



RAPPORT DE PROJET

P-ANDROIDE - Projet ANDROIDE

Étude de l'apprentissage par renforcement profond sur Pendulum

MASTER D'INFORMATIQUE SPÉCIALITÉ ANDROIDE

PREMIÈRE ANNÉE

ANNÉE UNIVERSITAIRE 2020 - 2021

PROFESSEUR :
ÉTUDIANTS :

OLIVIER SIGAUD
VINCENT FU
YUHAO LIU

Table des matières

1. État de l'art	2
1.1 Contexte	2
1.2 Problématique	3
1.3 Descriptions de Pendulum et des librairies	3
1.4 Algorithmes en apprentissage par renforcement	4
1.4.1 <i>Policy Gradient</i>	5
1.4.2 <i>Soft Actor-Critic</i>	6
2. Étude expérimentale	9
2.1 Étude par ablation des algorithmes	9
2.2 Résultats et analyses des performances	13
3. Conclusion	21
4. Annexes et Références	23

1. État de l'art

1.1 Contexte

Les algorithmes d'apprentissage par renforcement permettent à un agent autonome (robot, logiciel) de résoudre des problèmes complexes tels que gagner à un jeu vidéo, ramasser des balles de tennis, trouver la sortie d'un labyrinthe ou encore faire marcher une créature artificielle sur des terrains variés.

Pour représenter ces problèmes, l'apprentissage par renforcement utilise le modèle mathématique des processus de décision markovien dans lequel un agent interagit avec un environnement en effectuant une suite d'actions. processus de décision markovien est défini par un quadruplet $\{S, A, T, R\}$ avec S un ensemble d'états décrivant l'environnement perçu par l'agent, A un ensemble d'actions, T une fonction de transition décrivant l'effet des actions sur l'environnement et R une fonction de récompense associée aux transitions.

Par exemple, dans un environnement de labyrinthe, les états correspondent aux positions de l'agent dans lesquels l'agent ne peut qu'effectuer des actions de déplacement qui sont vers l'avant, vers l'arrière, à sa gauche ou à sa droite afin de trouver la sortie. Plus l'agent se rapproche de la sortie, plus il sera récompensé. Selon le problème étudié, les actions effectuées par l'agent peuvent être déterministes ou stochastiques.

La résolution de ces problèmes revient donc à trouver la meilleure suite d'actions possible pour l'agent par rapport à l'environnement étudié, on parle alors de politique optimale. En effet, chaque action de l'agent dénote une transition d'état accompagnée d'une valeur de récompense donnée par l'environnement décrivant la qualité de cette transition. La politique optimale est donc associée à une fonction indiquant quelle action effectuer à chaque état amenant à une agrégation maximale des récompenses.

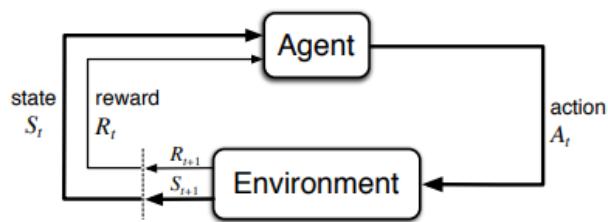


FIGURE 1 – Représentation générale des processus de décision markovien

Cependant, il arrive parfois que le nombre d'états et actions possibles soient très grands et même infinis. Dans l'exemple du labyrinthe, le nombre d'actions devient infini si on autorise les déplacements angulaires. Dans ce cas, les algorithmes d'apprentissage par renforcement vont utiliser en plus du modèle de processus de décision markovien un réseau de neurones pour décrire leur politique. L'apprentissage par renforcement est alors dit profond.

Dans la littérature, pour mesurer les performances des différents algorithmes, les algorithmes d'apprentissage par renforcement profond sont souvent appliqués en simulation à

des problèmes de contrôle de mouvements complexes tels que les environnements Ant, Half-Cheetah, Hopper ou encore Humanoid. Cependant, il reste beaucoup à apprendre de ces algorithmes sur des problèmes de contrôle classiques comme les environnements CartPole, MountainCar ou encore Pendulum.

Ainsi, notre projet a pour but d'en apprendre plus sur ces algorithmes dans le cadre des problèmes de contrôle classiques et notamment sur l'environnement Pendulum.

1.2 Problématique

Sachant que les valeurs des récompenses obtenues par les algorithmes définissent leurs performances, des précédents travaux ont montré que les algorithmes standards de gradient sur les politiques (*Policy Gradient*) n'atteignent pas une performance satisfaisante sur l'environnement Pendulum, alors que les algorithmes d'apprentissage par renforcement profond issus de l'état de l'art comme l'algorithme *Deep Deterministic Policy Gradient* (Lillicrap et al., 2016) ou encore l'algorithme *Soft Actor-Critic* (Haarnoja et al., 2018) y parviennent. Or, les algorithmes d'apprentissage par renforcement profond cités précédemment sont tous dérivés des algorithmes de *Policy Gradient* standard.

L'objectif du projet est alors de comprendre pourquoi les algorithmes d'apprentissage par renforcement arrivent à obtenir des bonnes performances sur Pendulum, quels sont finalement les composants supplémentaires dans ces algorithmes amenant à des bonnes performances.

1.3 Descriptions de Pendulum et des librairies

L'environnement Pendulum est un problème dans lequel on souhaite qu'une machine immobile tenant une pendule à tige rigide à masse arrive à équilibrer sa tige vers la position verticale haute en la balançant et à y rester le plus longtemps possible sous contrainte de gravité (voir FIGURE 2)¹.

Par ailleurs dans l'environnement Pendulum dans lequel un épisode dure 200 pas de temps, chaque action effectuée apporte toujours une récompense négative comprise entre $-16,2736044$ et 0 . Plus précisément, la récompense est définie par :

$$r(\theta, F) = - \left(\theta^2 + 0,1\dot{\theta}^2 + 0,001F^2 \right) \quad (1)$$

où θ correspond à l'angle du pendule par rapport au repère, $\dot{\theta} \in [-8,0 ; 8,0]$ la vitesse angulaire et $F \in [-2,0 ; 2,0]$ la force exercée. $(\theta, \dot{\theta})$ correspond alors à un état et F une action possible du modèle.

Évidemment, l'action de ne pas balancer lorsque le pendule est à l'équilibre en position verticale haute et donc de ne rien faire est la seule action rapportant une récompense de 0 .

Ainsi sachant qu'une politique optimale est associée à une agrégation maximale des récompenses, plus la politique optimale obtenue par un algorithme possède une agrégation des

1. Veuillez utiliser Adobe Reader pour observer l'animation.

récompenses proche de 0, plus l'algorithme est performant.

FIGURE 2 – Exemple d'exécution d'un épisode de Pendulum

Pour étudier les algorithmes qui nous intéressent et répondre à la problématique du projet, nous avons à disposition la librairie *Basic Policy Gradient Labs* de notre professeur sur github contenant l'implémentation de l'algorithme standard de *Policy Gradient* ainsi que la librairie *Stable Baselines 3* d'Antonin Raffin également sur github contenant toutes les implémentations des algorithmes d'apprentissage par renforcement profond issus de l'état de l'art.

1.4 Algorithmes en apprentissage par renforcement

Le but des algorithmes d'apprentissage par renforcement est de trouver la politique optimale permettant de maximiser la récompense. Cette récompense est une fonction qu'on nomme également le *return* qui correspond à la somme des récompenses des transitions données par la trajectoire. Autrement dit, on souhaite dans le cas d'une politique déterministe optimiser la formule suivante :

$$\pi^* = \operatorname{argmax}_{\pi} R(\tau_{\pi}(s_0))$$

où π est une politique, R la fonction de récompense et τ_{π} une trajectoire obtenue selon la politique π et un état initial s_0 .

Cependant, la plupart des modèles étudiés en apprentissage par renforcement ont une politique stochastique. Dans ce cas, pour une seule politique, il est possible d'obtenir plusieurs trajectoires. L'optimisation revient donc à maximiser l'espérance des récompenses :

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E}_{\tau \sim \pi} [R(\tau)]$$

où la trajectoire τ est obtenue selon un échantillonnage de la politique π et d'un état initial.

En outre, il arrive parfois que la politique soit une fonction paramétrée par une valeur θ décrivant le contrôleur de l'agent. La politique est alors entièrement définie par le contrôleur θ et l'optimisation devient :

$$\theta^* = \operatorname{argmax}_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \operatorname{argmax}_{\theta} J(\theta) \quad (2)$$

où θ est un contrôleur et $J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)]$ la fonction de coût (ou d'utilité).

1.4.1 Policy Gradient

L'algorithme *Policy Gradient* consiste à optimiser la politique en utilisant la méthode de descente de gradient directement sur la politique. En optimisant de cette manière, chaque pas de gradient permet de réduire les probabilités de sélectionner les mauvaises actions tandis que les probabilités de prendre les bonnes actions augmentent jusqu'à convergence. La formule du gradient est la suivante :

$$\begin{aligned} \nabla_{\theta} J(\theta) &= \frac{1}{m} \sum_{i=1}^m \sum_{t=1}^H R(\tau^{(i)}) \nabla_{\theta} \log \pi_{\theta}(a_t^{(i)} | s_t^{(i)}) \\ &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=1}^H R(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right] \end{aligned} \quad (3)$$

où $m \in \mathbb{N}^*$ correspond au nombre de trajectoires possibles selon π_{θ} , $H \in \mathbb{N}^* \cup \{\infty\}$ la longueur d'une trajectoire, $s_t^{(i)}$ et $a_t^{(i)}$ respectivement les états et actions au temps t de la trajectoire $\tau^{(i)}$ décrite par $\tau^{(i)} = (s_1^{(i)}, a_1^{(i)}, s_2^{(i)}, a_2^{(i)}, s_3^{(i)}, \dots, s_{H+1}^{(i)})$.

Cependant, l'algorithme de *Policy Gradient* standard souffre beaucoup de grande variance en raison d'un nombre élevé de trajectoires possibles. Pour contourner cela, on propose dans la littérature de remplacer la fonction de *return* R par différentes estimations Ψ_t ne dépendant plus entièrement de la trajectoire mais que d'une partie. La formule générale du gradient est alors :

$$\nabla_{\theta} J(\theta) = \mathbb{E} \left[\sum_{t=1}^H \Psi_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right] \quad (4)$$

où l'espérance suit la dynamique de la politique π_{θ} , c'est-à-dire que les trajectoires sont échantillonées selon les distributions issues de π_{θ} .

Voici les estimations possibles pour Ψ_t en posant $r_t = r(s_t, a_t, s_{t+1})$ la récompense de la transition de s_t à s_{t+1} en faisant l'action a_t et $\gamma \in]0, 1]$:

$\sum_{t=0}^H r_t$	Récompense totale d'une trajectoire.
$\sum_{t=0}^H \gamma^t r_t$	Récompense totale avec coefficient d'actualisation.
$\sum_{t'=t}^H \gamma^{t'-t} r_{t'}$	Somme des récompenses futures avec coefficient d'actualisation.
$\sum_{t'=t}^H \gamma^{t'-t} r_{t'} - b(s_t)$	Somme des récompenses futures avec coefficient d'actualisation et <i>baseline</i> b .
$Q^\pi(s_t, a_t)$	Fonction valeur d'état-action $Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1:\infty}, a_{t+1:\infty}} \left[\sum_{l=0}^H r_{t+l} \right]$
$r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$	Erreur de différence temporelle avec fonction valeur d'état $V^\pi(s_t) = \mathbb{E}_{s_{t+1:\infty}, a_{t:\infty}} \left[\sum_{l=0}^H r_{t+l} \right]$
$A^\pi(s_t, a_t)$	Fonction avantage $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$

Les algorithmes de *Policy Gradient* standard optimisent ainsi d'une manière dite de Monte Carlo. À chaque pas de gradient, une nouvelle politique est créée permettant de générer une famille de trajectoires. Avec cette nouvelle famille de trajectoires, est issue un gradient afin de générer à nouveau une nouvelle politique qui n'a aucun lien avec l'ancienne politique. Cependant, certains algorithmes d'apprentissage par renforcement profond optimisent d'une manière différente dite de *Bootstrap* en utilisant une architecture *Actor-Critic*.

1.4.2 Soft Actor-Critic

Les algorithmes d'*Actor-Critic* contrairement aux algorithmes de *Policy Gradient* standard ne cherchent pas à générer une toute nouvelle politique à chaque pas d'optimisation mais plutôt à générer une politique issue de l'amélioration de l'ancienne politique par le biais d'un indicateur disant si une telle politique amène à des récompenses correctes et quels sont les ajustements à effectuer pour obtenir des meilleures récompenses. La nouvelle politique générée possédera alors des caractéristiques de l'ancienne politique.

Plus précisément, l'acteur correspond à la politique et qui décide quelles sont les actions à sélectionner en donnant les probabilités des actions à chaque état et le *critic* est celui qui va dire à l'acteur à quel point il est intéressant de sélectionner ces actions en utilisant nécessairement comme indicateur l'erreur de différence temporelle.

Bien que la distinction entre *Policy Gradient* et *Actor-Critic* reste difficile à appréhender

der, la différence principale qui permet de distinguer d'un algorithme de *Policy Gradient* à un algorithme d'*Actor-Critic* est que dans *Policy Gradient*, l'estimation Ψ_t est uniquement issue d'une trajectoire tandis que dans *Actor-Critic*, l'estimation Ψ_t devient un *critic* s'il est à la fois estimé avec une trajectoire et le précédent *critic*. On parle alors de *Bootstrapping*.

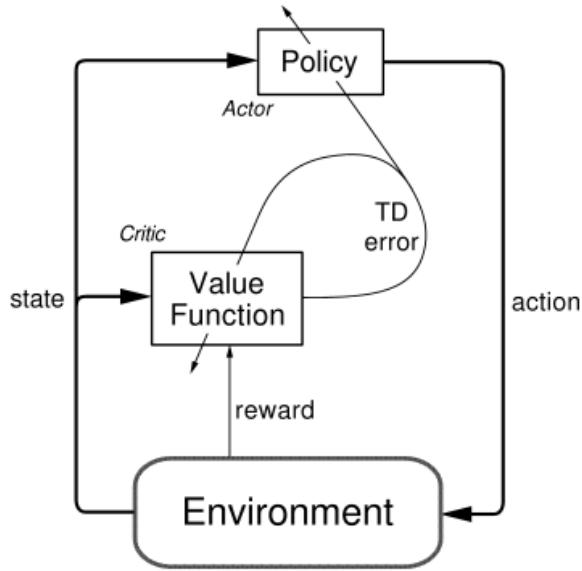


FIGURE 3 – Représentation générale d'une architecture *Actor-Critic*

En plus de ce qu'il a été cité ci-dessus, l'algorithme *Soft Actor-Critic* intègre une entropie dans la formule d'optimisation afin de favoriser l'exploration de l'agent dans l'espace des politiques.

En effet, l'entropie est une mesure quantifiant l'imprédictibilité d'une distribution. Autrement dit, plus une distribution est uniforme, plus l'entropie sera élevée. Plus une distribution est proche d'une distribution déterministe, plus l'entropie sera faible. Dans le cadre d'une entropie² sur une politique, elle est définie par la formule suivante :

$$\mathcal{H}(\pi(\cdot|s_t)) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t)} [-\log \pi(a_t|s_t)]. \quad (5)$$

L'optimisation consiste alors à maximiser non seulement les récompenses mais en plus l'entropie :

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \mathbb{E}_{(s_t, a_t) \sim \rho_{\pi}} \left[\sum_t \gamma^t (r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))) \right] \quad (6)$$

où ρ_{π} correspond à la distribution des trajectoires induite par π et $\alpha \geq 0$ le coefficient d'entropie contrôlant l'importance relative de l'entropie dans l'optimisation.

2. La formule générale est : $H(P) = \mathbb{E}_{x \sim P} [-\log P(x)]$ où P est une distribution.

Ainsi, l'algorithme *Soft Actor-Critic* est capable d'apprendre une politique stochastique tout en ne négligeant pas l'exploration des politiques. En effet, dans le cas d'une optimisation classique comme *Policy Gradient*, chaque pas d'optimisation réduit l'espace de recherche des politiques en vue d'une convergence. *Soft Actor-Critic* quant à lui converge vers différentes solutions en explorant un espace des politiques plus large.

En pratique, pour trouver la politique optimale dans le cas d'un apprentissage par renforcement profond, un réseau de neurones représentant la politique est à optimiser. Dans le cas de *Policy Gradient*, l'optimisation concerne le contrôleur représentant les poids du réseau et la mise à jour par descente de gradient s'effectue par une propagation des erreurs. Dans un modèle d'*Actor-Critic*, deux réseaux sont utilisés : un réseau pour l'acteur ainsi qu'un réseau pour le *critic*. En effet, pour que l'architecture fonctionne, chaque pas d'optimisation nécessite une mise à jour de l'acteur ainsi qu'une mise à jour du critique jusqu'à convergence.

La mise à jour du critique s'effectue en minimisant la formule suivante :

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(s_t, a_t) - \hat{Q}(s_t, a_t) \right)^2 \right] \quad (7)$$

où Q_θ correspond à la fonction *state-action value* paramétrée par θ , \mathcal{D} le *replay buffer* et

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p} [V_{\bar{\psi}}(s_{t+1})] \quad (8)$$

correspondant au fonction *soft state-action value* avec la fonction *soft state value* du *target network* $V_{\bar{\psi}}$:

$$V_{\bar{\psi}}(s_t) = \mathbb{E}_{a_t \sim \pi} [Q_{\bar{\psi}}(s_t, a_t) - \log \pi(a_t | s_t)] . \quad (9)$$

Malheureusement, la mise à jour du critique est instable et peut parfois diverger. En effet, sachant que la valeur de convergence du critique est liée à la fonction *value* du critique, chaque mise à jour du critique va altérer cette valeur de convergence. Pour contourner cela, on propose dans la littérature d'utiliser une copie périodique du critique qu'on nomme le *target network*. Par exemple, on effectue une copie des valeurs du critique tous les x pas de temps et ensuite, les x mises à jour suivantes sont calculées par rapport à cette copie et ainsi de suite.

En utilisant le *target network*, les mises à jour du critique pourront se faire non plus par rapport aux précédentes valeurs du critique mais par rapport à des valeurs de base déterminées périodiquement.

Le second problème en apprentissage par renforcement est la forte corrélation entre les couples (état, action) successifs obtenus pendant l'apprentissage. Or, de manière générale en apprentissage, les données doivent être indépendamment et identiquement distribuées. On propose alors dans la littérature d'utiliser un *replay buffer* \mathcal{D} qui a pour but de collecter les

échantillons de données (s_t, a_t, s_{t+1}, r_t) des différentes trajectoires avant et pendant l'apprentissage. L'apprentissage s'effectuera alors en pratique sur un *minibatch* contenant une partie des données du *replay buffer* sélectionnées aléatoirement permettant de contrer la corrélation.

Enfin, la mise à jour de l'acteur s'effectue en minimisant l'espérance d'une mesure de dissimilarité entre la politique et l'estimation issue du critique :

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[D_{\text{KL}} \left(\pi_\psi(\cdot | s_t) \middle\| \frac{\exp Q_\theta(s_t, \cdot)}{Z_\theta(s_t)} \right) \right] \quad (10)$$

où Z_θ correspond à une fonction de normalisation de distribution.

2. Étude expérimentale

Pour trouver les raisons des bonnes performances des algorithmes d'apprentissage par renforcement profond par rapport à la méthode Policy Gradient sur Pendulum, une étude par ablation entre l'algorithme *Soft Actor-Critic* et l'algorithme *Policy Gradient* a été envisagée.

2.1 Étude par ablation des algorithmes

Une étude par ablation consiste à évaluer systématiquement les performances de tous les algorithmes intermédiaires issus d'un débranchement ou d'une altération un à un des composants de l'algorithme initial.

Pour mieux comprendre les composants susceptibles d'améliorer les performances des algorithmes, voici une liste des différents composants des algorithmes :

<i>Policy Gradient</i>	<ul style="list-style-type: none"> — Descente de gradient sur la politique (*) — Différentes estimations possibles Ψ_t
<i>Soft Actor-Critic</i>	<ul style="list-style-type: none"> — Architecture <i>Actor-Critic</i> (*) — Intégration d'une entropie — Utilisation d'un <i>replay buffer</i> (*) — Utilisation d'un <i>target network</i> — Utilisation d'un <i>minibatch</i>

Pour que les algorithmes soient exécutables, les caractéristiques (*) sont inhérents aux algorithmes.

Afin d'avoir une idée sur l'écart de performance entre *Policy Gradient* et *Soft Actor-Critic*, voici les courbes de récompenses des algorithmes en fonction du pas de temps d'apprentissage.

Voici tout d'abord la liste des hyper-paramètres étudiés :

<i>learning rate</i>	Coefficient de descente de gradient contrôlant la vitesse d'apprentissage.
γ	Coefficient d'actualisation contrôlant le poids des récompenses récentes et futures.
$ \mathcal{D} $	Taille du <i>replay buffer</i> stockant les données les plus récentes.
α	Coefficient d'entropie contrôlant l'influence de l'entropie dans l'optimisation.
<i>minibatch size</i>	Taille du <i>minibatch</i> stockant les données sélectionnées aléatoirement pour l'apprentissage.
<i>learning starts</i>	Pas de temps où l'algorithme démarre l'apprentissage.
<i>target update interval</i>	Fréquence de mise à jour du <i>target network</i> .
<i>policy type</i>	Type de distribution utilisé pour la politique.
<i>n-step</i>	Nombre de récompenses consécutives prises en compte dans le calcul de la somme des récompenses dans le <i>return</i> à partir de l'état s_t .
<i>max value gradient clipping</i>	Borne maximale de coupure de la valeur du gradient.

Il y a encore d'autres hyper-paramètres que les algorithmes utilisent mais qui sont plus d'ordre pratique et ne sont pas essentiel dans notre étude sur les algorithmes.

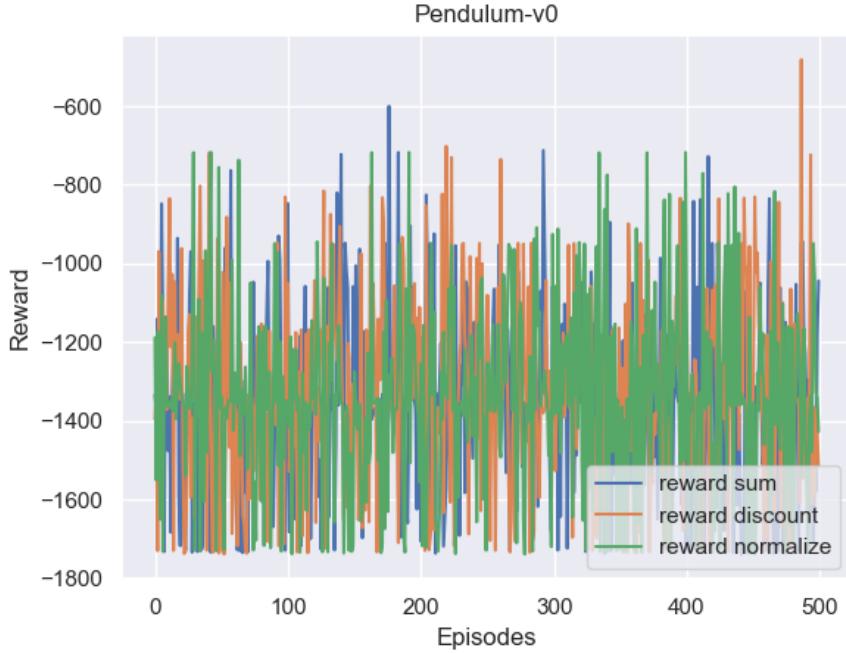


FIGURE 4 – Courbes des différents rewards de Policy Gradient sur Pendulum

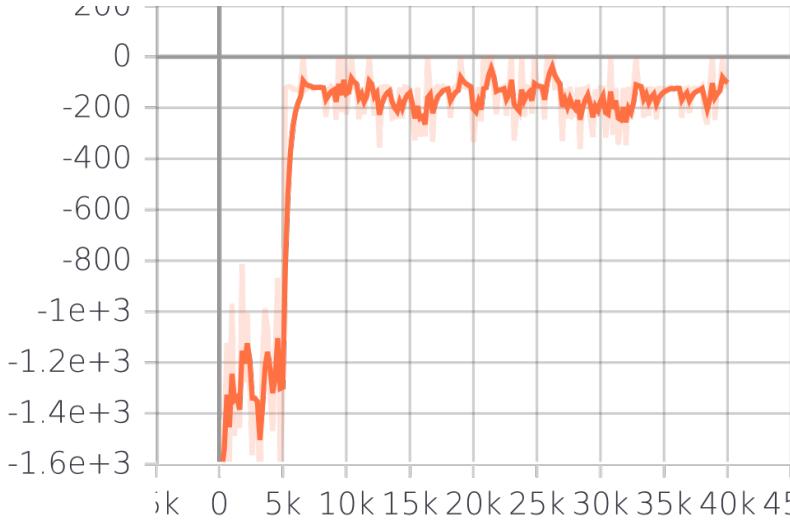


FIGURE 5 – Courbe de reward de Soft Actor-Critic sur Pendulum

En observant les performances de *Policy Gradient* sur Pendulum (FIGURE 4), la récompense varie entre environ -1700 et -800 sans convergence explicite et avec beaucoup de bruits. En effet, les bruits sont liés à la fonction de récompense de Pendulum. La performance moyenne sur 900 épisodes est d'environ -1313 traduisant ainsi une mauvaise performance de *Policy Gradient*.

Hyper-paramètres :

$\gamma = 0,99$
 $learning rate = 1 \cdot 10^{-2}$
 $policy type : squashed gaussian$

Hyper-paramètres :

$|\mathcal{D}| = 10^6$
 $\alpha = 0,2$
 $\gamma = 0,99$
 $minibatch size = 256$
 $learning starts = 5 \cdot 10^{-4}$
 $learning rate = 7 \cdot 10^{-4}$
 $target update interval = 1$
 $policy type : log probability$

Au contraire, les performances de *Soft Actor-Critic* sur Pendulum (FIGURE 5) avec les hyper-paramètres ci-dessus et les autres hyper-paramètres en pratique par défaut semblent bien meilleures puisqu'il y a convergence vers une récompense d'environ -158 après les 5000 pas de temps. En effet, l'agent va d'abord stocker les trajectoires dans le *replay buffer* avant de lancer l'apprentissage. La convergence est alors atteinte après environ 7000 pas de temps soit seulement 2000 pas de temps d'apprentissage pour apprendre correctement Pendulum.

Cela amène à deux hypothèses possibles expliquant cette bonne performance : soit il existe un composant critique dans *Soft Actor-Critic* amenant à une bonne performance sur Pendulum, soit il existe plusieurs composants critiques et c'est la conjonction de ces composants qui permet d'apprendre correctement l'environnement Pendulum.

Pour vérifier ces hypothèses, voici les manipulations effectuées sur *Soft Actor-Critic* :

- Modification de la sélection aléatoire des données pour le *minibatch* en une sélection contiguë.
- Modification de l'influence de l'entropie voire suppression l'entropie.
- Modification de la taille du *replay buffer*.
- Modification de la taille du *minibatch*.
- Suppression du *target network*.

2.2 Résultats et analyses des performances

Pour obtenir des résultats fiables, les résultats ci-dessus sont calculés à partir d'une moyenne de 100 récompenses à chaque pas de temps et avec les hyper-paramètres par défaut sauf indication contraire.

Vérifions à présent l'influence de sélection des données pour le *minibatch* ainsi que l'influence de l'entropie :

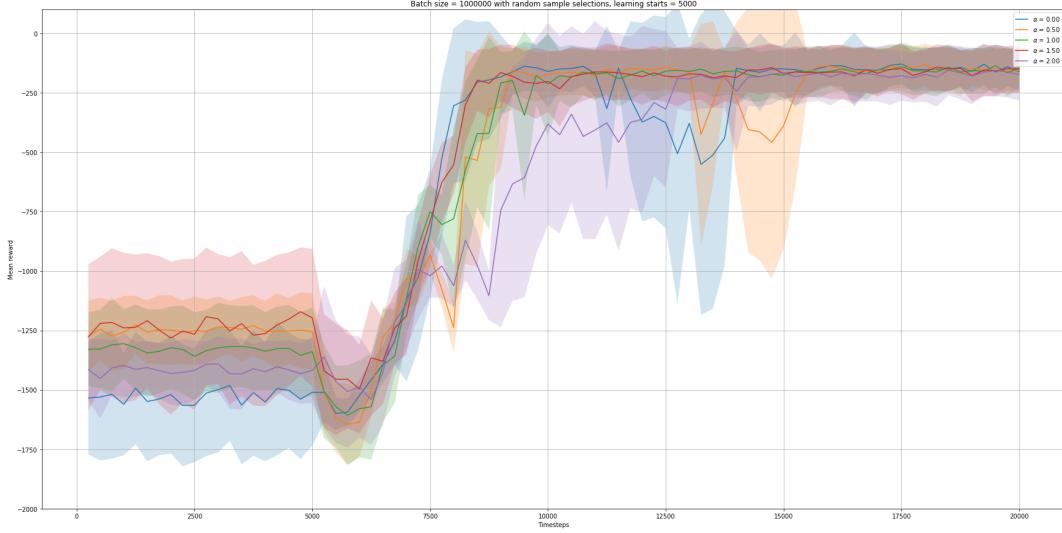


FIGURE 6 – Courbes des différents rewards de Soft Actor-Critic sur Pendulum avec sélection aléatoire des données pour le minibatch

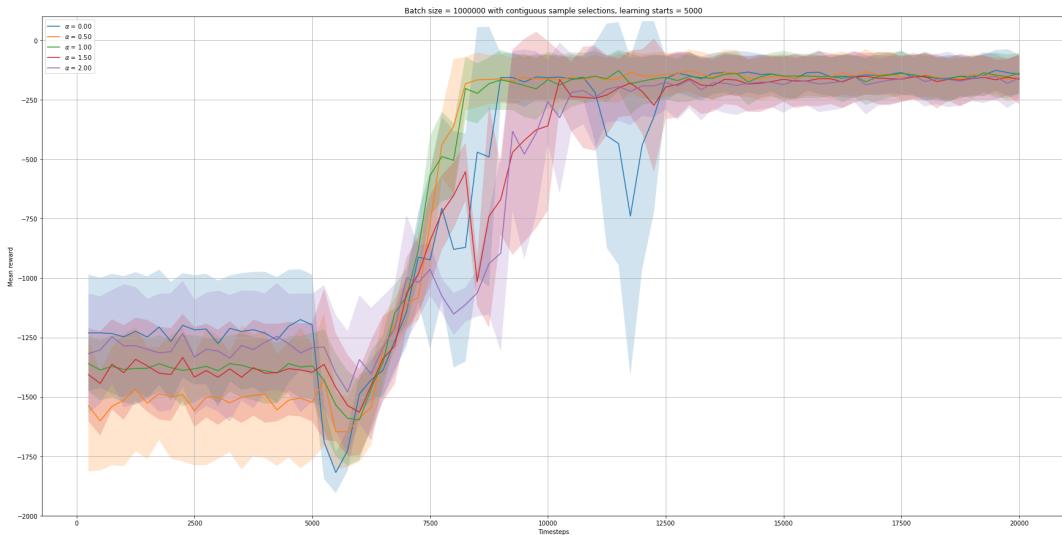


FIGURE 7 – Courbes des différents rewards de Soft Actor-Critic sur Pendulum avec sélection contiguë des données pour le minibatch

Sur les deux figures, les courbes et les convergences sont quasiment identiques : la sé-

lection aléatoire des données pour le *minibatch* n'est donc pas un facteur critique au bonne performance de *Soft Actor-Critic*. De plus pour $\alpha = 0$, l'algorithme arrive toute même à obtenir la même convergence que les autres courbes ayant $\alpha \neq 0$.

Cela signifie que l'entropie n'est également pas un facteur critique pour l'apprentissage de Pendulum et qu'un algorithme utilisant une architecture *Actor-Critic* avec les mêmes astuces de *replay buffer* et de *target network* est capable aussi d'apprendre Pendulum.

Vérifions à présent l'influence de la taille du *replay buffer*.

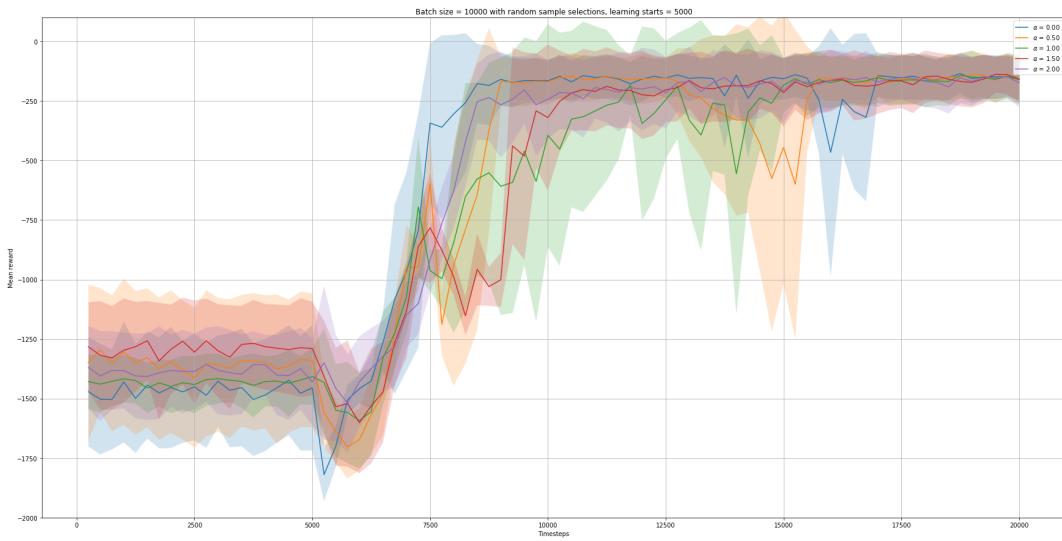


FIGURE 8 – Courbes des différents rewards de *Soft Actor-Critic* sur Pendulum avec une taille de *replay buffer* de 10^4

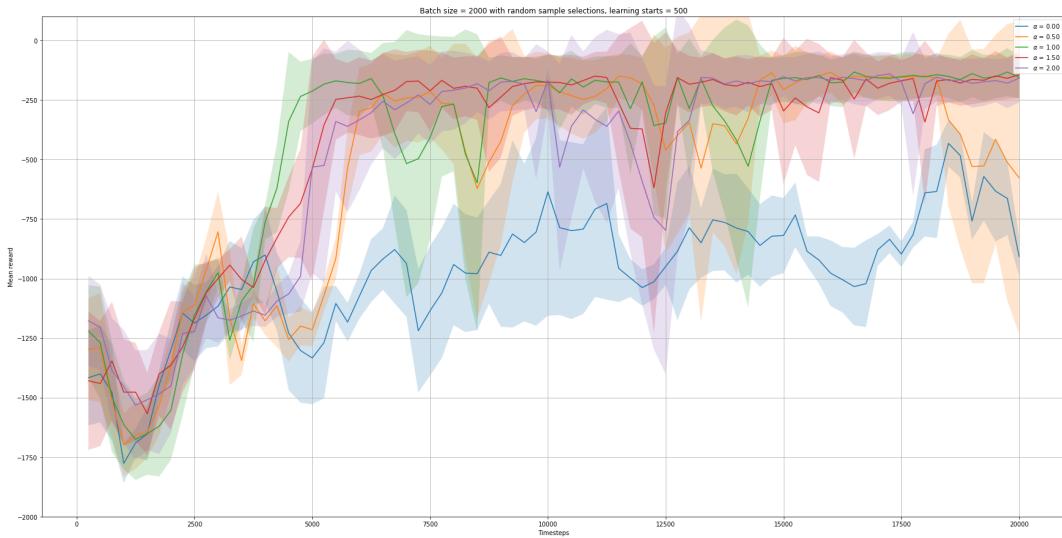


FIGURE 9 – Courbes des différents rewards de *Soft Actor-Critic* sur Pendulum avec une taille de *replay buffer* de $2 \cdot 10^3$ avec un learning starts de $5 \cdot 10^2$

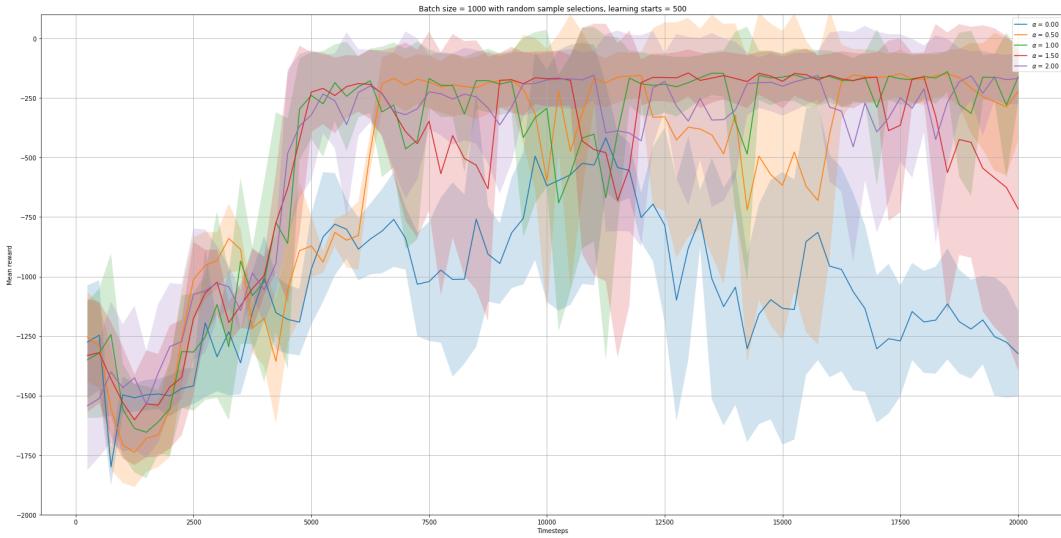


FIGURE 10 – Courbes des différents rewards de Soft Actor-Critic sur Pendulum avec une taille de replay buffer de 10^3 avec un learning starts de $5 \cdot 10^2$

Pour un *replay buffer* de taille 10^4 (FIGURE 8), l'algorithme est toujours capable d'apprendre. Ce n'est que lorsque la taille du *replay buffer* est inférieure à 10^4 qu'il y a une nuance à observer. Sachant qu'un *replay buffer* de taille 10^6 est considéré comme normal, lorsque la taille du *replay buffer* devient critique, les performances semblent être affectées par un écart-type plus élevé mais arrivent toute de même à converger.

Cependant, lorsque l'entropie est absente ($\alpha = 0$), l'algorithme n'arrive plus à converger. Cela montre bien le rôle de l'entropie dans l'exploration de l'espace des politiques. En effet, si la taille du *replay buffer* est réduite, cela limite considérablement la diversité des données pour la sélection pour le *minibatch* et un des moyens d'avoir des données plus ou moins différentes en très peu de quantité, c'est l'utilisation de l'entropie.

Ainsi, la taille du *replay buffer* peut être un facteur critique à la bonne performance de l'algorithme. Cependant, les performances liées à la taille du *replay buffer* lorsqu'elle devient très petite peuvent être compensées par l'utilisation de l'entropie.

Vérifions à présent l'influence de la taille du *minibatch*, le nombre de données utilisées pour l'apprentissage à chaque pas de gradient.

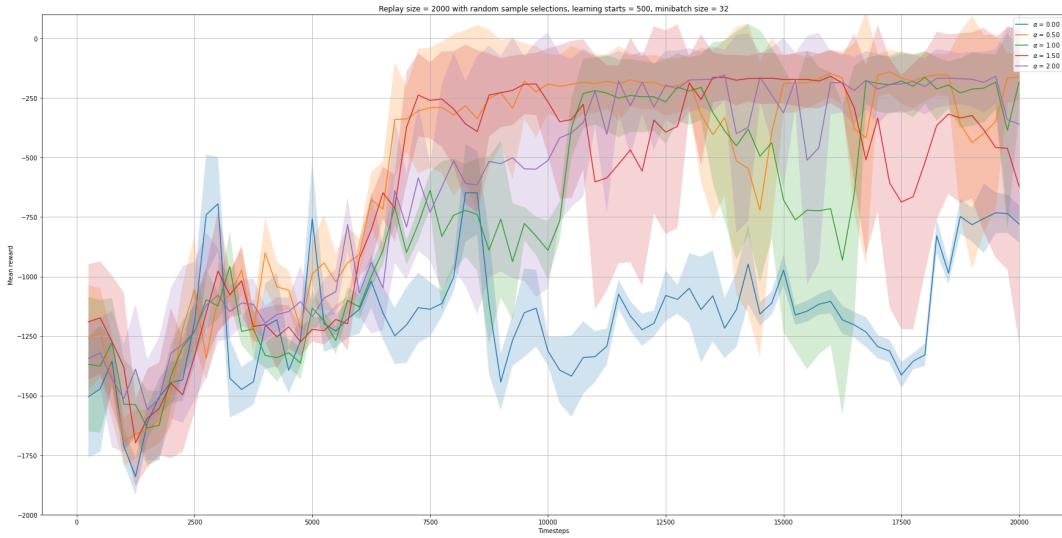


FIGURE 11 – Courbes des différents rewards de Soft Actor-Critic sur Pendulum avec une taille de replay buffer de $2 \cdot 10^3$ et avec un minibatch size de 32

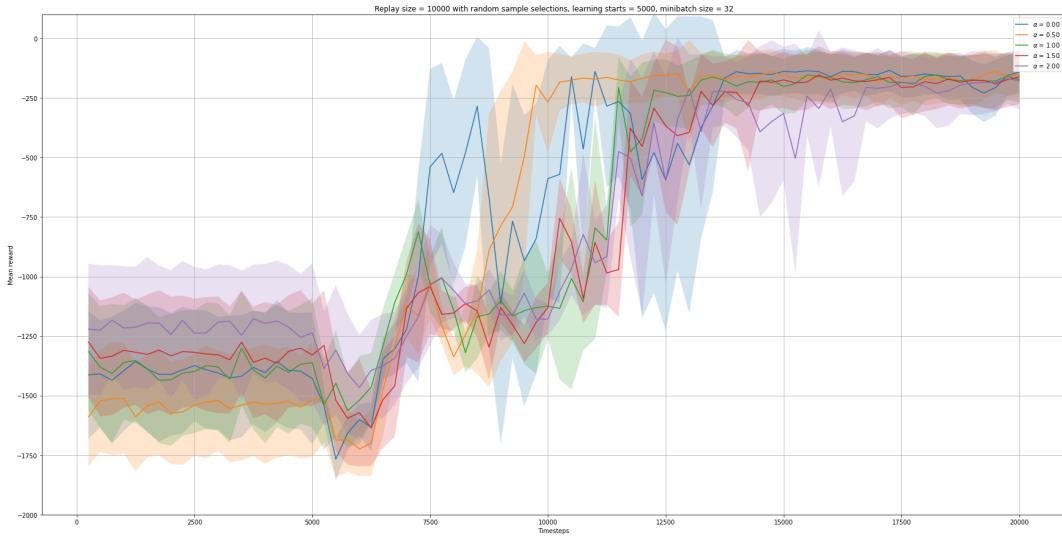


FIGURE 12 – Courbes des différents rewards de Soft Actor-Critic sur Pendulum avec une taille de replay buffer de 10^4 avec un minibatch size de 32

Pour un *replay buffer* de taille $2 \cdot 10^3$ avec un *minibatch* de taille 32 (FIGURE 11), on retrouve la même conclusion que pour le cas d'étude sur le *replay buffer* à différence près que l'écart-type au niveau de la valeur de convergence est un peu plus grande.

Cela s'explique par le fait qu'on utilise beaucoup de moins données pour le *minibatch* à chaque pas d'apprentissage et donc parmi les 100 récompenses à chaque pas de temps, on peut potentiellement obtenir plus de récompenses faibles.

Ainsi, lorsque la taille du *minibatch* devient critique (pas en dessous de 32), l'algorithme est toujours capable d'apprendre pour $\alpha \neq 0$ mais avec une convergence un peu moins stable.

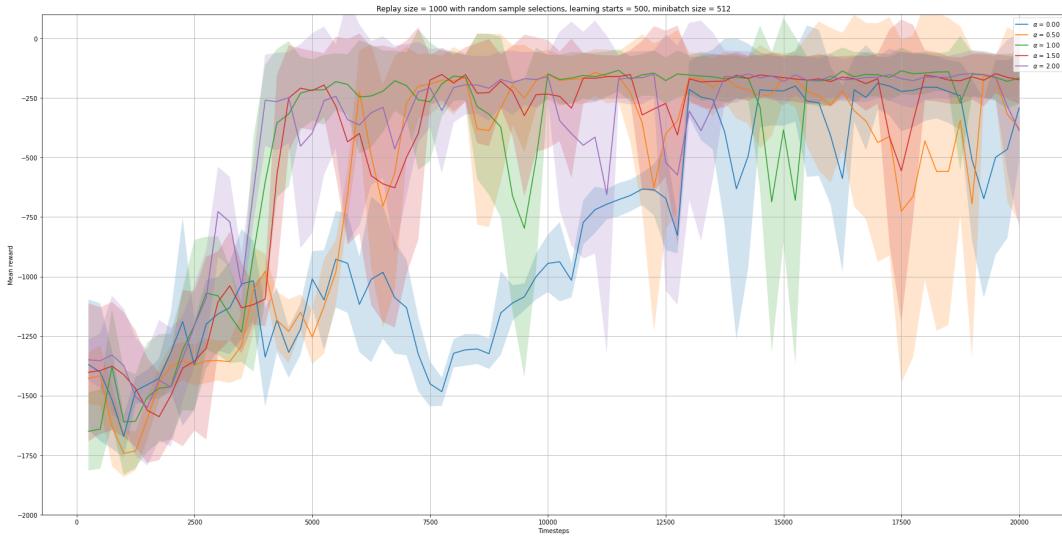


FIGURE 13 – Courbes des différents rewards de Soft Actor-Critic sur Pendulum avec une taille de replay buffer de 10^3 avec un minibatch size de 512

Cependant, pour un *replay buffer* de taille 10^3 avec un *minibatch* de taille 512 (FIGURE 13), l'algorithme est capable d'apprendre même sans entropie alors que ce n'était pas le cas pour un *minibatch* de taille 256.

Ainsi, lorsque la taille du *replay buffer* devient critique et qu'il n'y a pas d'entropie, l'apprentissage peut être suffisamment performant à condition d'utiliser un *minibatch* assez grand.

On en déduit ainsi une forme d'interdépendance entre : la capacité mémoire de l'agent, la diversité des expériences ainsi que la taille d'apprentissage. En effet, la taille du *replay buffer* défini concrètement la capacité mémoire de l'agent. Plus son *replay buffer* est grand, plus l'agent est capable d'avoir en mémoire des expériences passées lointaines.

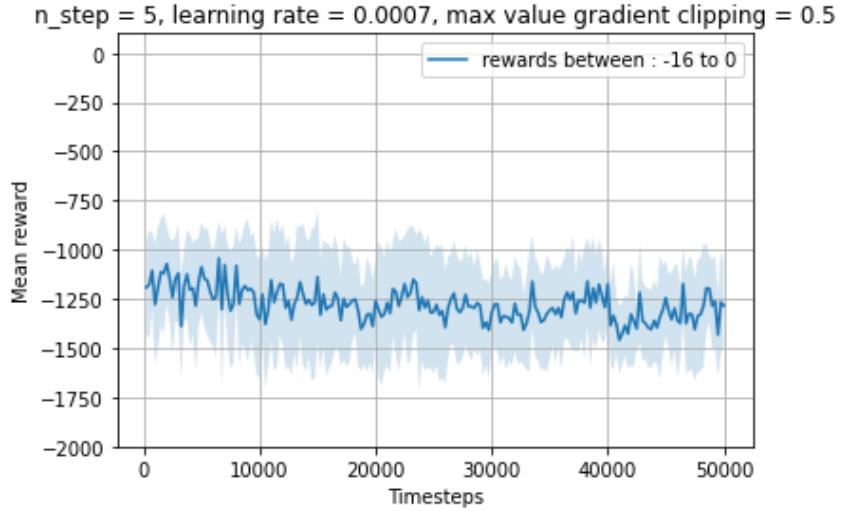
Tandis que la taille d'apprentissage est justement définie par la taille du *minibatch* contrôlant l'influence de la mémoire de l'agent dans l'apprentissage. Plus précisément, plus la taille du *minibatch* est grande, plus l'agent va utiliser une proportion mémoire grande dans l'apprentissage.

En résumé, voici les conclusions qu'on a vu jusqu'à présent :

- Si la taille du *replay buffer* est suffisamment grande, alors peu importe l'influence de l'entropie voire sans entropie, l'agent est capable d'apprendre Pendulum.
- Si la taille du *replay buffer* devient critique, alors plusieurs cas se présentent :
 - Si l'entropie est absente :
 1. Si la taille du *minibatch* est assez grande, alors l'agent est capable d'apprendre.
 2. Sinon, l'agent n'est pas capable d'apprendre.
 - Sinon, l'agent est toujours capable d'apprendre.

On a donc essayé par la suite de vérifier si le *replay buffer* est un composant critique pour obtenir une bonne performance sur Pendulum.

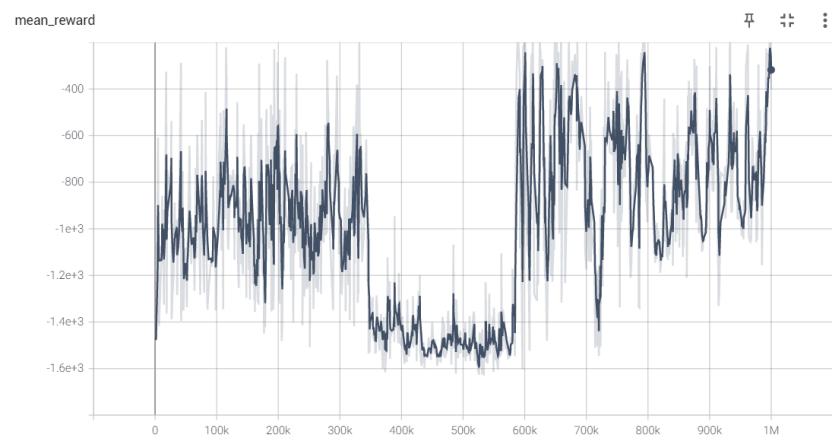
Pour cela, nous avons introduit un *replay buffer* dans un autre algorithme d'*Actor-Critic* n'ayant pas ce composant comme par exemple l'algorithme *Advantage Actor-Critic* (A2C) issu de *Policy Gradient* avec pour estimation Ψ_t la fonction *advantage* et voici les résultats :



Hyper-paramètres :

$$\begin{aligned}\gamma &= 0,99 \\ \text{learning rate} &= 7 \cdot 10^{-4} \\ n\text{-step} &= 5 \\ \text{max value gradient clipping} \\ &= 0,5\end{aligned}$$

FIGURE 14 – Courbe de reward de Advantage Actor-Critic classique (sans replay buffer) sur Pendulum



Hyper-paramètres :

$$\begin{aligned}\gamma &= 0,98 \\ \text{learning rate} &= 6 \cdot 10^{-4} \\ n\text{-step} &= 1 \\ \text{max value gradient clipping} \\ &= 0,3\end{aligned}$$

FIGURE 15 – Courbe de reward de Advantage Actor-Critic classique (sans replay buffer) sur Pendulum correctement paramétrée sur les hyper-paramètres en pratique

En utilisant les hyper-paramètres par défaut d'*Advantage Actor-Critic*, l'agent n'est pas capable d'apprendre. Cependant, il est généralement possible en réalité d'apprendre plus ou moins un environnement à condition de paramétrer correctement³ (FIGURE 15). La question ici est plutôt de savoir si avec l'ajout d'un *replay buffer*, l'agent est capable d'apprendre avec des hyper-paramètres plus robustes tout en obtenant une bonne performance.

3. Voir l'annexe pour le paramétrage.

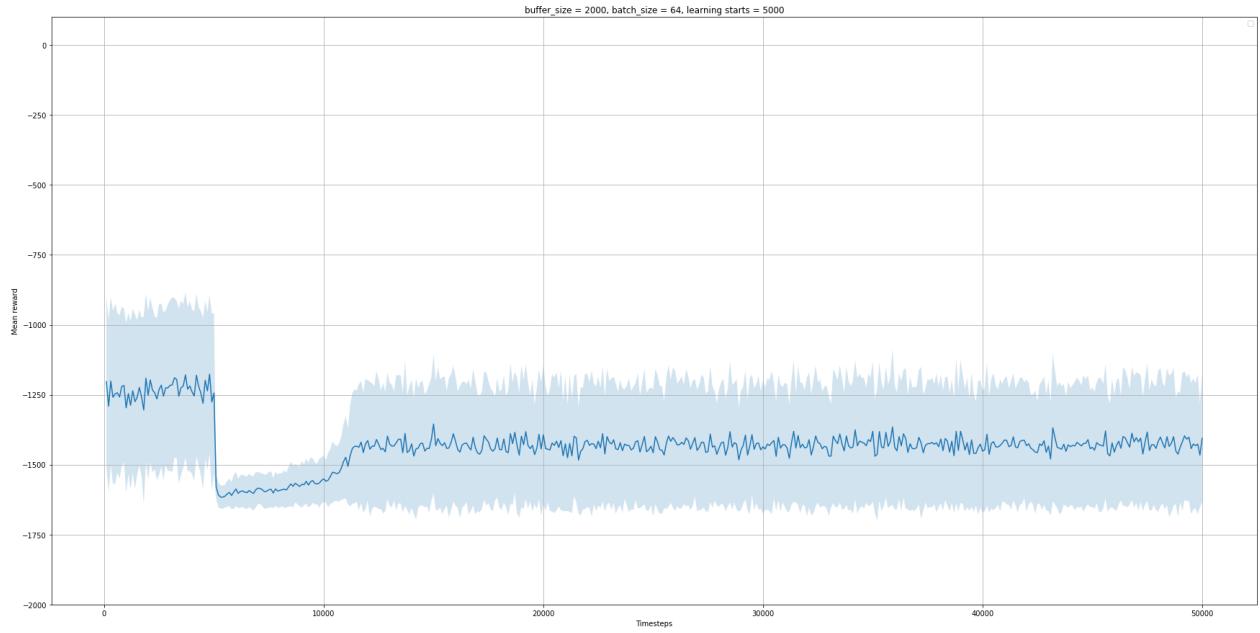


FIGURE 16 – Courbe de reward de Advantage Actor-Critic sur Pendulum avec replay buffer de taille $2 \cdot 10^3$ et avec un minibatch size de 64

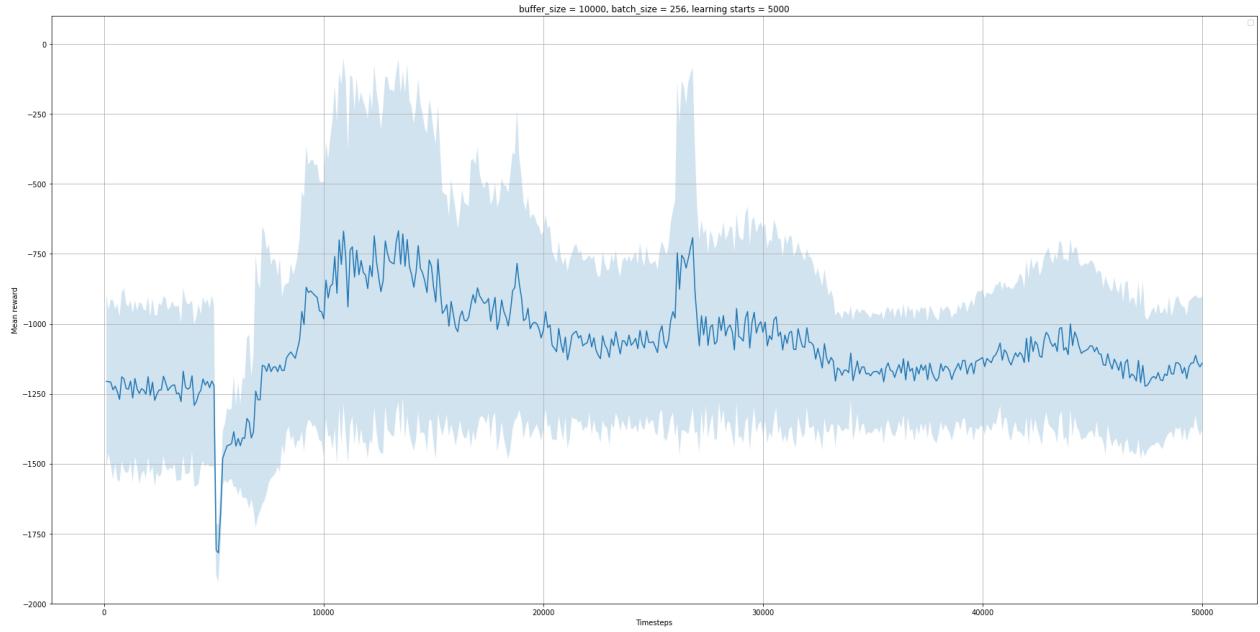


FIGURE 17 – Courbe de reward de Advantage Actor-Critic sur Pendulum avec replay buffer de taille $2 \cdot 10^4$ et avec un minibatch size de 256

Il est étonnant d'observer qu'avec l'ajout d'un *replay buffer* de petite taille, l'algorithme n'est pas capable d'apprendre d'autant plus que les performances obtenues sont identiques à celles de *Advantage Actor-Critic* sans *replay buffer*.

Néanmoins, en augmentant la taille du *replay buffer*, l'algorithme arrive à obtenir une augmentation des performances au début de l'apprentissage avant de converger vers de mauvaises

performances.

Ainsi, l'influence du *replay buffer* dans l'amélioration des performances sur Pendulum reste indistinct malgré les observations obtenus dans l'étude sur *Soft Actor-Critic*. Deux hypothèses sont alors possibles expliquant cette indistinction : soit c'est un autre composant qui est responsable des bonnes performances sur Pendulum, soit le composant *replay buffer* ne suffit pas pour obtenir une bonne performance et qu'il est nécessaire d'intégrer également un autre composant de *Soft Actor-Critic*.

Ce second composant peut être le *target network* ou encore la formule du mise à jour de l'acteur.

Vérifions à présent l'influence du *target network* en observant les performances de *Soft Actor-Critic* sans ce dernier :

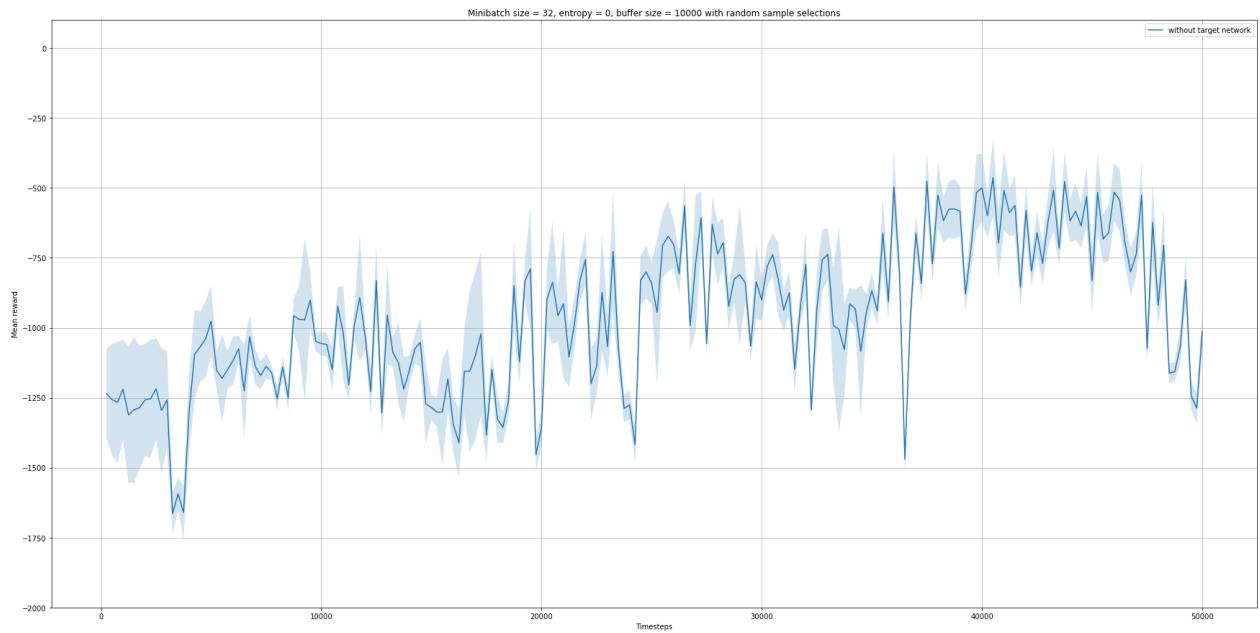


FIGURE 18 – Courbe de reward de *Soft Actor-Critic* sans *target network* ni entropie sur Pendulum avec *replay buffer* de taille 10^4 et avec un minibatch size de 32

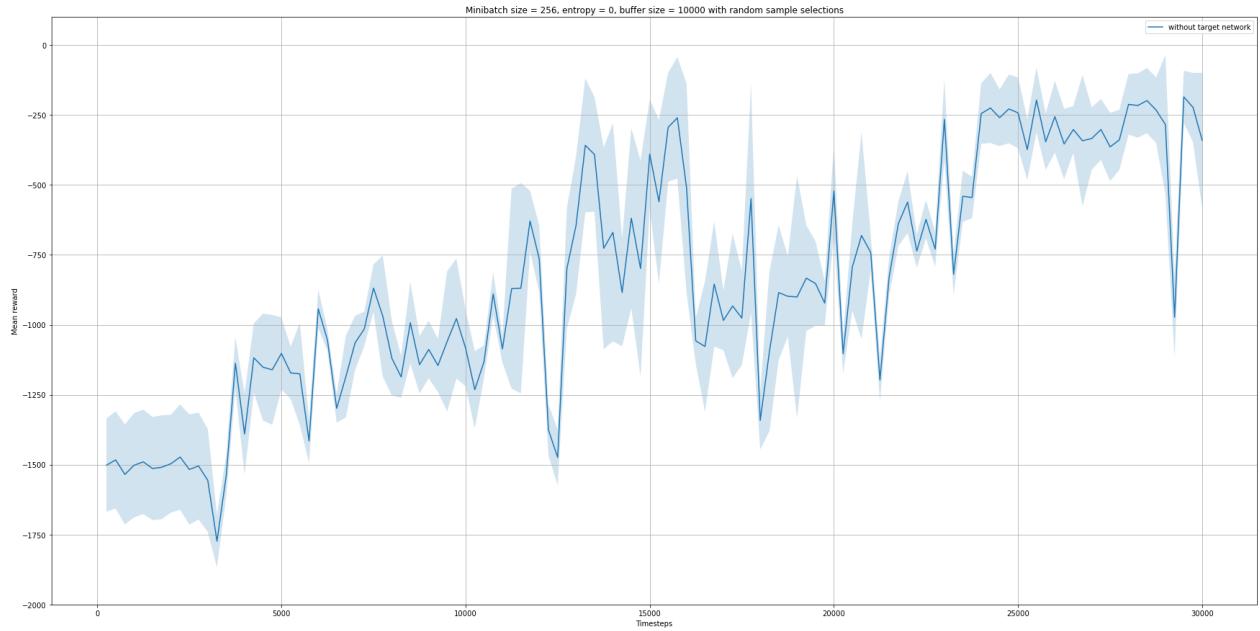


FIGURE 19 – Courbe de reward de Advantage Actor-Critic sans target network ni entropie sur Pendulum avec replay buffer de taille 10^4 et avec un minibatch size de 256

Pour un *minibatch* de taille 32, il est intéressant d'observer que l'algorithme sans un *target network* n'est pas capable d'apprendre Pendulum malgré une tendance générale de hausse des performances pendant l'apprentissage. La convergence est instable. Or pour un *minibatch* de taille 256, l'algorithme est toute même capable d'atteindre une bonne performance avec néanmoins beaucoup plus de pas d'apprentissage que par rapport à un *Soft Actor-Critic* classique.

On en déduit ainsi que le *target network* devient un facteur critique à la bonne performance sur Pendulum seulement dans le cas où la taille du *minibatch* devient critique.

3. Conclusion

Ce projet a permis par une étude par ablation d'observer que certains composants de *Soft Actor-Critic* ont une influence plus ou moins importante sur la bonne performance de l'algorithme sur Pendulum. En effet, en détaillant les différences entre *Policy Gradient* et *Soft Actor-Critic*, nous avons pu distinguer les composants majeurs et leurs mécanismes dans le fonctionnement de certains algorithmes.

Bien que la conclusion à la problématique de savoir quels sont les composants à l'origine de la bonne performance de *Soft Actor-Critic* ne soit pas explicite dans notre projet, plusieurs autres composants possibles comme la formule de mise à jour de l'acteur de *Soft Actor-Critic* peuvent être les raisons de la bonne performance sur Pendulum et méritent d'être étudiés.

Néanmoins, voici une synthèse de ce que le projet a permis d'éclaircir :

Composants non critiques pour la bonne performance sur Pendulum	— La sélection aléatoire des données pour le <i>minibatch</i>
Composants non critiques mais se révèlent critiques lorsque les hyper-paramètres deviennent contraignants	— L'entropie — La taille du <i>minibatch</i> — Le <i>target network</i>
Composants ayant une certaine influence sur la bonne performance	— Le <i>replay buffer</i>

Remerciements

Nous remercions notre professeur M. Sigaud, chercheur à l'Institut des systèmes intelligents et de Robotique (ISIR) d'avoir relu notre rapport et de nous avoir parfaitement encadré tout au long du projet.

Malgré les conditions sanitaires et la complexité des notions nouvelles abordées dans le sujet, le projet a été une très bonne occasion pour nous de découvrir et de comprendre l'apprentissage par renforcement profond et de nous inciter à y découvrir plus.

4. Annexes et Références

Pour observer les différents environnements comme Ant, HalfCheetah, Hopper, Humanoid :
<https://gym.openai.com/envs/#mujoco>

Le github de *Stable Baselines 3* par Antoine Raffin :
<https://github.com/DLR-RM/stable-baselines3>

Le github de *Basic Policy Gradient Labs* par notre professeur :
<https://github.com/DLR-RM/rl-baselines3-zoo>

Le cours sur les réseaux de neurones :
<https://dac.lip6.fr/wp-content/uploads/2021/03/ML-cours5-2020.pdf>

Le cours sur la descente de gradient et sur la notion du *minibatch* :
<https://dac.lip6.fr/wp-content/uploads/2021/02/ML-cours3-2020.pdf>

Le cours en vidéo sur *Policy Gradient* :
https://www.youtube.com/watch?v=_RQYWsvMyyc&list=PLe5mY-Da-ksVku6MoVI5DRLES529BheYI

Le cours en vidéo sur *Soft Actor-Critic* :
https://www.youtube.com/watch?v=_nFX0Zpo50U&list=PLe5mY-Da-ksVku6MoVI5DRLES529BheYI&index=3

Le cours en vidéo sur *Deep Q-Network* pour comprendre le *replay buffer* et le *target network* :
<https://www.youtube.com/watch?v=CXwvOMJujZk&list=PLe5mY-Da-ksWV330WbfazLuy0uR59sers&index=7>

La liste des cours en vidéos sur l'apprentissage par renforcement :
<https://www.youtube.com/c/OlivierSigaud/videos>

Le github de notre projet :
<https://github.com/Vincent-Fu-Lab/Apprentissage-par-renforcement-profound-sur-Pendulum>

Références

- [1] Zafarali AHMED et al. “Understanding the Impact of Entropy on Policy Optimization”. en. In : *International Conference on Machine Learning*. ISSN : 2640-3498. PMLR, mai 2019, p. 151-160. URL : <http://proceedings.mlr.press/v97/ahmed19a.html> (visité le 26/02/2021).

- [2] Marcin ANDRYCHOWICZ et al. "What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study". In : *arXiv :2006.05990 [cs, stat]* (juin 2020). arXiv : 2006.05990. URL : <http://arxiv.org/abs/2006.05990> (visité le 26/02/2021).
- [3] Andras ANTOS, Rémi MUNOS et Csaba SZEPESVARI. *Fitted Q-iteration in continuous action-space MDPs*. en. report. 2007, p. 24. URL : <https://hal.inria.fr/inria-00185311> (visité le 26/02/2021).
- [4] Jonathan BAXTER et Peter L. BARTLETT. "Infinite-horizon policy-gradient estimation". In : *Journal of Artificial Intelligence Research* 15.1 (nov. 2001), p. 319-350. ISSN : 1076-9757.
- [5] Marc Peter DEISENROTH. "A Survey on Policy Search for Robotics". en. In : *Foundations and Trends in Robotics* 2.1-2 (2011), p. 1-142. ISSN : 1935-8253, 1935-8261. DOI : 10.1561/2300000021. URL : <http://www.nowpublishers.com/articles/foundations-and-trends-in-robotics/ROB-021> (visité le 26/02/2021).
- [6] Scott FUJIMOTO, Herke HOOF et David MEGER. "Addressing Function Approximation Error in Actor-Critic Methods". en. In : *International Conference on Machine Learning*. ISSN : 2640-3498. PMLR, juil. 2018, p. 1587-1596. URL : <http://proceedings.mlr.press/v80/fujimoto18a.html> (visité le 26/02/2021).
- [7] Mohammad GHAVAMZADEH, Yaakov ENGEL et Michal VALKO. "Bayesian policy gradient and actor-critic algorithms". In : *The Journal of Machine Learning Research* 17.1 (jan. 2016), p. 2319-2371. ISSN : 1532-4435.
- [8] Tuomas HAARNOJA et al. "Reinforcement Learning with Deep Energy-Based Policies". en. In : *International Conference on Machine Learning*. ISSN : 2640-3498. PMLR, juil. 2017, p. 1352-1361. URL : <http://proceedings.mlr.press/v70/haarnoja17a.html> (visité le 26/02/2021).
- [9] Tuomas HAARNOJA et al. "Soft Actor-Critic Algorithms and Applications". In : *arXiv :1812.05905 [cs, stat]* (jan. 2019). arXiv : 1812.05905. URL : <http://arxiv.org/abs/1812.05905> (visité le 26/02/2021).
- [10] Tuomas HAARNOJA et al. "Soft Actor-Critic : Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". en. In : *International Conference on Machine Learning*. ISSN : 2640-3498. PMLR, juil. 2018, p. 1861-1870. URL : <http://proceedings.mlr.press/v80/haarnoja18b.html> (visité le 26/02/2021).
- [11] Dingcheng LI et al. "Video Recommendation with Multi-gate Mixture of Experts Soft Actor Critic". In : *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '20. New York, NY, USA : Association for Computing Machinery, juil. 2020, p. 1553-1556. ISBN : 978-1-4503-8016-4. DOI : 10.1145/3397271.3401238. URL : <https://doi.org/10.1145/3397271.3401238> (visité le 02/04/2021).

- [12] Yiming PENG et al. "NEAT for large-scale reinforcement learning through evolutionary feature learning and policy gradient search". In : *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '18. New York, NY, USA : Association for Computing Machinery, juil. 2018, p. 490-497. ISBN : 978-1-4503-5618-3. DOI : 10.1145/3205455.3205536. URL : <https://doi.org/10.1145/3205455.3205536> (visité le 02/04/2021).
- [13] Martin RIEDMILLER. "Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method". en. In : *Machine Learning : ECML 2005*. Sous la dir. de João GAMA et al. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2005, p. 317-328. ISBN : 978-3-540-31692-3. DOI : 10.1007/11564096_32.
- [14] John SCHULMAN et al. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In : *arXiv :1506.02438 [cs]* (oct. 2018). arXiv : 1506.02438. URL : <http://arxiv.org/abs/1506.02438> (visité le 26/02/2021).
- [15] Olivier SIGAUD et Freek STULP. "Policy search in continuous action domains : An overview". en. In : *Neural Networks* 113 (mai 2019), p. 28-40. ISSN : 0893-6080. DOI : 10.1016/j.neunet.2019.01.011. URL : <https://www.sciencedirect.com/science/article/pii/S089360801930022X> (visité le 26/02/2021).
- [16] B. STAPELBERG et K. M. MALAN. "Global structure of policy search spaces for reinforcement learning". In : *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '19. New York, NY, USA : Association for Computing Machinery, juil. 2019, p. 1773-1781. ISBN : 978-1-4503-6748-6. DOI : 10.1145/3319619.3326843. URL : <https://doi.org/10.1145/3319619.3326843> (visité le 02/04/2021).
- [17] Richard S. SUTTON et Andrew G. BARTO. *Reinforcement Learning, second edition : An Introduction*. en. Google-Books-ID : uWV0DwAAQBAJ. MIT Press, nov. 2018. ISBN : 978-0-262-35270-3.