Multi-Agent RL, PPO, MAPPO

Milan Straka

January 02, 2023

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Multi-Agent Reinforcement Learning

We use the thesis

Cooperative Multi-Agent Reinforcement Learning
https://dspace.cuni.cz/handle/20.500.11956/127431

as an introduction text.

Natural Policy Gradient

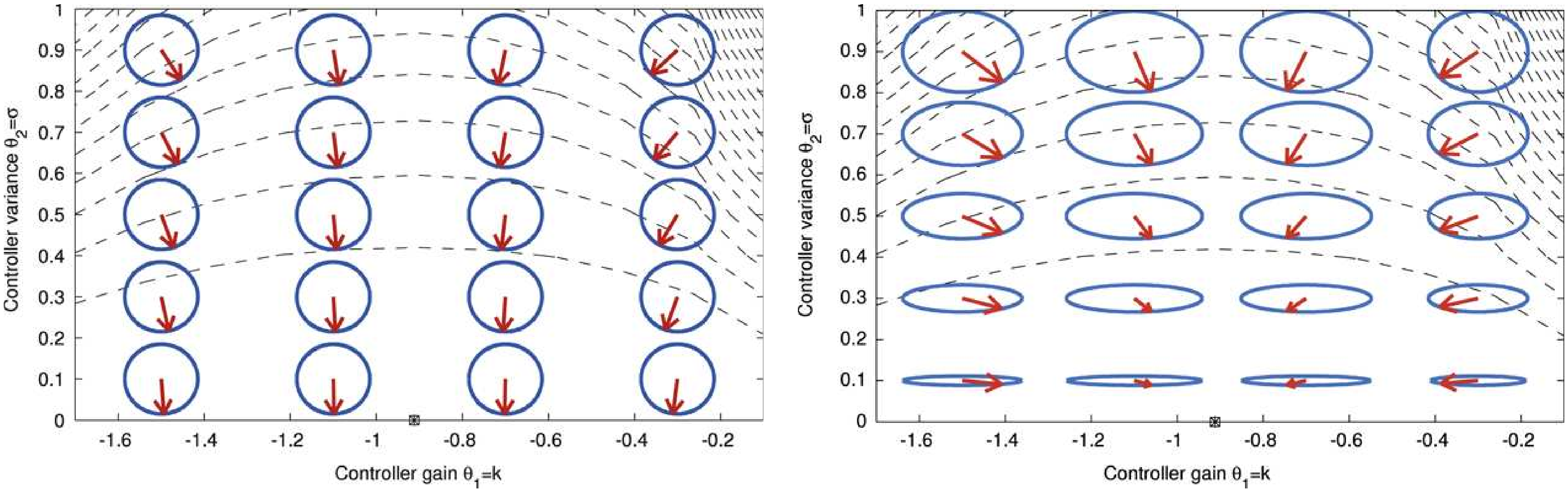
The following approach has been introduced by Kakade (2002).

Using policy gradient theorem, we are able to compute ∇v_π. Normally, we update the parameters by using directly this gradient. This choice is justified by the fact that a vector $\boldsymbol d$ which maximizes $v_\pi(s; \boldsymbol\theta + \boldsymbol d)$ under the constraint that $\|\boldsymbol d\|^2$ is bounded by a small constant is exactly the gradient ∇v_π.

Normally, the length $|\boldsymbol d|^2$ is computed using Euclidean metric. But in general, any metric could be used. Representing a metric using a positive-definite matrix $\boldsymbol G$ (identity matrix for Euclidean metric), we can compute the distance as $\|\boldsymbol d\|^2_{\boldsymbol G} = \sum_{ij} G_{ij} d_i d_j = \boldsymbol d^T \boldsymbol G \boldsymbol d$. The steepest ascent direction is then given by $\boldsymbol G^{-1} \nabla v_\pi$.

Note that when $\boldsymbol G$ is the Hessian $\boldsymbol H v_\pi$, the above process is exactly Newton's method.

Natural Policy Gradient



(a) Vanilla policy gradient.

(b) Natural policy gradient.

Figure 3 of "Reinforcement learning of motor skills with policy gradients" by Jan Peters et al.

Natural Policy Gradient

A suitable choice for the metric is *Fisher information matrix*, which is defined as a *covariance matrix* of the **score function** $\nabla_{\boldsymbol\theta} \log \pi(a|s;\boldsymbol\theta)$. The expectaton of the score function is 0:

$$\mathbb{E}_{\pi(a|s;\boldsymbol\theta)} \nabla_{\boldsymbol\theta} \log \pi(a|s;\boldsymbol\theta) = \int \pi(a|s;\boldsymbol\theta) \frac{\nabla_{\boldsymbol\theta} \pi(a|s;\boldsymbol\theta)}{\pi(a|s;\boldsymbol\theta)} \,\mathrm{d}a = \nabla_{\boldsymbol\theta} \int \pi(a|s;\boldsymbol\theta) \,\mathrm{d}a = \nabla_{\boldsymbol\theta} 1 = 0.$$

The Fisher information matrix is therefore

$$F_s(\boldsymbol\theta) \stackrel{\text{def}}{=} \mathbb{E}_{\pi(a|s;\boldsymbol\theta)} \Big[\big(\nabla_{\boldsymbol\theta} \log \pi(a|s;\boldsymbol\theta)\big) \big(\nabla_{\boldsymbol\theta} \log \pi(a|s;\boldsymbol\theta)\big)^T\Big].$$

It can be shown that the Fisher information metric is the only Riemannian metric (up to rescaling) invariant to change of parameters under sufficient statistic.

The Fisher information matrix is also a Hessian of the $D_\mathrm{KL}(\pi(a|s;\boldsymbol\theta) \| \pi(a|s;\boldsymbol\theta'))$:

$$F_s(\boldsymbol\theta) = \frac{\partial^2}{\partial\theta'_i \partial\theta'_j} D_\mathrm{KL}\big(\pi(a|s;\boldsymbol\theta) \| \pi(a|s;\boldsymbol\theta')\big)\Big|_{\boldsymbol\theta'=\boldsymbol\theta}.$$

Natural Policy Gradient

Using the metric

$$F(\boldsymbol\theta) = \mathbb{E}_{s\sim\mu_{\boldsymbol\theta}} F_s(\boldsymbol\theta)$$

we want to update the parameters using $\boldsymbol d_F \stackrel{\text{def}}{=} F(\boldsymbol\theta)^{-1}\nabla v_\pi$.

An interesting property of using the $\boldsymbol d_F$ to update the parameters is that

- updating $\boldsymbol\theta$ using ∇v_π will choose an arbitrary *better* action in state s;
- updating $\boldsymbol\theta$ using $F(\boldsymbol\theta)^{-1}\nabla v_\pi$ chooses the *best* action (maximizing expected return), similarly to tabular greedy policy improvement.

However, computing $\boldsymbol d_F$ in a straightforward way is too costly.

Truncated Natural Policy Gradient

Duan et al. (2016) in paper *Benchmarking Deep Reinforcement Learning for Continuous Control* propose a modification to the NPG to efficiently compute $\boldsymbol{d}_F$.

Following Schulman et al. (2015), they suggest to use *conjugate gradient algorithm*, which can solve a system of linear equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ in an iterative manner, by using $\boldsymbol{A}$ only to compute products $\boldsymbol{A}\boldsymbol{v}$ for a suitable $\boldsymbol{v}$.

Therefore, $\boldsymbol{d}_F$ is found as a solution of

$$\boldsymbol{F}(\boldsymbol{\theta})\boldsymbol{d}_F = \nabla v_\pi$$

and using only 10 iterations of the algorithm seem to suffice according to the experiments.

Furthermore, Duan et al. suggest to use a specific learning rate suggested by Peters et al (2008) of

$$\frac{\alpha}{\sqrt{(\nabla v_\pi)^T \boldsymbol{F}(\boldsymbol{\theta})^{-1} \nabla v_\pi}}.$$

Schulman et al. in 2015 wrote an influential paper introducing TRPO as an improved variant of NPG.

Considering two policies $\pi, \tilde\pi$, we can write

$$v_{\tilde\pi} = v_\pi + \mathbb{E}_{s\sim\mu(\tilde\pi)}\mathbb{E}_{a\sim\tilde\pi(a|s)} a_\pi(a|s),$$

where $a_\pi(a|s)$ is the advantage function $q_\pi(a|s) - v_\pi(s)$ and $\mu(\tilde\pi)$ is the on-policy distribution of the policy $\tilde\pi$.

Analogously to policy improvement, we see that if $a_\pi(a|s) \geq 0$, policy $\tilde\pi$ performance increases (or stays the same if the advantages are zero everywhere).

However, sampling states $s \sim \mu(\tilde\pi)$ is costly. Therefore, we instead consider

$$L_\pi(\tilde\pi) = v_\pi + \mathbb{E}_{s\sim\mu(\pi)}\mathbb{E}_{a\sim\tilde\pi(a|s)} a_\pi(a|s).$$

Trust Region Policy Optimization

$$L_\pi(\tilde\pi) = v_\pi + \mathbb{E}_{s\sim\mu(\pi)}\mathbb{E}_{a\sim\tilde\pi(a|s)}a_\pi(a|s)$$

It can be shown that for parametrized $\pi(a|s;\boldsymbol\theta)$ the $L_\pi(\tilde\pi)$ matches $v_{\tilde\pi}$ to the first order.

Schulman et al. additionally proves that if we denote $\alpha = D_\mathrm{KL}^\mathrm{max}(\pi_\mathrm{old}\|\pi_\mathrm{new}) = \max_s D_\mathrm{KL}\big(\pi_\mathrm{old}(\cdot|s)\|\pi_\mathrm{new}(\cdot|s)\big)$, then

$$v_{\pi_\mathrm{new}} \geq L_{\pi_\mathrm{old}}(\pi_\mathrm{new}) - \frac{4\varepsilon\gamma}{(1-\gamma)^2}\alpha \quad \text{where} \quad \varepsilon = \max_{s,a}|a_\pi(s,a)|.$$

Therefore, TRPO maximizes $L_{\pi_{\boldsymbol\theta_0}}(\pi_{\boldsymbol\theta})$ subject to $D_\mathrm{KL}^{\boldsymbol\theta_0}(\pi_{\boldsymbol\theta_0}\|\pi_{\boldsymbol\theta}) < \delta$, where

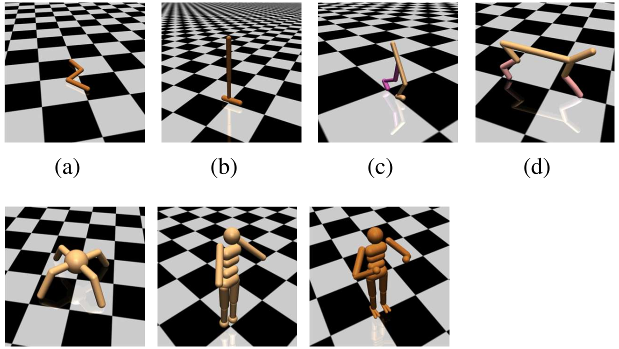
- $D_\mathrm{KL}^{\boldsymbol\theta_0}(\pi_{\boldsymbol\theta_0}\|\pi_{\boldsymbol\theta}) = \mathbb{E}_{s\sim\mu(\pi_{\boldsymbol\theta_0})}\big[D_\mathrm{KL}\big(\pi_\mathrm{old}(\cdot|s)\|\pi_\mathrm{new}(\cdot|s)\big)\big]$ is used instead of $D_\mathrm{KL}^\mathrm{max}$ for performance reasons;
- δ is a constant found empirically, as the one implied by the above equation is too small;
- importance sampling is used to account for sampling actions from π.

Trust Region Policy Optimization

$$\text{maximize } L_{\pi_{\boldsymbol\theta_0}}(\pi_{\boldsymbol\theta}) = \mathbb{E}_{s\sim\mu(\pi_{\boldsymbol\theta_0}),a\sim\pi_{\boldsymbol\theta_0}(a|s)}\left[\frac{\pi_{\boldsymbol\theta}(a|s)}{\pi_{\boldsymbol\theta_0}(a|s)}a_{\pi_{\boldsymbol\theta_0}}(a|s)\right] \text{ subject to } D_{\mathrm{KL}}^{\boldsymbol\theta_0}(\pi_{\boldsymbol\theta_0}\|\pi_{\boldsymbol\theta}) < \delta$$

The parameters are updated using $\boldsymbol{d}_F = F(\boldsymbol\theta)^{-1}\nabla L_{\pi_{\boldsymbol\theta_0}}(\pi_{\boldsymbol\theta})$, utilizing the conjugate gradient algorithm as described earlier for TNPG (note that the algorithm was designed originally for TRPO and only later employed for TNPG).

To guarantee improvement and respect the D_{KL} constraint, a line search is in fact performed. We start by the learning rate of $\sqrt{\delta/(\boldsymbol{d}_F^T F(\boldsymbol\theta)^{-1}\boldsymbol{d}_F)}$ and shrink it exponentially until the constraint is satistifed and the objective improves.



(a) (b) (c) (d) (e) (f) (g)

Figure 1. Illustration of locomotion tasks: (a) Swimmer; (b) Hopper; (c) Walker; (d) Half-Cheetah; (e) Ant; (f) Simple Humanoid; and (g) Full Humanoid.

Figure 1 of "Benchmarking Deep Reinforcement Learning for Continuous Control" by Duan et al.

Task	Random	REINFORCE	TNPG	RWR	REPS	TRPO	CEM	CMA-ES	DDPG
Cart-Pole Balancing	77.1 ± 0.0	4693.7 ± 14.0	**3986.4 ± 748.9**	**4861.5 ± 12.3**	565.6 ± 137.6	**4869.8 ± 37.6**	4815.4 ± 4.8	2440.4 ± 568.3	4634.4 ± 87.8
Inverted Pendulum*	−153.4 ± 0.2	13.4 ± 18.0	**209.7 ± 55.5**	84.7 ± 13.8	−113.3 ± 4.6	**247.2 ± 76.1**	38.2 ± 25.7	−40.1 ± 5.7	40.0 ± 244.6
Mountain Car	−415.4 ± 0.0	−67.1 ± 1.0	**-66.5 ± 4.5**	−79.4 ± 1.1	−275.6 ± 166.3	**-61.7 ± 0.9**	−66.0 ± 2.4	−85.0 ± 7.7	−288.4 ± 170.3
Acrobot	−1904.5 ± 1.0	−508.1 ± 91.0	−395.8 ± 121.2	−352.7 ± 35.9	−1001.5 ± 10.8	−326.0 ± 24.4	−436.8 ± 14.7	−785.6 ± 13.1	**-223.6 ± 5.8**
Double Inverted Pendulum*	149.7 ± 0.1	4116.5 ± 65.2	**4455.4 ± 37.6**	3614.8 ± 368.1	446.7 ± 114.8	**4412.4 ± 50.4**	2566.2 ± 178.9	1576.1 ± 51.3	2863.4 ± 154.0
Swimmer*	−1.7 ± 0.1	92.3 ± 0.1	**96.0 ± 0.2**	60.7 ± 5.5	3.8 ± 3.3	**96.0 ± 0.2**	68.8 ± 2.4	64.9 ± 1.4	85.8 ± 1.8
Hopper	8.4 ± 0.0	714.0 ± 29.3	**1155.1 ± 57.9**	553.2 ± 71.0	86.7 ± 17.6	**1183.3 ± 150.0**	63.1 ± 7.8	20.3 ± 14.3	267.1 ± 43.5
2D Walker	−1.7 ± 0.0	506.5 ± 78.8	**1382.6 ± 108.2**	136.0 ± 15.9	−37.0 ± 38.1	**1353.8 ± 85.0**	84.5 ± 19.2	77.1 ± 24.3	318.4 ± 181.6
Half-Cheetah	−90.8 ± 0.3	1183.1 ± 69.2	**1729.5 ± 184.6**	376.1 ± 28.2	34.5 ± 38.0	**1914.0 ± 120.1**	330.4 ± 274.8	441.3 ± 107.6	**2148.6 ± 702.7**
Ant*	13.4 ± 0.7	548.3 ± 55.5	**706.0 ± 127.7**	37.6 ± 3.1	39.0 ± 9.8	**730.2 ± 61.3**	49.2 ± 5.9	17.8 ± 15.5	326.2 ± 20.8
Simple Humanoid	41.5 ± 0.2	128.1 ± 34.0	**255.0 ± 24.5**	93.3 ± 17.4	28.3 ± 4.7	**269.7 ± 40.3**	60.6 ± 12.9	28.7 ± 3.9	99.4 ± 28.1
Full Humanoid	13.2 ± 0.1	262.2 ± 10.5	**288.4 ± 25.2**	46.7 ± 5.6	41.7 ± 6.1	**287.0 ± 23.4**	36.9 ± 2.9	N/A ± N/A	119.0 ± 31.2

Table 1 of "Benchmarking Deep Reinforcement Learning for Continuous Control" by Duan et al.

Proximal Policy Optimization

A simplification of TRPO which can be implemented using a few lines of code.

Let $r_t(\boldsymbol\theta) \stackrel{\text{def}}{=} \frac{\pi(A_t|S_t;\boldsymbol\theta)}{\pi(A_t|S_t;\boldsymbol\theta_\text{old})}$. PPO maximizes the objective (i.e., you should minimize its negation)

$$L^\text{CLIP}(\boldsymbol\theta) \stackrel{\text{def}}{=} \mathbb{E}_t\Big[\min\big(r_t(\boldsymbol\theta)\hat A_t, \operatorname{clip}(r_t(\boldsymbol\theta), 1-\varepsilon, 1+\varepsilon)\hat A_t\big)\Big].$$

Such a $L^\text{CLIP}(\boldsymbol\theta)$ is a lower (pessimistic) bound.

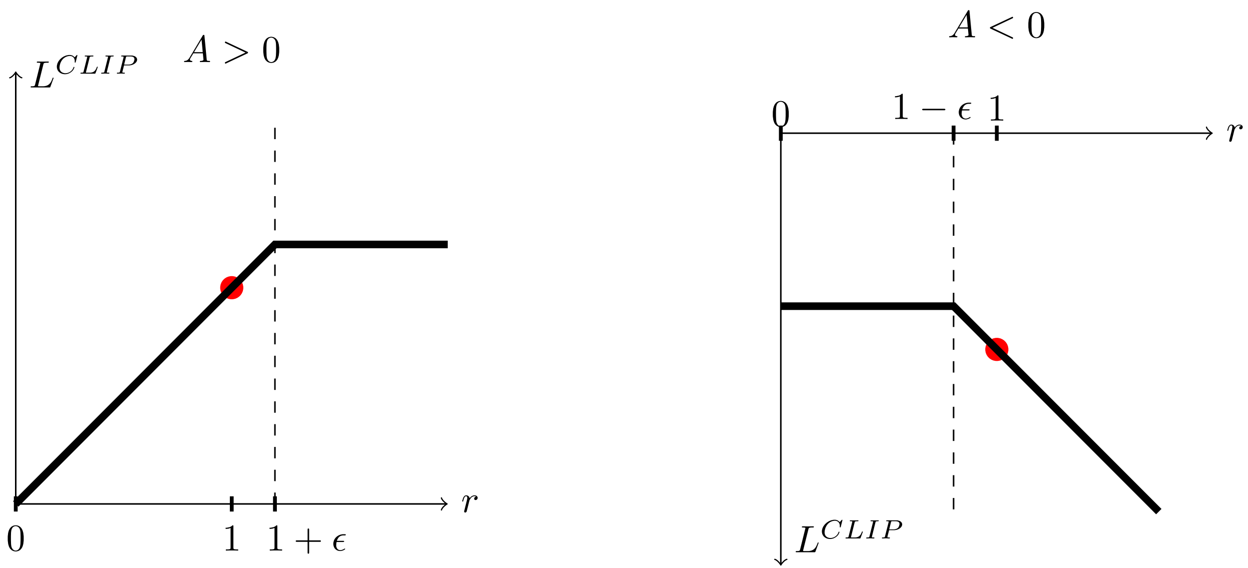


Figure 1 of "Proximal Policy Optimization Algorithms" by Schulman et al.

Proximal Policy Optimization

The advantages $\hat{A}_t$ are additionally estimated using the so-called *generalized advantage estimation*, which is just an analogue of the truncated n-step lambda-return:

$$\hat{A}_t = \sum_{i=0}^{n-1} \gamma^i \lambda^i \big(R_{t+1+i} + \gamma V(S_{t+i+1}) - V(S_{t+i})\big).$$

Algorithm 1 PPO, Actor-Critic Style

for iteration$=1, 2, \ldots$ **do**
  for actor$=1, 2, \ldots, N$ **do**
    Run policy π_{θ_old} in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_\text{old} \leftarrow \theta$
end for

Algorithm 1 of "Proximal Policy Optimization Algorithms" by Schulman et al.

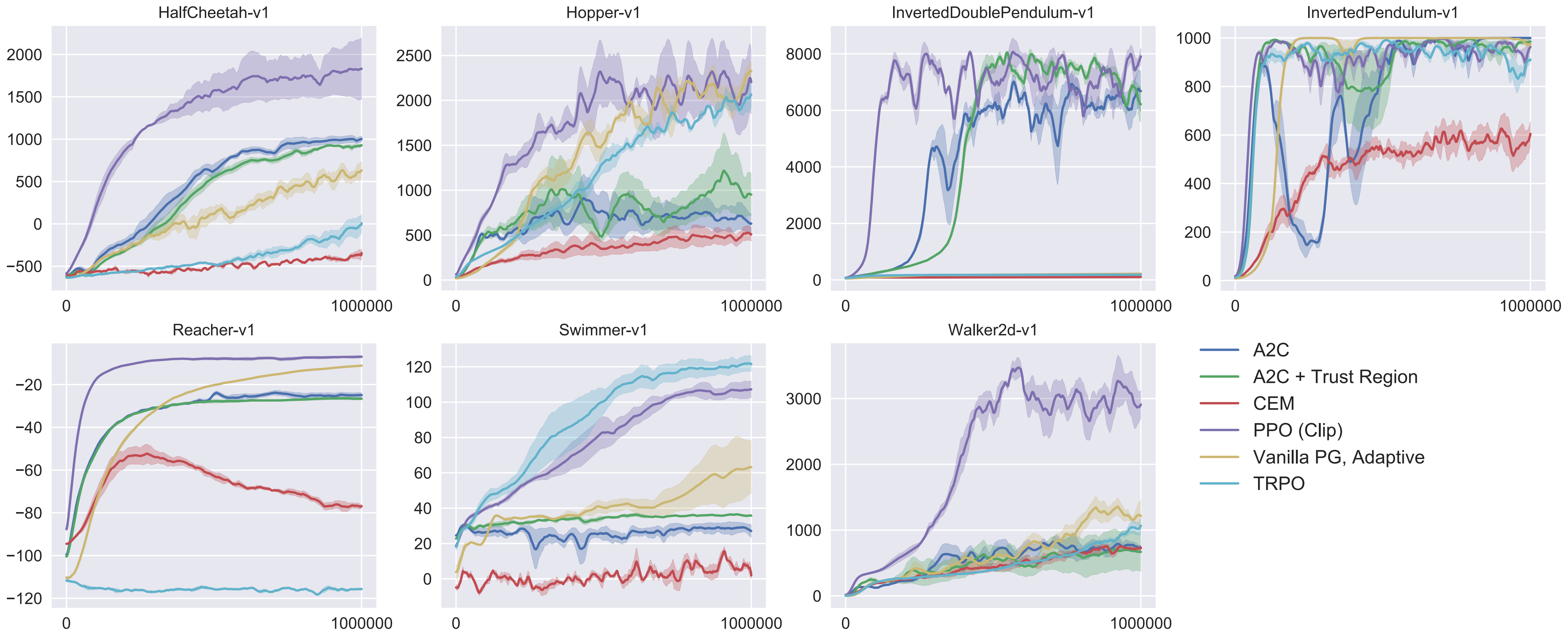


Figure 3: Comparison of several algorithms on several MuJoCo environments, training for one million timesteps.

Figure 3 of "Proximal Policy Optimization Algorithms" by Schulman et al.

Multi-Agent Hide-and-Seek

As another example, consider https://openai.com/blog/emergent-tool-use/.