

Apprentissage par renforcement

Cours 5: Advanced Policy Gradients

Sylvain Lamprier

UE RLD - Master DAC

2021

Policy Gradients Optimization : Challenges

Règle de mise à jour des paramètres pour Policy Gradients :

policy gradient (steepest direction to maximize rewards)

$$\mathbf{g} = \nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) A^{\pi_{\theta}}(s_t, a_t) \right]$$

$$\theta_{k+1} = \theta_k + \alpha \mathbf{g}$$

take a gradient step in updating the policy



Malheureusement, cette mise à jour fait l'hypothèse que la surface de la fonction à optimiser est plate (méthode du premier ordre)

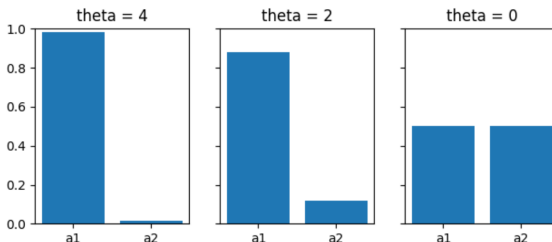
- ▶ Si on se déplace trop vite (α trop grand), on peut effectuer des mouvements catastrophiques
- ▶ Si on se déplace trop lentement (α trop petit), on risque d'apprendre trop lentement (et si l'exploration nous amène dans une zone plate avec politique qui fonctionne mal localement, on risque d'avoir du mal à en sortir).

En RL, pas de pas de gradient idéal sur l'ensemble du problème (risque d'exploding ou vanishing gradient)

Policy Gradients Optimization : Challenges

Considérons la paramétrisation suivante :

$$\pi_{\theta}(a) = \begin{cases} \sigma(\theta) & a = 1 \\ 1 - \sigma(\theta) & a = 2 \end{cases}$$



⇒ Un changement mineur dans les paramètres peut modifier drastiquement la politique

Policy Gradients Optimization : Challenges

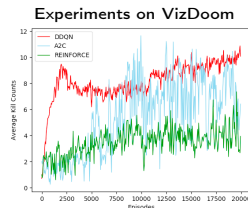
Proposition : Limiter les déplacements de la politique pour qu'elle ne varie pas au delà d'un seuil à chaque étape

Problèmes :

- ▶ Comment régler le seuil ?
- ▶ Comment transposer ce seuil dans l'espace des paramètres ?

Autre problème :

- ▶ PG souffre d'une forte variance
- ▶ On-Policy : Même politique pour sampler et apprendre
- ⇒ Chaque trajectoire utilisée une seule fois
- ⇒ Faible efficacité d'apprentissage (très grand nombre de trajectoires à échantillonner)



Importance Sampling

- ▶ Forte Variance \Rightarrow requiert un grand nombre de trajectoires à échantillonner à chaque étape
- ▶ Comment réduire le nombre d'échantillons de trajectoires ?

importance sampling

$$\begin{aligned} E_{x \sim p(x)}[f(x)] &= \int p(x) f(x) dx \\ &= \int \frac{q(x)}{q(x)} p(x) f(x) dx \\ &= \int q(x) \frac{p(x)}{q(x)} f(x) dx \\ &= E_{x \sim q(x)} \left[\frac{p(x)}{q(x)} f(x) \right] \end{aligned}$$

$$\frac{\pi_{\theta'}(\tau)}{\pi_{\theta}(\tau)} = \frac{\prod_{t=1}^T \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\prod_{t=1}^T \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)}$$

\Rightarrow Échantillons passés peuvent encore servir. Soit $\pi_{\theta(i)}(\tau^{(i)})$ la distribution ayant servi à échantillonner $\tau^{(i)}$:


$$\begin{aligned} \nabla_{\theta} J(\theta) &\approx \frac{1}{M} \sum_{\tau^{(i)}} \frac{\pi_{\theta}(\tau^{(i)})}{\pi_{\theta(i)}(\tau^{(i)})} \left[R(\tau^{(i)}) \nabla_{\theta} \log \pi_{\theta}(\tau^{(i)}) \right] \\ &\approx \frac{1}{M} \sum_{\tau^{(i)}} \exp(\log \pi_{\theta}(\tau^{(i)}) - \log \pi_{\theta(i)}(\tau^{(i)})) \left[R(\tau^{(i)}) \nabla_{\theta} \log \pi_{\theta}(\tau^{(i)}) \right] \end{aligned}$$

\Rightarrow Off-Policy Policy Gradient (experience replay, target policy)

Importance Sampling

Importance Sampling permet de réutiliser d'anciennes trajectoires :

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim \pi_{\theta}(\tau)} \left[\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \left(\prod_{t'=1}^t \frac{\pi_{\theta'}(\mathbf{a}_{t'} | \mathbf{s}_{t'})}{\pi_{\theta}(\mathbf{a}_{t'} | \mathbf{s}_{t'})} \right) \left(\sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \right]$$



refine the current policy

Sample data from another policy

Malheureusement, l'estimation par IS de l'espérance de $f(x)$ selon P en utilisant une distribution Q donnée possède une variance de :

$$\frac{1}{N} \left(\mathbb{E}_{x \sim Q} \left[\left(\frac{P(x)}{Q(x)} f(x) \right)^2 \right] - \mathbb{E}_{x \sim P} \left[f(x) \right]^2 \right) \quad \text{Preuve}$$

- ⇒ Si P trop différente de Q , la variance peut exploser
- ⇒ Si la politique de la trajectoire utilisée est trop différente de la politique courante : mises à jour très risquées
- ⇒ On ne peut pas utiliser les trop anciennes trajectoires (dépend du learning rate)

Policy Performance Bounds

Pour résumer, ce que l'on souhaite :

- ▶ Ne pas sampler et apprendre selon les mêmes politiques (on-policy peu efficace)
- ▶ Contrôler les déplacements dans l'espace des politiques (et non dans l'espace des paramètres)

Pour cela, on définit la performance relative d'une politique π' par rapport à une autre π selon :

$$J(\pi') = J(\pi) + \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi}(s_t, a_t) \right]$$

avec A^{π} la fonction d'avantage selon π :

$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$$

Preuve ?

Policy Performance Bounds

Est-ce qu'on pourrait exploiter cette relation de performances pour définir une méthodologie d'optimisation plus efficace (avec π' la nouvelle politique et π l'ancienne à chaque itération) ?

$$\begin{aligned}\max_{\pi'} J(\pi') &= \max_{\pi'} J(\pi') - J(\pi) \\ &= \max_{\pi'} \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi}(s_t, a_t) \right]\end{aligned}$$

- 👍 On exprime la performance de π' en fonction d'avantages sur π
- 👎 On sample toujours de la politique qu'on optimise π'

Policy Performance Bounds

Soit la distribution discountée des futurs états :

$$d^\pi(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t P(s_t = s | \pi)$$

On peut alors ré-écrire le problème sous la forme d'une somme sur les états :

$$\begin{aligned} J(\pi') - J(\pi) &= \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \right] \\ &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s \sim P(s_t=s | \pi'), a \sim \pi'(a|s)} [A^\pi(s, a)] \\ &= \frac{1}{1 - \gamma} \mathbb{E}_{s \sim d^{\pi'}(s), a \sim \pi'(a|s)} [A^\pi(s, a)] \\ &= \frac{1}{1 - \gamma} \mathbb{E}_{s \sim d^{\pi'}(s), a \sim \pi(a|s)} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^\pi(s, a) \right] \end{aligned}$$

👍 On risque moins d'exploser que IS classique car on n'a plus de rapport de produits sur trajectoires entières

👎 On a toujours le problème de $s \sim d^{\pi'}(s)$

Policy Performance Bounds

Et si on prenait comme objectif :

$$\begin{aligned} L^\pi(\pi') &= \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^\pi(s), a \sim \pi(a|s)} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^\pi(s_t, a_t) \right] \\ &= \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t \frac{\pi'(a_t|s_t)}{\pi(a_t|s_t)} A^\pi(s_t, a_t) \right] \end{aligned}$$

Ok tant que $d^\pi \approx d^{\pi'}$. Mais si on s'en écarte trop, on n'optimise plus ce que l'on souhaite...

⇒ Risque de diverger/osciller, etc.

- Comment quantifier ? À partir de quand n'est-ce plus une bonne approximation ?

En fait on peut établir la borne (Relative performance bound) :

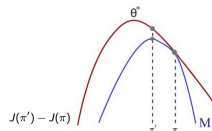
$$J(\pi') - J(\pi) \geq L^\pi(\pi') - C D_{KL}^{\max}(\pi || \pi') \quad \text{Preuve ?}$$

avec $C = 4\epsilon\gamma/(1-\gamma)^2$ et $\epsilon = \max_{s,a} |A^\pi(s, a)|$

TRPO : Trust-Region Policy Optimization

Minorize-Maximization

- ▶ Famille de méthodes (incluant EM) qui considèrent à chaque étape un minorant M de la quantité qu'ils souhaitent maximiser



TRPO [Sch+15] exploite cette idée en considérant l'objectif suivant à chaque étape :

$$M = L^{\pi}(\pi') - C D_{KL}^{max}(\pi || \pi')$$

En théorie, un algorithme maximisant M à chaque étape assure des mises à jour de politiques améliorant J .

Mais...

- ▶ C difficile à calculer (estimation de $\max_{s,a} |A^{\pi}(s, a)|$) et risque d'impliquer un apprentissage trop lent (trop grand lorsque $\gamma \approx 1$)
- ▶ $D_{KL}^{max}(\pi || \pi')$ impossible à calculer pour la plupart des problèmes
- ▶ Erreurs d'approximation de A^{π} qui doit être estimée sur trajectoires échantillonnées

TRPO : Trust-Region Policy Optimization

En pratique, TRPO considère le problème sous contrainte suivant (trust-region) :

$$\max_{\pi'} \mathcal{L}_{\pi}(\pi') \quad \text{s.t.} \quad \mathbb{E}_{s \sim d^{\pi}} [D_{KL}(\pi || \pi')(s)] \leq \delta$$



Line search
(like gradient ascent)



Trust region

Plus de garanties mais :

- ▶ Hyper-paramètre δ permet de contrôler la zone de confiance dans l'espace des politiques \Rightarrow plus de risque d'explosion du gradient, stratégie off-policy
- ▶ $\bar{D}_{KL}(\pi || \pi') = \mathbb{E}_{s \sim d^{\pi}} [D_{KL}(\pi || \pi')(s)]$ bien plus facile à estimer que $D_{KL}^{max}(\pi || \pi')$
- ▶ A^{π} estimé par exemple par temporal difference selon NN V_{ϕ}

TRPO : Trust-Region Policy Optimization

On veut donc optimiser un problème de la forme :

$$\pi_{k+1} = \arg \max_{\pi'} \mathcal{L}_{\pi_k}(\pi') \quad \text{s.t.} \quad \bar{D}_{KL}(\pi_k || \pi') \leq \delta$$

La méthode des gradients naturels propose de s'intéresser à l'expansion de Taylor de second ordre : $f(\theta) \approx f(\theta_k) + \nabla_{\theta} f(\theta)|_{\theta_k}^T (\theta - \theta_k) + \frac{1}{2}(\theta - \theta_k)^T \nabla_{\theta}^2 f(\theta)|_{\theta_k} (\theta - \theta_k)$

Appliquée à notre problème cela donne :

$$\mathcal{L}_{\theta_k}(\theta) \approx g^T (\theta - \theta_k) \text{ avec } g = \nabla_{\theta} \mathcal{L}_{\theta_k}(\theta)|_{\theta_k}$$

$$\bar{D}_{KL}(\theta_k || \theta) \approx \frac{1}{2}(\theta - \theta_k)^T F (\theta - \theta_k) \text{ avec } F = \nabla_{\theta}^2 \bar{D}_{KL}(\theta_k || \theta)|_{\theta_k}$$

car $\mathcal{L}_{\theta_k}(\theta_k) = 0$ et $\nabla_{\theta}^2 \mathcal{L}_{\theta_k}(\theta)|_{\theta_k}$ insignifiant par rapport à $\nabla_{\theta} \mathcal{L}_{\theta_k}(\theta)|_{\theta_k}$;
d'autre part $\bar{D}_{KL}(\theta_k || \theta_k) = 0$ et $\nabla_{\theta} \bar{D}_{KL}(\theta_k || \theta)|_{\theta_k} = 0$.

Preuve

La matrice de Fisher F peut être estimée en exploitant :

$$F = \mathbb{E}_{s, a \sim \pi_{\theta_k}} [\nabla_{\theta} \log \pi_{\theta}(a|s)|_{\theta_k} \nabla_{\theta} \log \pi_{\theta}(a|s)|_{\theta_k}^T] \quad \text{Preuve}$$

Notons que dans l'expression du gradient de $J(\theta)$ pris en θ_k , le ratio d'IS disparaît :

$$\begin{aligned} g = \nabla_{\theta} \mathcal{L}_{\theta_k}(\theta)|_{\theta_k} &= \frac{1}{1 - \gamma} \mathbb{E}_{s \sim d^{\pi_{\theta_k}}(s), a \sim \pi_{\theta_k}(a|s)} [\nabla_{\theta} \log \pi_{\theta}(a|s)|_{\theta_k} A^{\pi_{\theta_k}}(s_t, a_t)] \\ &= \nabla_{\theta} J(\theta)|_{\theta_k} \text{ (policy gradient classique)} \end{aligned}$$

TRPO : Trust-Region Policy Optimization

Soit le Lagrangien de notre problème considérant les expansions de Taylor précédentes et une contrainte d'égalité sur la KL (plutôt que l'inégalité $KL \leq \delta$) :

$$L(\theta, \lambda) = g^T(\theta - \theta_k) + \lambda \left(\frac{1}{2}(\theta - \theta_k)^T F(\theta - \theta_k) - \delta \right)$$

Selon les KKT, on a l'optimum :
$$\begin{cases} \nabla_{\theta} L(\theta, \lambda) = 0 \\ \nabla_{\lambda} L(\theta, \lambda) = 0 \end{cases}$$

En exploitant ces conditions, on en vient à la règle de mise à jour suivante :

$$\theta = \theta_k + \beta F^{-1} g$$

avec $\beta = -\frac{1}{\lambda} = \sqrt{\frac{2\delta}{g^T F^{-1} g}}$ correspondant à un pas de gradient.

La matrice de Fisher correspond à un mapping entre l'espace de paramètres et l'espace des politiques. $\tilde{\nabla}_{\theta} J(\theta) = F^{-1} g$ est appelé gradient naturel de $J(\theta)$. Il permet de rendre l'optimisation indépendante de la paramétrisation du modèle.

Natural Policy Gradient

Le gradient d'ordre 1 $\nabla_{\theta} J(\theta)$ donne la direction de plus forte pente, selon un déplacement infinitésimal dans l'espace des paramètres

$$\operatorname{argmax}_d \text{ s.t. } \|d\| \leq \epsilon \mathcal{L}(\theta + d)$$

- ⇒ Dépend de la paramétrisation de la politique
- ⇒ Ne prend pas en compte la "courbure" de l'espace, suppose une surface plate aux alentours de la politique courante
- ⇒ Déplacement infinitésimal dans l'espace des paramètres \neq Déplacement infinitésimal dans l'espace des politiques

Le gradient naturel $\tilde{\nabla}_{\theta} J(\theta) = F^{-1} \nabla_{\theta} J(\theta)$ donne la direction de plus forte pente selon un déplacement infinitésimal dans l'espace des politiques

$$\operatorname{argmax}_d \text{ s.t. } \text{KL}[p_{\theta} \| p_{\theta+d}] \leq \epsilon \mathcal{L}(\theta + d)$$

- ⇒ Déplacement dans l'espace des politiques contrôlé selon une divergence KL avec l'ancienne politique
- ⇒ Indépendant de la paramétrisation

TRPO : Trust-Region Policy Optimization

Algorithm 1 Natural Policy Gradient

Input: initial policy parameters θ_0

for $k = 0, 1, 2, \dots$ **do**

Collect set of trajectories \mathcal{D}_k on policy $\pi_k = \pi(\theta_k)$

Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

Form sample estimates for

- policy gradient \hat{g}_k (using advantage estimates)
- and KL-divergence Hessian / Fisher Information Matrix \hat{H}_k

Compute Natural Policy Gradient update:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{\hat{g}_k^T \hat{H}_k^{-1} \hat{g}_k}} \hat{H}_k^{-1} \hat{g}_k$$

end for

Ok mais... Trouver l'inverse de F (ou H dans l'algo) peut s'avérer trop coûteux, en particulier pour les modèles deep comprenant de très nombreux paramètres.

TRPO : Trust-Region Policy Optimization

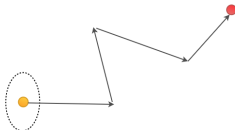
Plutôt que de calculer l'inverse de la matrice de Fisher à chaque étape, on peut chercher à estimer directement : $x_k \approx F_k^{-1} g_k$, en considérant : $g_k \approx F_k x_k$.

Or, chercher à résoudre $Ax = b$ est équivalent à minimiser $f(x) = \frac{1}{2}x^T Ax - b^T x$ selon x : la dérivée de f en x^* vérifie $f'(x^*) = Ax^* - b = 0$.

Cela permet d'exprimer notre problème sous la forme d'une minimisation quadratique :

$$x_k = \operatorname{argmin}_{x \in \mathbb{R}^d} \frac{1}{2}x^T F_k x + g_k^T x$$

Ce genre de problème peut se résoudre en au maximum d étapes pour un problème à d paramètres en utilisant la méthode du gradient conjugué.



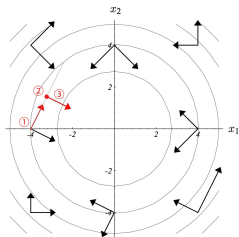
Gradient ascent



Conjugate gradient

*Conjugate Gradient

L'algorithme du gradient conjugué est un algo itératif qui cherche à chaque itération une direction qui est orthogonale à toutes les directions déjà prises, afin de garantir de ne pas "défaire" les progrès apportés par les mises à jour précédentes.



Formellement, cela signifie que la nouvelle direction $d_{(j)}$ à l'étape j doit être conjuguée avec toutes les précédentes par rapport à A (avec A la matrice du problème quadratique à résoudre). Autrement dit, pour tout $d_{(i)}$ avec $i < j$, on doit vérifier pour $d_{(j)}$:

$$d_{(i)}^T A d_{(j)} = 0$$

Une fois la direction trouvée en fonction du résidu du modèle, on cherche le meilleur pas de gradient lui associer de manière analytique.

*Conjugate Gradient

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
 $\mathbf{d}_0 := \mathbf{r}_0$
 $k := 0$
repeat

how far should we move in direction \mathbf{p} $\alpha_k := \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{d}_k^\top \mathbf{A} \mathbf{d}_k}$

the next point $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{d}_k$

a.k.a a remaning error from the optimal point $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{d}_k$

if r_{k+1} is sufficiently small, then exit loop

next direction to go $\beta_k := \frac{\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{r}_k^\top \mathbf{r}_k}$ the new direction will be A-orthogonal

$\mathbf{d}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{d}_k$

$k := k + 1$

end repeat

The result is \mathbf{x}_{k+1}

TRPO : Trust-Region Policy Optimization

Input: initial policy parameters θ_0

for $k = 0, 1, 2, \dots$ **do**

Collect set of trajectories \mathcal{D}_k on policy $\pi_k = \pi(\theta_k)$

Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

Form sample estimates for

- policy gradient \hat{g}_k (using advantage estimates)
- and KL-divergence Hessian-vector product function $f(v) = \hat{H}_k v$

Use CG with n_{cg} iterations to obtain $x_k \approx \hat{H}_k^{-1} \hat{g}_k$

Estimate proposed step $\Delta_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k x_k}} x_k$

$\theta_{k+1} = \theta_k + \Delta_k$

end for

CG bien moins coûteux que l'inversion de la matrice de Fisher.

TRPO : Trust-Region Policy Optimization

Ok mais...

- ▶ On a considéré une contrainte $KL = \delta$ plutôt que $KL \leq \delta$. Il se peut cela implique un déplacement dégradant $\mathcal{L}_{\pi_k}(\pi)$ si la politique optimale se trouve plus proche dans la zone de confiance
 - ▶ L'ensemble des approximations effectuées aux étapes précédentes (+ les erreurs d'estimations de A^π et F) induisent une violation de la contrainte pour des pas trop grands
- ⇒ TRPO effectue une line search itérative (en partant d'une valeur a^0 donnée et en réduisant exponentiellement cette valeur jusqu'à atteindre un déplacement respectant à la fois la contrainte sur la KL et améliorant $\mathcal{L}_{\pi_k}(\pi)$).

Algorithm 2 Line Search for TRPO

Compute proposed policy step $\Delta_k = \sqrt{\frac{2\delta}{\hat{g}_k^T \hat{H}_k^{-1} \hat{g}_k}} \hat{H}_k^{-1} \hat{g}_k$

for $j = 0, 1, 2, \dots, L$ **do**

 Compute proposed update $\theta = \theta_k + \alpha^j \Delta_k$

if $\mathcal{L}_{\theta_k}(\theta) \geq 0$ and $\bar{D}_{KL}(\theta_k || \theta) \leq \delta$ **then**

 accept the update and set $\theta_{k+1} = \theta_k + \alpha^j \Delta_k$

break

end if

end for

TRPO : Trust-Region Policy Optimization

Algorithm 3 Trust Region Policy Optimization

Input: initial policy parameters θ_0

for $k = 0, 1, 2, \dots$ **do**

Collect set of trajectories \mathcal{D}_k on policy $\pi_k = \pi(\theta_k)$

Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

Form sample estimates for

- policy gradient \hat{g}_k (using advantage estimates)
- and KL-divergence Hessian-vector product function $f(v) = \hat{H}_k v$

Use CG with n_{cg} iterations to obtain $x_k \approx \hat{H}_k^{-1} \hat{g}_k$

Estimate proposed step $\Delta_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k x_k}} x_k$

Perform backtracking line search with exponential decay to obtain final update

$$\theta_{k+1} = \theta_k + \alpha^j \Delta_k$$

end for



Déplacements moins risqués que PG classiques



Approximation de la matrice Fisher très coûteuse (nombreux rollouts à réaliser à chaque étape)

TRPO : Trust-Region Policy Optimization

Algorithm 1 Trust Region Policy Optimization

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: Hyperparameters: KL-divergence limit δ , backtracking coefficient α , maximum number of backtracking steps K
- 3: **for** $k = 0, 1, 2, \dots$ **do**
- 4: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 5: Compute rewards-to-go \hat{R}_t .
- 6: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 7: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) |_{\theta_k} \hat{A}_t.$$

- 8: Use the conjugate gradient algorithm to compute

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k,$$

where \hat{H}_k is the Hessian of the sample average KL-divergence.

- 9: Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k,$$

where $j \in \{0, 1, 2, \dots, K\}$ is the smallest value which improves the sample loss and satisfies the sample KL-divergence constraint.

- 10: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left(V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 11: **end for**

*Gradient Naturel avec Fonctions Compatibles

Plutôt que d'utiliser une estimation de A^π pour le gradient, on peut utiliser une approximation f_ϕ , avec f_ϕ une fonction compatible : $f_\phi(s, a) = (\nabla_\theta \log \pi_\theta(a|s))^T \phi$

Si on a :
$$\sum_{\tau} \pi_{\theta}(\tau) \left[\sum_{t=0}^{|\tau|-1} (Q^{\pi}(s_t, a_t) - f_{\phi}(s_t, a_t) - v_w(s_t)) \nabla_{\phi} f_{\phi}(s_t, a_t) \right] = 0$$

Alors :
$$\nabla_{\theta} J(\theta) = F \phi$$

Ou encore :
$$\tilde{\nabla}_{\theta} J(\theta) = \phi \quad \text{Preuve}$$

Suivant une approche Natural Policy Gradient, [GP10] suggère alors les règles de mises à jour suivantes à chaque étape t de chaque trajectoire (Natural Policy Gradient incrémental) : $\delta_t = r_t + \gamma(f_{\phi}(s_{t+1}, a_{t+1}) + v_w(s_{t+1})) - f_{\phi}(s_t, a_t) - v_w(s_t)$

$$\phi \leftarrow \phi + \alpha_i \delta_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

$$w \leftarrow w + \alpha_i \delta_t \nabla_w v_w(s_t)$$

$$\theta \leftarrow \theta + \beta_i \phi$$

Avec : $\sum \alpha_i = \sum \beta_i = \infty \quad \sum \alpha_i^2 = \sum \beta_i^2 < \infty \quad \lim_{i \rightarrow \infty} \frac{\beta_i}{\alpha_i} = 0$
(Convergence plus rapide de la critique que de l'acteur)

Version Trust-Region incrémentale ?
$$\theta \leftarrow \theta + \beta_i \frac{\sqrt{2\delta}}{|f_{\phi}(s_t, a_t)|} \phi$$



Pas de calcul de la matrice de Fisher



Risque d'instabilité (difficile de régler des bons pas d'apprentissage) [IA19]

*Gradient Naturel avec Fonctions Compatibles

Version Trust-Region par Batches avec critère d'entropie [Paj+19] (où $\tilde{G}_w^{\pi^{old}}(s, a)$ correspond à notre $f_\phi(s, a)$) :

Algorithm 2 COPOS discrete actions

Initialize policy network $\pi_{\theta, \beta}$ with non-linear parameters β and linear parameters θ and $\Theta = (\theta, \beta)$

for episode $\leftarrow 1$ **to** MaxEpisode **do**

 Initialize empty batch \mathcal{B}

while collected samples $<$ batchsize **do**

 Run policy $\pi_{\theta, \beta}(a|s)$ for T timesteps or until termination: Draw action $a_t \sim \pi_{\theta, \beta}(a_t|s_t)$,

 observe reward r_t

 Add samples (s_t, a_t, r_t) to \mathcal{B}

end while

 Compute advantage values $A^{\pi_{old}}(s_i, a_i)$

 Compute $w = (w_\theta, w_\beta)$ using conjugate gradient to solve

$$w = F^{-1} \nabla_{\Theta} J_{PG}(\pi_{\Theta})|_{\Theta=\Theta_{old}} \quad \nabla_{\Theta} J_{PG}(\pi_{\Theta}) = \sum_i^{|B|} \nabla_{\Theta} \log \pi_{\Theta}(a_i|s_i) A^{\pi_{old}}(s_i, a_i)$$

 Use $\tilde{G}_w^{\pi_{old}}(s, a)$ to solve for $\eta > 0$ and $\omega > 0$ using the dual to the corresponding trust region optimization problem:

$$\begin{aligned} & \operatorname{argmax}_{\pi_{\theta}} \mathbb{E}_{s \sim p(s)} \left[\int \pi_{\theta}(a|s) \tilde{G}_w^{\pi_{old}}(s, a) da \right] \\ & \text{subject to } \mathbb{E}_{s \sim p(s)} [KL(\pi_{\theta}(\cdot|s) || \pi_{\Theta_{old}}(\cdot|s))] < \epsilon \\ & \quad \mathbb{E}_{s \sim p(s)} [H(\pi_{\theta}(\cdot|s)) - H(\pi_{\Theta_{old}}(\cdot|s))] < \beta \end{aligned}$$

Apply updates for the new policy:

$$\theta_{new} = \frac{\eta \theta_{old} + w_{\theta}}{\eta + \omega} \quad \beta_{new} = \beta_{old} + s \frac{w_{\beta}}{\eta}$$

where s is a rescaling factor found by line search

end for



Meilleure stabilité (exploite la fonction d'approximation de l'avantage)



Meilleure exploration (terme d'entropie qui l'empêche de réduire trop rapidement)

* ACKTR : Actor Critic using Kronecker-Factored Trust Region

Kronecker-factored Approximate Curvature (K-FAC)

- ▶ Le calcul global d'une matrice Hessienne telle que la matrice de Fisher est de l'ordre de $O(n^2)$ avec n le nombre de paramètres du réseau
 - ▶ Impossible sur des réseaux Deep comprenant des centaines de milliers de paramètres
 - ▶ Méthodes du second-ordre plus efficaces en théorie mais la complexité de ce calcul ne permet généralement pas de justifier de leur utilisation
- ▶ K-FAC approxime l'inverse de la matrice de Fisher un layer à la fois, ce qui réduit considérablement la complexité.

let $p(y|x)$ denote the output distribution of a neural network

$L = \log p(y|x)$ denote the log-likelihood

$W \in \mathbb{R}^{C_{out} \times C_{in}}$ be the weight matrix in the ℓ^{th} layer

$a \in \mathbb{R}^{C_{in}}$ be the the input activation vector

$s = Wa$

Appliqué sur chaque layer l , le calcul de la matrice de Fisher pour les paramètres W de l devient :

$$F_l = \mathbb{E}[\text{vec}\{\nabla_W L\} \text{vec}\{\nabla_W L\}^T]$$

avec $\nabla_W L = (\nabla_s L) a^T$ (backpropagation chain-rule)

* ACKTR : Actor Critic using Kronecker-Factored Trust Region

ACKTR [Wu+17] exploite cette idée en considérant :

$$\begin{aligned} F_\ell &= \mathbb{E}[\text{vec}\{\nabla_W L\} \text{vec}\{\nabla_W L\}^\top] = \mathbb{E}[aa^\top \otimes \nabla_s L(\nabla_s L)^\top] \\ &\approx \mathbb{E}[aa^\top] \otimes \mathbb{E}[\nabla_s L(\nabla_s L)^\top] := A \otimes S := \hat{F}_\ell, \\ &\text{where } A \text{ denotes } \mathbb{E}[aa^\top] \text{ and } S \text{ denotes } \mathbb{E}[\nabla_s L(\nabla_s L)^\top]. \end{aligned}$$

avec \otimes le produit de Kronecker entre deux matrices : $A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}$

⇒ Approximation fait l'hypothèse que les statistiques du second-ordre des activations et du gradient de la sortie ne sont pas corrélées

* ACKTR : Actor Critic using Kronecker-Factored Trust Region

En notant les propriétés du produit de Kronecker : $(P \otimes Q)^{-1} = P^{-1} \otimes Q^{-1}$ et $(P \otimes Q)\text{vec}\{T\} = \text{vec}\{PTQ^T\}$, on obtient le gradient naturel selon W :

$$\text{vec}\{\tilde{\nabla}_W J\} = F_I^{-1} \text{vec}\{\nabla_W J\} = \text{vec}\{A^{-1} \nabla_W JS^{-1}\}$$

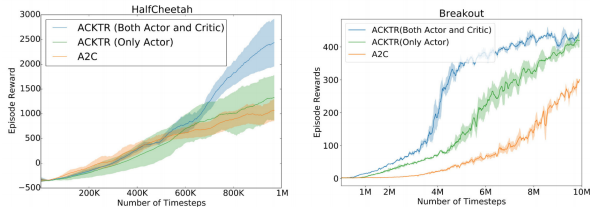
On a alors la règle d'update : $W \leftarrow W + \beta \tilde{\nabla}_W J$, avec :

$$\beta = \min(\beta_{\max}, \sqrt{\frac{2\delta}{\text{vec}\{\nabla_W J\}^T F_I^{-1} \text{vec}\{\nabla_W J\}}}) = \min(\beta_{\max}, \sqrt{\frac{2\delta}{\text{vec}\{\nabla_W J\}^T \text{vec}\{\tilde{\nabla}_W J\}}})$$

- ▶ ACKTR estime A et S par moyennes mobiles
- ▶ Considérant la critique v comme issue d'une gaussienne $\mathcal{N}(V_\phi(s), 1)$, K-FAC applicable également pour l'optimisation de la critique
- ▶ Pour les layers dont les paramètres sont partagés entre l'acteur et la critique, on suppose l'indépendance conditionnelle : $p(a, v|s) = p(a|s)p(v|s)$, puis on applique K-FAC en considérant :

$$F_I = \mathbb{E}[\text{vec}\{\nabla_W \log(p(a, v|s))\} \text{vec}\{\nabla_W \log(p(a, v|s))\}^T]$$

* ACKTR : Actor Critic using Kronecker-Factored Trust Region



Domain	Human level	ACKTR		A2C		TRPO (10 M)	
		Rewards	Episode	Rewards	Episode	Rewards	Episode
Beamrider	5775.0	13581.4	3279	8148.1	8930	670.0	N/A
Breakout	31.8	735.7	4094	581.6	14464	14.7	N/A
Pong	9.3	20.9	904	19.9	4768	-1.2	N/A
Q-bert	13455.0	21500.3	6422	15967.4	19168	971.8	N/A
Seaquest	20182.0	1776.0	N/A	1754.0	N/A	810.4	N/A
Space Invaders	1652.0	19723.0	14696	1757.2	N/A	465.1	N/A

Table 1: ACKTR and A2C results showing the last 100 average episode rewards attained after 50 million timesteps, and TRPO results after 10 million timesteps. The table also shows the episode N , where N denotes the first episode for which the mean episode reward over the N^{th} game to the $(N + 100)^{th}$ game crosses the human performance level [17], averaged over 2 random seeds.



Bonnes performances, quasi aussi rapide que A2C, meilleure stabilité



Pas vraiment applicable sur des architectures type RNN

PPO : Proximal Policy Optimization

Plutôt qu'un problème d'optimisation sous contraintes, qui implique des méthodes d'optimisation du second ordre complexes, PPO considère [Sch+17] :

$$\theta_{k+1} = \arg \max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\pi_{\theta_k} || \pi_{\theta})$$

avec :

- ▶ $\mathcal{L}_{\theta_k}(\theta) = \mathbb{E}_{(s,a) \sim \pi_{\theta_k}} \left[\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)} \hat{A}^{\pi_{\theta_k}}(s, a) \right]$
 - ▶ $\bar{D}_{KL}(\theta_k | \theta) = \mathbb{E}_{(s,a) \sim \pi_{\theta_k}} [D_{KL}(\pi_{\theta_k}(\cdot|s) | \pi_{\theta}(\cdot|s))]$
 - ▶ β_k un poids adaptatif qui augmente lors de changements de politique trop brutaux et diminue lorsque les changements sont trop faibles.
- ⇒ Hypothèse : il est possible de se permettre de mauvais déplacements de temps en temps, compensés par le gain de rapidité des méthodes du premier ordre
- ⇒ Maximisation réalisée en K étapes de montée de gradient
- ▶ Que vaut la KL lors du premier calcul de gradient (1ère étape) ?
 - ▶ À quoi revient-on si on considère $K = 1$?
- ▶ À noter qu'il est aussi possible de considérer la KL en mode reverse : $D_{KL}(\pi_{\theta}(\cdot|s) | \pi_{\theta_k}(\cdot|s))$ plutôt que $D_{KL}(\pi_{\theta_k}(\cdot|s) | \pi_{\theta}(\cdot|s))$
 - ▶ TRPO pourrait aussi être défini en reverse KL (voir Preuves)
 - ▶ [HMDH20] montre que PPO avec KL reverse légèrement supérieur

PPO : Proximal Policy Optimization

Algorithme 1 : Algorithme PPO - version KL adaptatif

Input : Paramètres initiaux θ_0 et ϕ , KL cible δ , pas d'apprentissage α , nombre d'étapes d'optimisation K

```
1  $\beta_0 \leftarrow 1$ 
2 for  $k = 0, 1, 2, \dots$  do
3   Collecte d'un ensemble de trajectoires  $\mathcal{D}_k$  selon politique  $\pi_{\theta_k}$ 
4   Calcul des avantages  $\hat{A}^{\pi_{\theta_k}}$  pour toutes les transitions de  $\mathcal{D}_k$ 
   selon  $TD(\lambda)$ 
5    $\theta \leftarrow \theta_k$ 
6   for  $s$  de 1 à  $K$  do
7      $\theta \leftarrow \theta + \alpha (\nabla_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \nabla_{\theta} \bar{D}_{KL}(\theta_k | \theta))$ 
8     avec  $\mathcal{L}_{\theta_k}(\theta) = \frac{1}{|\mathcal{D}_k|} \sum_{(s,a) \in \mathcal{D}_k} \frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)} \hat{A}^{\pi_{\theta_k}}(s, a)$ 
9     et  $\bar{D}_{KL}(\theta_k | \theta) = \frac{1}{|\mathcal{D}_k|} \sum_{s \in \mathcal{D}_k} D_{KL}(\pi_{\theta_k}(\cdot | s) | \pi_{\theta}(\cdot | s))$ .
10  end
11   $\theta_{k+1} \leftarrow \theta$ 
12  if  $\bar{D}_{KL}(\theta_k | \theta_{k+1}) \geq 1.5\delta$  then
13     $\beta_{k+1} \leftarrow 2\beta_k$ 
14  end
15  if  $\bar{D}_{KL}(\theta_k | \theta_{k+1}) \leq \delta/1.5$  then
16     $\beta_{k+1} \leftarrow 0.5\beta_k$ 
17  end
18  Mise à jour de  $V_{\phi}$  selon  $TD(\lambda)$  sur  $\mathcal{D}_k$ 
19 end
```

👍 + **sample efficient** : Chaque trajectoire utilisée K fois (plutôt que 1 en PG classique)

PPO : Proximal Policy Optimization

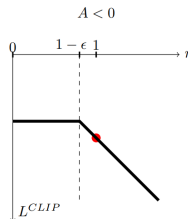
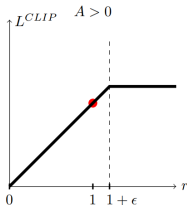
PPO propose une autre solution pour le contrôle du déplacement de la politique,

considérant le ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$

- ▶ Lorsque la modification améliore la fonction avantage, on empêche le ratio $r_t(\theta)$ de lui donner trop de poids
 - ▶ Lorsque la modification dégrade la fonction avantage, on conserve le ratio $r_t(\theta)$ suffisamment élevé pour éviter ce déplacement
- ⇒ On souhaite favoriser les déplacements dans une zone de confort, favorable pour la fonction d'avantage

$$\theta_{k+1} = \arg \max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

avec :
$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathbb{E}_{\tau \sim \pi_k} \left[\sum_{t=0}^T \left[\min(r_t(\theta) \hat{A}_t^{\pi_k}, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t^{\pi_k}) \right] \right]$$



PPO : Proximal Policy Optimization

Pour mieux comprendre, prenons l'exemple d'une action pour laquelle on a modifié la probabilité à l'étape précédente :

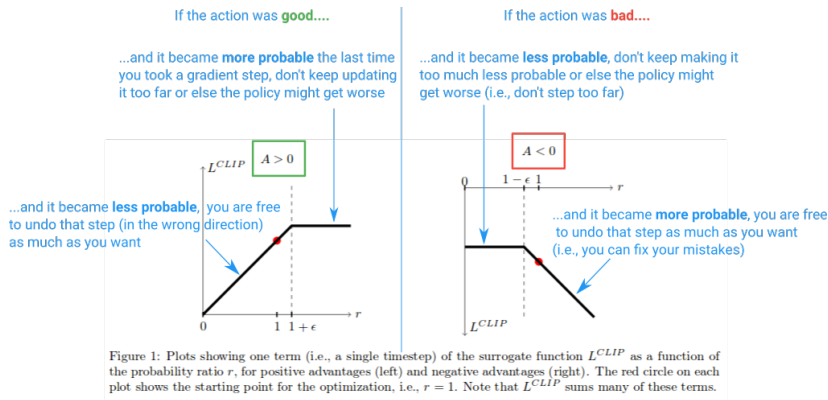


Image issue de :

<https://stackoverflow.com/questions/46422845/>

what-is-the-way-to-understand-proximal-policy-optimization-algorithm-in-rl/50663200#50663200

PPO : Proximal Policy Optimization

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0 , clipping threshold ϵ

for $k = 0, 1, 2, \dots$ **do**

Collect set of partial trajectories \mathcal{D}_k on policy $\pi_k = \pi(\theta_k)$

Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

Compute policy update

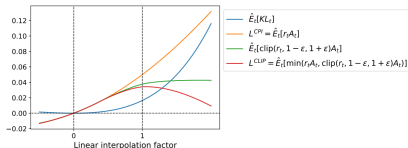
$$\theta_{k+1} = \arg \max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathbb{E}_{\tau \sim \pi_k} \left[\sum_{t=0}^T \left[\min(r_t(\theta) \hat{A}_t^{\pi_k}, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t^{\pi_k}) \right] \right]$$

end for

Après un pas de modification selon PPO-clipped (avec $\epsilon = 0.02$), interpolation entre θ_k (0) et θ_{k+1} (1) :



► Lowerbound (pessimiste) de $r_t(\theta) \hat{A}_t$

- plus on va loin plus on risque d'avoir de forts ratios IS
- on ne garde quasi que les gradients de ratios qui vont dans le mauvais sens pour \hat{A}_t : tend à rapprocher du θ initial

⇒ Empêche une trop grande mise à jour de θ

PPO : Proximal Policy Optimization

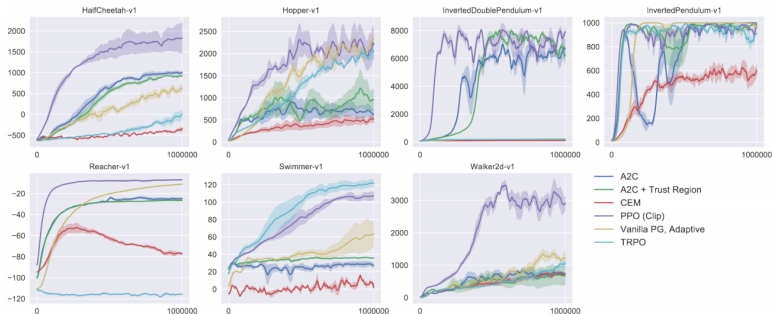


Figure: Performance comparison between PPO with clipped objective and various other deep RL methods on a slate of MuJoCo tasks.

- 👍 Résultats comparables à TRPO (voire meilleurs)
- 👍 Beaucoup plus simple et plus rapide
- Ajout d'un terme d'entropie semble largement stabiliser l'algo

- ▶ Sergey Levine (UC Berkeley, Spring 2017)
- ▶ Daniel Takeshi :
<https://danieltakeshi.github.io/2017/03/28/going-deeper-into-reinforcement-learning-fundamentals-o>
- ▶ Jonathan Hui : https://medium.com/@jonathan_hui/r1-deep-reinforcement-learning-series-833319a95530
- ▶ Lilian Weng : <https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html>
- ▶ Felix Yu :
<https://flyyufelix.github.io/2017/10/12/dqn-vs-pg.html>
- ▶ Joshua Achiam :
http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_13_advanced_pg.pdf

- ▶ Nathan Ratliff : http://ipvs.informatik.uni-stuttgart.de/mlr/wp-content/uploads/2015/01/mathematics_for_intelligent_systems_lecture12_notes_I.pdf
- ▶ OpenAI : <https://spinningup.openai.com/en/latest/algorithms/trpo.html>

References I

- [Bha+09] Shalabh Bhatnagar et al. « Natural actor–critic algorithms ». In : *Automatica* 45.11 (2009), p. 2471-2482.
- [GP10] Matthieu Geist et Olivier Pietquin. « Revisiting natural actor-critics with value function approximation ». In : *International conference on modeling decisions for artificial intelligence*. Springer. 2010, p. 207-218.
- [Gro+12] Ivo Grondman et al. « A survey of actor-critic reinforcement learning : Standard and natural policy gradients ». In : *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012), p. 1291-1307.
- [HMDH20] Chloe Ching-Yun Hsu, Celestine Mender-Dünner et Moritz Hardt. « Revisiting Design Choices in Proximal Policy Optimization ». In : *arXiv preprint arXiv :2009.10897* (2020).
- [IA19] Ryo Iwaki et Minoru Asada. « Implicit incremental natural actor critic algorithm ». In : *Neural Networks* 109 (2019), p. 103 -112.
- [Paj+19] Joni Pajarinen et al. « Compatible natural gradient policy search ». In : *Machine Learning* (2019), p. 1-24.
- [Sch+15] John Schulman et al. « Trust Region Policy Optimization ». In : *CoRR* abs/1502.05477 (2015). arXiv : 1502.05477.

References II

- [Sch+17] John Schulman et al. « Proximal policy optimization algorithms ». In : *arXiv preprint arXiv :1707.06347* (2017).
- [Wu+17] Yuhuai Wu et al. « Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation ». In : *Advances in neural information processing systems*. 2017, p. 5279-5288.