DDPG : explication de l'algorithme et analyse expérimentale sous différentes contraintes

Quentin LENET, Justine PEPIN, Arnaud VENET
École Polytechnique de Montréal

Abstract

L'algorithme *Deep Deterministic Policy Gradient* (DDPG) présenté dans l'article étudié (Lillicrap et al. 2016) est une méthode d'apprentissage par renforcement profond pouvant apprendre des politiques dans un espace d'actions continu. L'implémentation de cet algorithme nécessite l'utilisation de nombreux hyperparamètres. Nous avons donc voulu déterminer la robustesse de l'algorithme vis-à-vis des réglages expérimentaux différents de ceux proposé dans l'article. Nos résultats montrent que le type de bruit importe peu, ce qui compte est d'en avoir en quantité raisonnable. De même, pour les légères modifications de l'architecture du réseau, les résultats sont très similaires.

Introduction

Cette première section portera sur les travaux antérieurs à DDPG dans la littérature. Il y en a deux qui sont spécialement importants puisqu'ils sont les prédécesseurs de DDPG ; ils seront détaillés dans les prochains paragraphes.

DQN - Première influence de DDPG

DQN (*Deep Reinforcement Learning*) utilise un réseau de neurones pour apprendre à un agent une politique. Il a été présenté par DeepMind technologies en 2013 (Mnih et al. 2013). Cet algorithme peut apprendre une politique à un agent qui évolue dans un environnement avec un espace d'états très grand. En effet, le réseau de neurones joue le rôle de la fonction $\mathcal{Q}$ qui prend à sa couche d'entrée les états, et retourne en sortie, à l'aide d'un softmax, quelle action discrète emprunter. Plusieurs améliorations ont été proposées pour cet algorithme, comme le *replay buffer* (mémoire de reprise). Celui-ci emmagasine les transitions d'un état à un autre rencontrées lors de l'apprentissage. Une transition se présente sous cette forme (s_t, a_t, r_t, s_{t+1}) pour un agent qui passe de l'état s_t à l'état s_{t+1} en ayant effectué l'action a_t et qui a reçu la récompense r_t. La mémoire de reprise, en enregistrant beaucoup de transitions, peut retourner des lots aléatoires composés de plusieurs transitions non successives. Ceci apporte de la stabilité à l'algorithme lors de l'apprentissage. Une autre amélioration proposée par Van Hasselt et al. est l'ajout d'un *target network* (réseau cible) $\mathcal{Q}\prime$, (Wang et al. 2015) (Wang et al. 2015). Ce réseau cible est initialisé avec les mêmes poids que le réseau initial, et sa mise à jour est faite à partir de ce dernier, avec la formule de Polyak.

$$Q\prime = (1-\tau)Q\prime + \tau Q \tag{1}$$

Après l'initialisation, pour le reste de l'apprentissage, le paramètre τ est pris petit (vers 0.01), pour adoucir la mise à jour du réseau cible. Ceci ajoute encore de la stabilité à l'apprentissage.

Cet algorithme présente certaines limites et ne peut pas s'appliquer aux environnements trop complexes. Les choix d'actions sélectionnables par l'agent restent restreints ; pour évoluer d'un état à l'autre, notre agent n'a le choix qu'entre quelques actions dont le nombre correspondra au nombre de neurones de la couche de sortie. Pour que l'agent puisse effectuer un grand nombre d'actions, voire des actions continues, il faut aller voir d'autres approches et d'autres algorithmes.

DPG - Deuxième influence de DDPG

DPG (*Deterministic Policy Gradient*) est un algorithme qui a été présenté en 2014 par des gens de chez DeepMind (Silver et al. 2014). Il vise à éviter les intégrales trop laborieuses à calculer en utilisant le gradient d'une politique déterministe au lieu de celui d'une politique stochastique. Ainsi, on peut améliorer la performance et l'efficacité dans l'estimation du gradient par rapport au Q-learning, qui, lorsqu'appliqué directement aux espaces d'actions continus, tente d'optimiser l'action a au temps t à chaque pas de temps pour trouver la politique optimale, ce qui est bien sûr vorace en temps lorsqu'on se sert d'un grand espace d'actions.

DPG est un algorithme avec un acteur et un critique où typiquement l'acteur ajuste les paramètres en performant la descente des gradients, tandis que le critique estime la fonction d'utilité selon l'action. Dans notre cas, le critique utilise tout comme dans DQN l'équation de Bellman, mais comme DPG travaille avec une politique déterministe, la fonction représentant la politique du choix d'action $\mu(s|\theta^\mu)$ associe

directement chaque état s avec une action a, ce qui fait qu'on peut éviter l'espérance qui se trouve normalement en milieu d'équation.

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))\right] \tag{2}$$

On utilise l'erreur quadratique pour la fonction de perte, *MSELoss*, pour mettre à jour notre $\mathcal{Q}$.

$$L = \frac{1}{N}\Sigma_i(y_i - \mathcal{Q}(s_i, a_i|\theta^{\mathcal{Q}}))^2 \tag{3}$$

Du côté de l'acteur, pour apprendre la politique déterministe cible $\mu_{\theta(s)}$, on utilise des trajectoires générées par une politique stochastique arbitraire $\pi(s, a)$. Avec celle-ci, on peut calculer le gradient de la performance J.

$$\nabla_{\theta^{\mu}} J \approx \mathbb{E}_{s_t \sim \rho^{\beta}}\left[\nabla_a \mathcal{Q}(s, a|\theta^{\mathcal{Q}})|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_{\mu}} \mu(s|\theta^{\mu})|_{s=s_t}\right] \tag{4}$$

Un détail intéressant à propos du gradient de la politique déterministe est qu'il s'agit d'un cas limite de la politique stochastique ; c'est pourquoi on peut employer la version déterministe au même titre que celle stochastique, avec bien entendu des façons de faire différentes et des avantages différents. Justement, un avantage potentiel d'utiliser une politique déterministe est de contourner l'échantillonnage d'importance, qui utilise les retours de la politique d'apprentissage $\pi(s, a)$ et qui permet de les ajuster à la politique de notre choix. Typiquement, pour les algorithmes de type acteur-critique comme le nôtre, mais qui font le choix d'action glouton avec une politique stochastique, on doit faire l'échantillonnage d'importance pour l'acteur comme pour le critique. Toutefois, DPG tire bien son épingle du jeu et permet d'éviter la très grande variance qui peut survenir avec l'échantillonnage d'importance grâce à la disparition d'une intégrale dans les calculs pour l'acteur lorsqu'on change la politique stochastique en politique déterministe. Finalement, on peut l'éviter aussi une seconde fois dans le critique en utilisant le Q-learning.

Approche théorétique

DDPG plus en détail

DDPG (*Deep Deterministic Policy Gradient*) s'inscrit à la fois dans la lignée de DPG et dans la lignée de DQN puisqu'il reprend volontiers les points caractéristiques de ces deux algorithmes et les intègre ensemble pour former un algorithme hybride. Les points repris sont notamment : l'usage d'un tampon pour rejouer les séquences d'apprentissage, la mémoire de reprise, ce qui permet de mettre à jour les paramètres du réseau à partir d'échantillons distribués indépendamment et équitablement ; la mise à jour adoucie des poids d'un réseau cible, ce qui permet d'avoir une plus grande stabilité dans l'apprentissage ; ainsi qu'un modèle d'algorithme acteur-critique avec le choix d'action glouton et une politique déterministe.

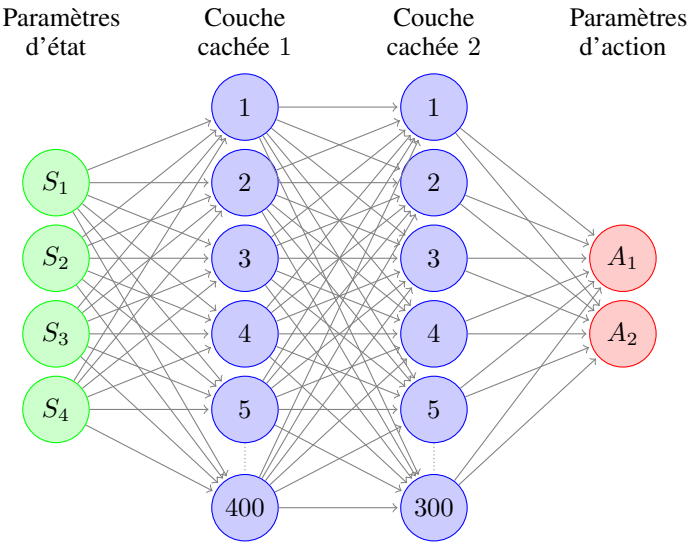

FIGURE 1 – Réseau acteur μ

Également, quelques techniques supplémentaires ont été utilisées pour obtenir un meilleur apprentissage, comme la normalisation des lots (S Ioffe 2015), qui permet d'éviter les entrées trop disparates qui augmentent la difficulté que les réseaux ont à trouver les bons hyperparamètres, ou encore comme l'ajout de bruit pour explorer plus efficacement l'espace d'action continu. Pour ce faire, on peut tout simplement ajouter $\mathcal{N}$ à notre politique, où $\mathcal{N}$ peut être n'importe quel type de bruit qui convient à notre environnement.

$$\mu\prime(s_t) = \mu(s_t|\theta_t^{\mu}) + \mathcal{N} \tag{5}$$

Dans l'article de DDPG, le bruit proposé était généré par le processus d'Ornstein-Uhlenbeck (Uhlenbeck and Ornstein 1930). Ce processus simule un mouvement brownien avec friction d'une particule. Il existe alors une autocorrélation temporelle du bruit.

En somme, décrit grossièrement, DDPG est un algorithme qui combine DQN et DPG. Il doit donc entretenir quatre réseaux de neurones : un acteur, un critique, un acteur cible et un critique cible. Les figures 1 et 2 montrent de façon visuelle à quoi ressemblent ces réseaux. Les réseaux cible subissent des mises à jour adoucies à partir de leur réseau d'apprentissage respectif, et ces derniers s'entraînent à partir de mémoires de reprise. Il y a également de la normalisation de lots pour optimiser l'apprentissage et du bruit pour prévenir le surapprentissage. Tout cet ensemble permet d'obtenir un algorithme à performance satisfaisante sur les environnements qui ont de grands espaces d'actions et d'états.

DDPG et les modèles classiques d'Acteur-Critique

Les méthodes dites d'acteur-critique implémentent de façon générale l'itération de la politique en alternant entre

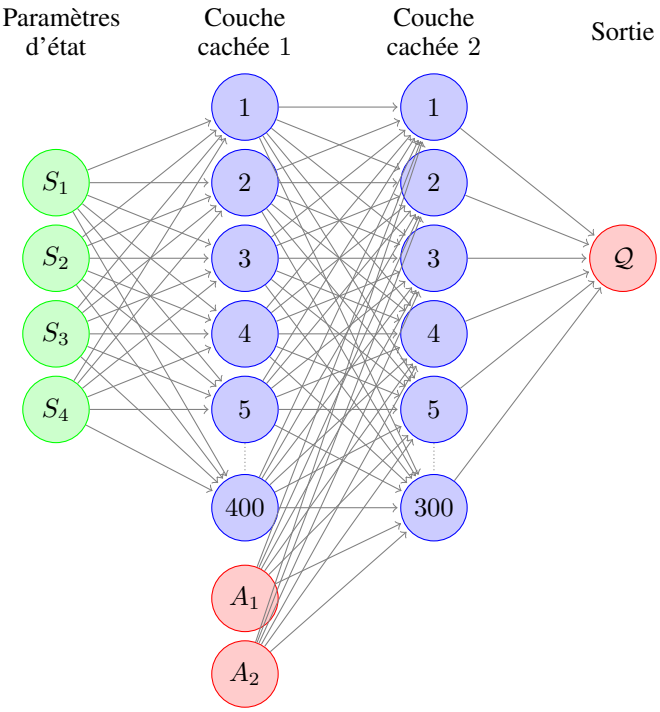

FIGURE 2 – Réseau critique $\mathcal{Q}$

une étape d'évaluation de la politique et une étape d'amélioration de la politique (Gasic). Le critique est en charge d'évaluer la politique dans son état présent, tandis que l'acteur est en charge de l'améliorer s'il réussit à trouver une action qui est meilleure que celle recommandée par la politique présente. Dans le cas où l'espace des actions et l'espace des états sont très grands, l'acteur comme le critique utilisent des approximateurs compatibles de fonctions, ces dernières étant trop complexes pour être évaluées directement. Comme l'acteur utilise une Q-valeur pour la mise à jour de la politique, le critique doit estimer la Q-fonction. Pour les grands espaces d'états, la différence temporelle des moindres carrés peut servir à estimer la Q-fonction. Quant à l'acteur, il performe d'ordinaire la descente de gradient stochastique sur la surface générée par les performances de la politique.

Il y a donc quelques différences entre DDPG et la version plus classique d'un acteur-critique, qui proviennent de DQN comme de DPG. Par exemple, DQN amène la notion de réseau cible qui ne se trouve pas dans le modèle acteur-critique de base, ou encore DPG amène la notion de politique de gradient déterministe alors que les algorithmes acteur-critique jusque-là se servaient de la politique de gradient stochastique. Néanmoins, DDPG reste, de par sa logique qui distingue le rôle d'acteur du rôle de critique de la même façon que le font les modèles classiques d'acteur-critique, dans la lignée de ce type d'algorithme.

Il faudrait aussi souligner que les méthodes utilisant des politiques déterministes comme DDPG ne sont pas la manne des algorithmes. D'autres algorithmes qui optimisent des politiques stochastiques de contrôle sont très efficaces sur les environnements dotés de grands espaces d'actions et d'états, comme par exemple TRPO (*Trust Region Policy Optimization*) (Schulman. et al. 2015), qui fait lui aussi plusieurs itérations afin de corriger graduellement une politique. Pour faire son approximation du retour de la politique, TRPO utilise une pénalité de divergence KL (*Kullback-Leibler*) qui est une méthode assez différente que ce qui est fait avec DDPG. Toutefois, un bon avantage de DDPG vis-à-vis de TRPO reste sa vitesse, puisqu'il est très facile de manipuler les politiques déterministes comparativement à celles stochastiques.

Résultats des expériences et discussion

Nous avons implémenté DDPG en suivant les algorithmes présentés dans l'article. L'environnement utilisé pour les tests est OpenAI Gym, qui intègre quelques problèmes à la physique réaliste mais simplifiés du moteur MuJoCo (Todorov, Erez, and Tassa 2012). Ceux-ci consistent généralement à contrôler différentes articulations d'un modèle caricatural d'animal ou d'humanoïde pour effectuer une séquence de gestes précis.

Intrigués par certains choix d'hyperparamètres qui n'étaient pas très bien justifiés dans l'article de DDPG, nous avons décidé de modifier le bruit d'exploration afin de voir en quoi il permet d'améliorer la convergence de l'apprentissage. Dans un premier temps, considérons un bruit d'exploration unifome (variance plus grande par rapport à un processus gaussien).
Commençons par un problème simple : pendulum, qui consiste à actionner une liaison pivot pour placer une tige à la verticale, en équilibre au dessus de la liaison.

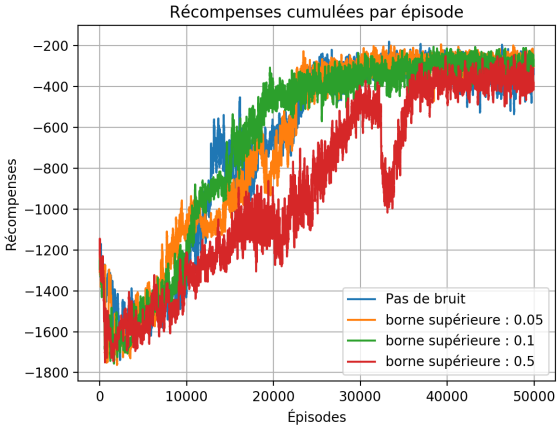

FIGURE 3 – Influence d'un bruit uniforme centré pour le pendule

Nous voyons sur la figure 3 qu'à part le cas où un très

grand bruit d'exploration (entre -0.5 et +0.5, sachant que les actions sont bornées entre -1 et +1) est imposé, les performances sont peu impactées par le choix du bruit.

Poursuivons avec un problème plus complexe, nommé demi guépard, qui consiste à apprendre à actionner deux pattes (une avant et une arrière) attachées à un corps de mammifère simplifié pour avancer le plus vite possible. Il y a ici 4 articulations à contrôler, les «hanches» et les «genoux». Il semblerait que mettre un léger bruit permette de réduire la

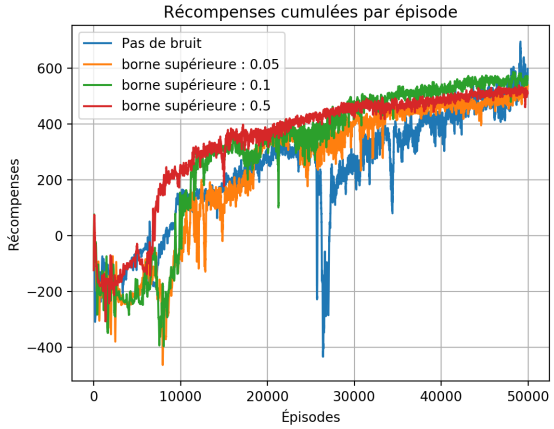

FIGURE 4 – Influence d'un bruit uniforme centré pour le demi guépard

variance de la récompense au cours de l'apprentissage, et d'accélérer légèrement la convergence comme montré sur la figure 4.

Étudions maintenant le cas d'un processus gaussien blanc, non corrélé dans le temps (contrairement à l'article).

Ici, sur la figure 5, nous voyons que l'impact est faible. Le problème pendulum étant simple, il est possible que nous puissions converger de toute façon.

Dans le cas du demi guépard, figure 6, nous observons qu'un bruit trop fort abaisse le "plafond" d'apprentissage : nous restons bloqués à un score de 400, alors qu'avec une petite valeur de bruit, nous pouvons atteindre 600. Comme pour le cas uniforme, l'ajout d'un petit bruit réduit la variance de la récompense, et accélère la convergence.
Jusqu'à maintenant, le type de bruit semble peu importer ; l'essentiel semble être d'introduire un petit bruit dans l'exploration, accélérant ainsi la convergence (découverte de politiques novatrices facilitée).

La figure 7 montre la performance pour les réglages de bruit correspondant au processus d'Ornstein-Uhlenbeck utilisé par les auteurs de l'article. Étonnamment, la récompense par épisode stagne à 500, ce qui est moins que ce que nous avions obtenu avec des processus gaussiens décorrélés temporellement.

La figure 8 montre les résultats d'expérimentations où nous avons légèrement changé le réseau du critique, en introduisant les actions au niveau de la couche d'entrée, avec les

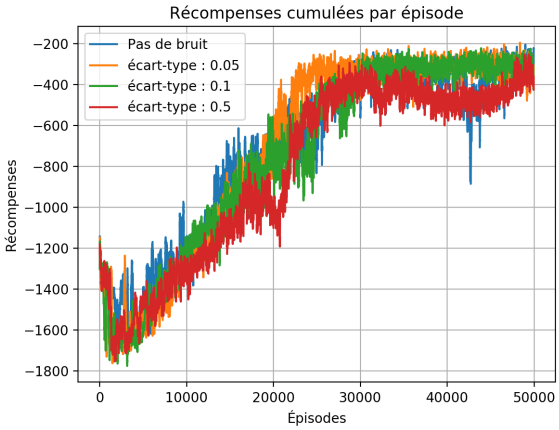

FIGURE 5 – Influence d'un bruit gaussien centré pour le pendule

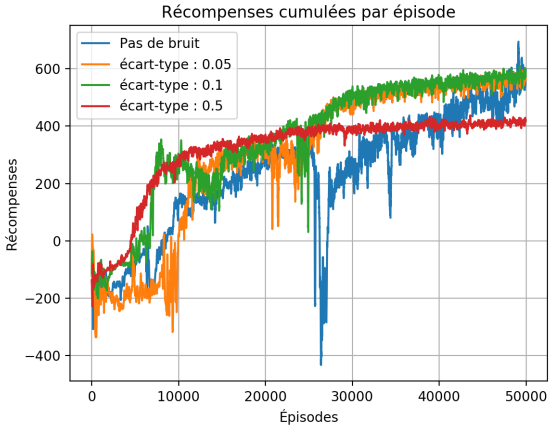

FIGURE 6 – Influence d'un bruit gaussien centré pour le demi guépard

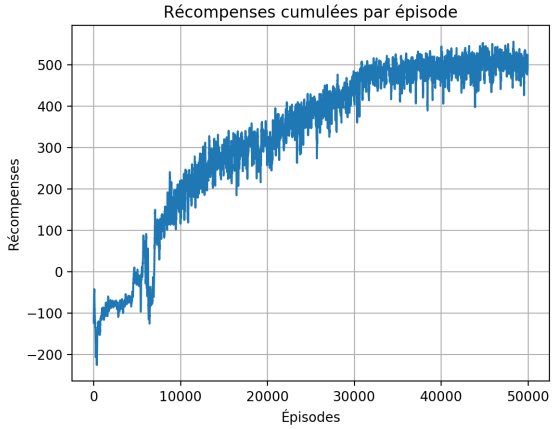

FIGURE 7 – Influence du bruit Ornstein-Uhlenbeck pour le demi guépard

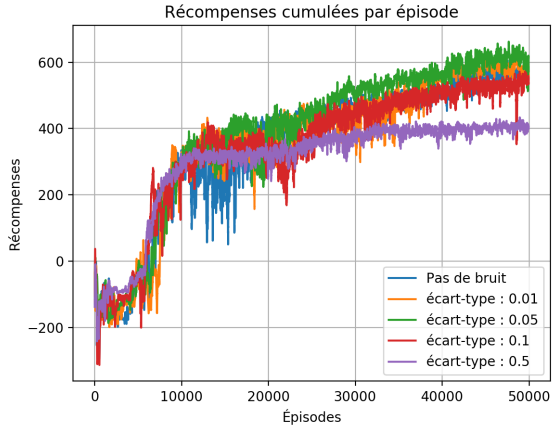

FIGURE 8 – Influence de l'injection des actions à la première couche du réseau, pour un bruit uniforme, sur le demi guépard

états, au lieu d'attendre la deuxième couche cachée comme ce qui était initialement présenté dans l'article. Nous observons que nous conservons, voire améliorons légèrement les propriétés de convergence de notre réseau.

Conclusion

L'algorithme DDPG étudié permet l'apprentissage de politiques à plusieurs actions continues avec coordination, commme dans le cas des pattes du demi guépard. Si nous percevons l'apport positif d'un petit bruit d'exploration dans le choix des actions à évaluer, l'intérêt de décaler l'entrée des actions à la deuxième couche cachée dans le réseau critique n'a pas été démontré. La raison pour laquelle les auteurs avaient opté pour ce choix n'est pas claire ; il aurait été pertinent de laisser l'algorithme se dérouler sur un plus long horizon, comme celui utilisé par les auteurs de DDPG, afin de vérifier si une différence survient plus tard au cours des expériences. Tout compte fait, notre approche aura sûrement manqué de raffinement, puisque nous n'avons pas testé sur tous les environnements proposés par les auteurs de DDPG lors de leurs propres tests, ni avec les horizons qui étaient également suggérés. Toutefois, étant donné le temps très long requis pour faire rouler ces environnements avec beaucoup de cycles, nous sommes satisfaits de l'étendue couverte par nos méthodes.

Lien vers notre Github

`https://github.com/JustineSurGithub/INF8225`

References

Gasic, M. Actor-critic methods. `http://mi.eng.cam.ac.uk/~mg436/LectureSlides/MLSALT7/L5.pdf`. Université de Cambridge.

Lillicrap, T. P. ; Hunt, J. J. ; Pritzel, A. ; Heess, N. ; Erez, T. ; Tassaa, Y. ; Silver, D. ; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. *arXiv preprint arXiv :1509.02971v5*.

Mnih, V. ; Kavukcuoglu, K. ; Silver, D. ; Graves, A. ; Antonoglou, I. ; Wierstra, D. ; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv :1312.5602*.

S Ioffe, C. S. 2015. Batch normalization : Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv :1502.03167*.

Schulman., J. ; Levine, S. ; Moritz, P. ; Jordan, M. I. ; and Abbeel, P. 2015. Trust region policy optimization. *arXiv preprint arXiv :1502.05477*.

Silver, D. ; Lever, G. ; Heess, N. ; Degris, T. ; Wierstra, D. ; and Riedmiller, M. 2014. Deterministic policy gradient algorithms. In *ICML*.

Todorov, E. ; Erez, T. ; and Tassa, Y. 2012. Mujoco : A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference*, 5026–5033.

Uhlenbeck, G. E., and Ornstein, L. S. 1930. On the theory of the brownian motion. *Physical review, 36(5) :823*.

Wang, Z. ; Schaul, T. ; Hessel, M. ; Van Hasselt, H. ; Lanctot, M. ; and De Freitas, N. 2015. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv :1511.06581*.