances, bonnes, le sont au prix d'une importante consommation de ressources.

3.3 Amélioration de TRPO : Proximal Policy Optimization (PPO)

PPO reprend les fondements théoriques de TRPO en contrôlant l'objectif de substitution de façon plus simple. TRPO définit son objectif de substitution comme suit :

$$L(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right]$$

où $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$.

À cet objectif de substitution est adjoint une contrainte qui permet d'éviter des ajustements de politiques trop importants.

SCHULMAN et al. (2017) proposent deux versions de l'algorithme PPO. Toutes deux reposent sur une approximation de la contrainte théorique de TRPO :

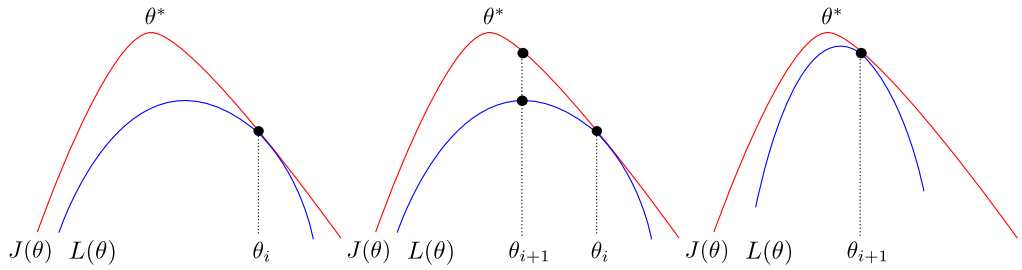

FIGURE 3.1 – Minoration de J par L. À chaque itération, θ_{i+1} est déterminé à partir de θ_i en procédant à une montée de gradient sur une fonction objectif secondaire L, qui est inférieure à la fonction objectif J. La fonction L dépend de i. Si θ^* réalise un extremum local de L, θ^* maximise localement J et π_{θ^*} est localement optimale.

PPO-clip remplace la contrainte par une correction de la fonction d'objectif de substitution, introduisant un objectif *tronqué* sous la forme [1] :

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, g\left(\varepsilon, \hat{A}_t\right) \right) \right]$$

où g est défini comme :

$$g(\varepsilon, A) = \begin{cases} (1+\varepsilon) A & \text{si } A \geq 0 \\ (1-\varepsilon) A & \text{si } A < 0 \end{cases}$$

L'introduction de l'hyperparamètre ε permet d'atténuer l'attrait des politiques qui s'éloigneraient trop de la politique initiale, ce qui va dans le sens de mises à jour de proximité.

PPO KL-penalty intègre la contrainte comme une pénalité sur la fonction objectif. L'objectif de substitution devient :

$$L^{\text{KLPEN}}(\theta) = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t - \beta \text{KL}\left(\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right) \right]$$

Après une mise à jour de politique, le paramètre β est éventuellement ajusté pour que la divergence de Kullback-Leibler reste proche d'une divergence cible d_{targ}. Il est proposé que si $\hat{\mathbb{E}}_t \left[\text{KL}\left(\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right) \right]$ est trop petit par rapport à d_{targ}, β soit contracté par un facteur multiplicateur inférieur à 1 ; si au contraire la divergence constatée est trop grande, β est dilaté.

Cette approche prolonge l'idée de régions de confiance, méthodes dans lesquelles ce type de mise à jour de paramètre est classique.

1. Nous donnons ici la formulation de l'algorithme telle qu'envisagée par ACHIAM (2018), dont les notations nous semblent plus claires que celles de l'article original.

3.4 Algorithme PPO SCHULMAN et al. (2017)

SCHULMAN et al. (2017) décrivent l'algorithme PPO dans le cadre d'un acteur critique (algorithme 4).

Cet algorithme comporte deux boucles imbriquées. Pour chaque itération, la politique $\pi_{\theta_{\text{old}}}$ est exécutée dans l'environnement pendant un certain nombres de pas T et les avantages $\hat{A}_1, \ldots, \hat{A}_T$ sont estimés pour chaque acteur. Ensuite, la fonction objectif L est optimisée par rapport à θ, en utilisant un certain nombre d'époques K et une taille de « mini-batch » $M \leq NT$. Enfin, la valeur de θ est mise à jour en tant que θ_{old} pour la prochaine itération.

for *iteration* $= 1, 2, \ldots$ **do**
  for *actor* $= 1, 2, \ldots, N$ **do**
    Exécuter la politique $\pi_{\theta_{\text{old}}}$ dans l'environnement pour T pas de temps
    Calculer les avantages estimés $\hat{A}_1, \ldots, \hat{A}_T$
  end
  Optimiser la fonction d'objectif secondaire L par rapport θ, en utilisant K époques et une taille de mini-batch $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end

Algorithme 4 : Algorithme PPO en version Acteur-Critique, proposé par SCHULMAN et al. (2017)

Chapitre 4

Résultats

4.1 Résultats sur des benchmarks

PPO a été testé sur une large gamme de problèmes de l'environnement Gym [1], ce qui permet de confronter les algorithmes à des situations standardisées et de les évaluer de manière comparative. Des tâches telles que le pendule inversé, l'atterrisseur lunaire, Pong, CartPole et bien d'autres ont été utilisées pour évaluer les performances de PPO.

Dans ces différentes tâches, PPO a régulièrement démontré des performances élevées et stables. Il a réussi à apprendre des politiques efficaces qui surpassent souvent celles des autres algorithmes. Par exemple, dans le cas du pendule inversé, PPO a réussi à maintenir l'équilibre du pendule en utilisant rapidement les actions optimales, ce qui a conduit à une performance supérieure par rapport aux autres algorithmes.

Cette capacité à obtenir des performances élevées et stables, ainsi que la capacité d'amélioration continue, font de PPO un algorithme de choix parmi les algorithmes d'apprentissage par renforcement dans l'environnement Gym. Les résultats obtenus ont contribué à l'adoption et à l'utilisation répandue de PPO dans le domaine de l'apprentissage par renforcement, en particulier pour des tâches où la stabilité et la performance sont des critères importants.

Dans leur article, SCHULMAN et al. (2017) comparent PPO-clip avec d'autres algorithmes considérés comme effectif pour des problèmes continus :

— TRPO décrit précédemment SCHULMAN et al. (2015) (se référer à la section 3.2) ;

— la méthode cross-entropy (CEM pour *Cross-Entropy Method*) (SZITA et LÖRINCZ 2006) ;

1. https://www.gymlibrary.dev/environments/classic_control/

— l'algorithme *Vanilla Policy Gradient* décrit également précédemment (algorithme 3) ;
— l'algorithme A2C (*Advantage Actor-Critic*) (MNIH et al. 2015) ;
— l'algorithme A2C avec région de confiance (*A2C with Trust region*)

Les auteurs notent de meilleures performances de PPO sur quasiment tous les problèmes de contrôle continu de l'environnement MuJoCo.

4.2 Autres résultats

Depuis 2017, OpenAI développe le programme *OpenAI Five* spécifiquement conçu pour jouer au jeu vidéo Dota 2 OPENAI et al. (2019). Ce choix s'explique par les nombreux défis qu'il présente pour les systèmes d'intelligence artificielle :

Complexité le jeu se caractérise par un vaste ensemble d'actions et de mécanismes, offrant des milliers de décisions possibles à chaque instant. Il existe plusieurs stratégies et tactiques permettant de remporter les parties.

Horizons temporels longs les décisions prises à un moment donné peuvent avoir des conséquences positives bien après. Les bots sont exécutés à une fréquence de 30 images par seconde sur des parties d'une durée de 45 minutes, ce qui représente 80 000 « ticks » par partie. Cela dépasse de loin le nombre de coups observés dans une partie d'échecs ou de Go.

États partiellement observables contrairement à certains jeux où tous les joueurs ont accès à toutes les informations à tout moment, les joueurs de Dota 2 disposent d'informations très incomplètes. À un instant donné, un joueur ne connaît qu'une petite partie de la carte.

Espaces d'états continus Dota 2 présente des espaces d'action et d'observation de grande dimension. Le jeu se déroule sur une carte étendue comprenant dix héros, plusieurs bâtiments, des unités non-joueuses et de nombreuses fonctionnalités telles que les runes, les arbres et les wards. *OpenAI Five* observe environ 16 000 valeurs au total à chaque étape de temps, principalement des nombres à virgule flottante et des valeurs catégoriques offrant des centaines de possibilités.

Vastes espaces d'action En moyenne, le modèle choisit parmi 8 000 à 80 000 actions à chaque instant, principalement des variables discrètes avec six états possibles.

PPO a par ailleurs été utilisé récemment pour de nombreuses applications dans des domaines aussi variés que la régulation électrique SONG et al. (2023), les véhicules autonomes SHAO et al. (2023) ou la robotique ZHAO et al. (2023).

Références

ACHIAM, J. (2018). OpenAI Spinning Up Proximal Policy Optimization.

CHADI, M.-A. & MOUSANNIF, H. (2023). Understanding Reinforcement Learning Algorithms : The Progress from Basic Q-learning to Proximal Policy Optimization.

CYBENKO, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, *2*(4), 303-314. https://doi.org/10.1007/bf02551274

HUANG, S., DOSSA, R. F. J., RAFFIN, A., KANERVISTO, A. & WANG, W. (2022). The 37 Implementation Details of Proximal Policy Optimization. *ICLR Blog Track*. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/

MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M., FIDJELAND, A. K., OSTROVSKI, G., PETERSEN, S., BEATTIE, C., SADIK, A., ANTONOGLOU, I., KING, H., KUMARAN, D., WIERSTRA, D., LEGG, S. & HASSABIS, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529-533. https://doi.org/10.1038/nature14236

NOCEDAL, J. & WRIGHT, S. (2006). *Numerical Optimization*. Springer New York.

OPENAI, : BERNER, C., BROCKMAN, G., CHAN, B., CHEUNG, V., DĘBIAK, P., DENNISON, C., FARHI, D., FISCHER, Q., HASHME, S., HESSE, C., JÓZEFOWICZ, R., GRAY, S., OLSSON, C., PACHOCKI, J., PETROV, M., d. O. PINTO, H. P., RAIMAN, J., ... ZHANG, S. (2019). Dota 2 with Large Scale Deep Reinforcement Learning.

SCHULMAN, J., LEVINE, S., ABBEEL, P., JORDAN, M. & MORITZ, P. (2015). Trust Region Policy Optimization. *Proceedings of the 32nd International Conference on Machine Learning*, *37*, 1889-1897. https://proceedings.mlr.press/v37/schulman15.html

SCHULMAN, J., MORITZ, P., LEVINE, S., JORDAN, M. I. & ABBEEL, P. (2016). High-Dimensional Continuous Control Using Generalized Advantage Estimation. In Y. BENGIO & Y. LECUN (Éd.), *4th International Conference on*

Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. http://arxiv.org/abs/1506.02438

SCHULMAN, J., WOLSKI, F., DHARIWAL, P., RADFORD, A. & KLIMOV, O. (2017). Proximal Policy Optimization Algorithms. https://doi.org/10.48550/ARXIV.1707.06347

SHAO, C., CHENG, F., XIAO, J. & ZHANG, K. (2023). Vehicular intelligent collaborative intersection driving decision algorithm in Internet of Vehicles. *Future Gener. Comput. Syst.*, *145*, 384-395.

SONG, S., JUNG, Y., JANG, G. & JUNG, S. (2023). Proximal policy optimization through a deep reinforcement learning framework for remedial action schemes of VSC-HVDC. *Int. J. Electr. Power Energy Syst.*, *150*(109117), 109117.

SUTTON, R. S. [Richard S.] & BARTO, A. G. (2018). *Reinforcement Learning : An Introduction* (Second). The MIT Press. http://incompleteideas.net/book/the-book-2nd.html

SUTTON, R. S. [Richard S], MCALLESTER, D., SINGH, S. & MANSOUR, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. In S. SOLLA, T. LEEN & K. MÜLLER (Éd.), *Advances in Neural Information Processing Systems*. MIT Press. https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf

SZITA, I. & LÖRINCZ, A. (2006). Learning Tetris Using the Noisy Cross-Entropy Method. *Neural Computation*, *18*(12), 2936-2941. https://doi.org/10.1162/neco.2006.18.12.2936

WAGNER, P. (2014). Policy oscillation is overshooting. *Neural Networks*, *52*, 43-61. https://doi.org/10.1016/j.neunet.2014.01.002

WIERING, M. & van OTTERLO, M. (Éd.). (2012). *Reinforcement learning* (2012^{e} éd.). Springer.

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

ZHAO, L.-Y., CHANG, T.-Q., ZHANG, J., ZHANG, L., CHU, K.-X., GUO, L.-B. & KONG, D.-P. (2023). A policy optimization algorithm based on sample adaptive reuse and dual-clipping for robotic action control. *Appl. Soft Comput.*, *134*(109967), 109967.

Annexe : autres ressources

— Version officielle (OpenAI) : https://github.com/openai/baselines
— PPO for Beginners : https://github.com/ericyangyu/PPO-for-Beginners
— Version détaillée : HUANG et al. 2022
— Démonstrations mathématiques et codage de nombreux algorithmes (dont REINFORCE , TRPO et PPO...) :
https://lilianweng.github.io/posts/2018-04-08-policy-gradient/
— Foundations of Deep RL — lecture series de Pieter Abbeel :
https://youtu.be/2GwBez0D20A