Soft Actor Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

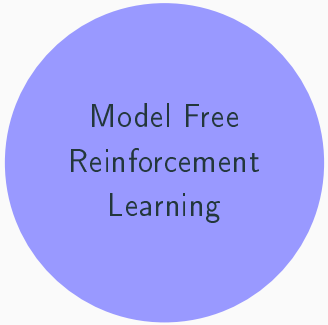
Lucien LE GALL - Alexandre ZENOU - Remi MOSHFEGHI

January 8, 2026

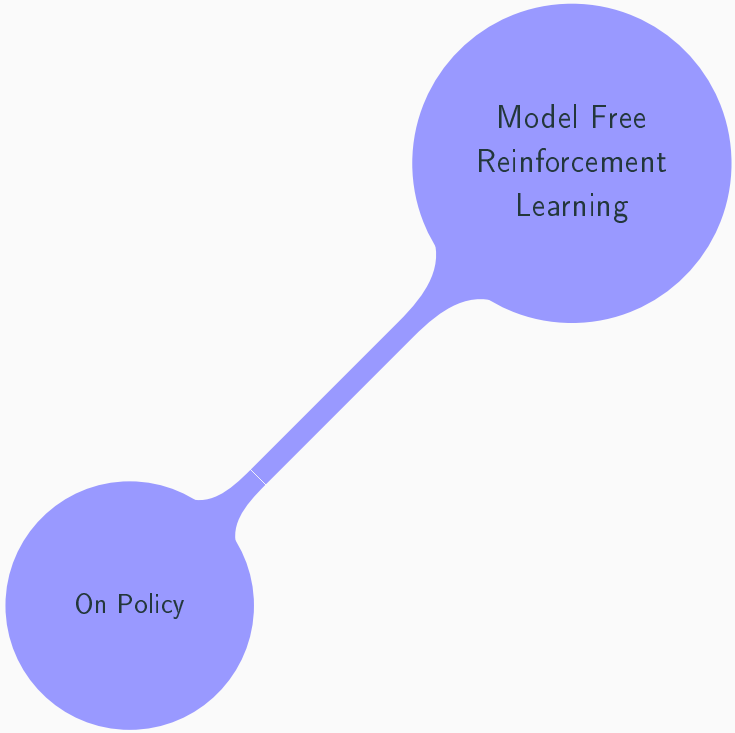
Based on the article: (Haarnoja et al. 2018)

Introduction and context

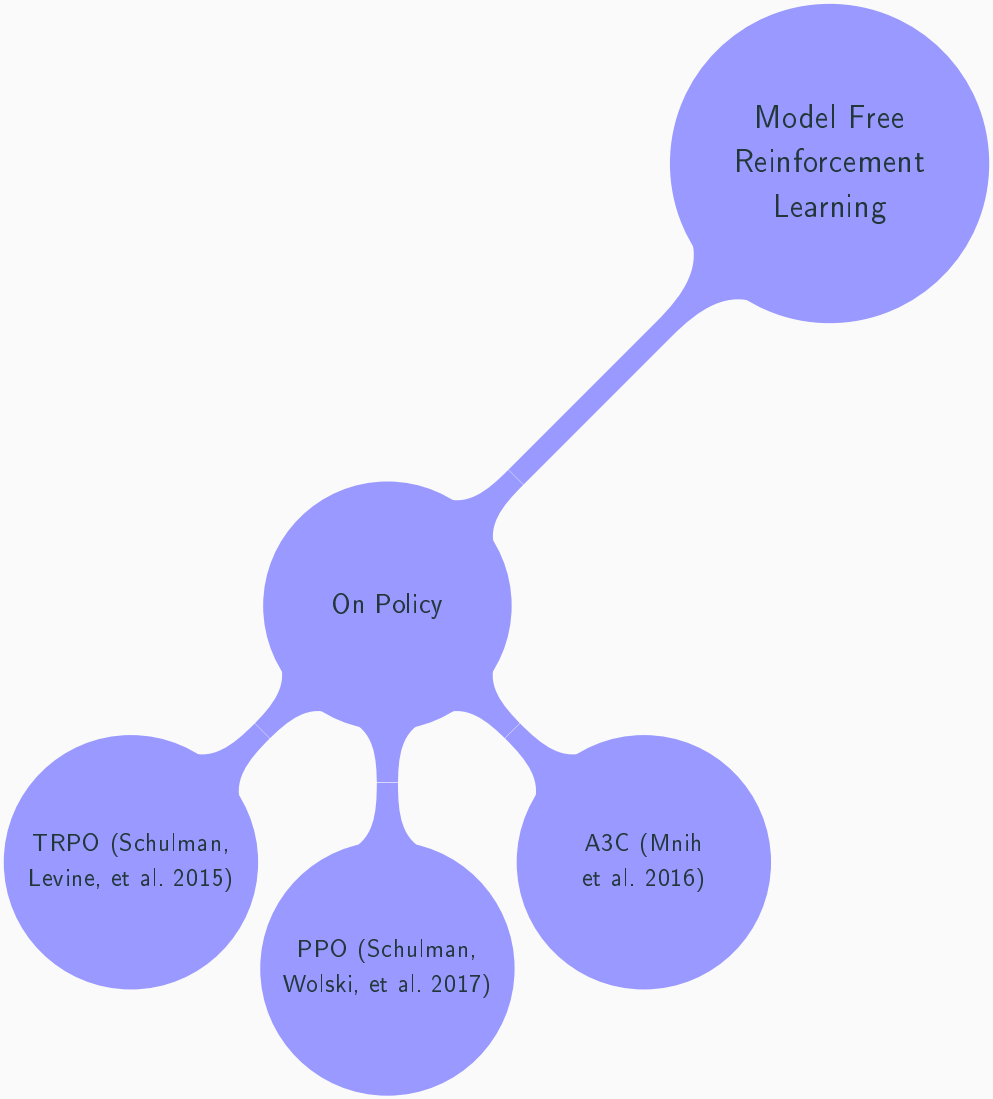
Accurate Recap of the context



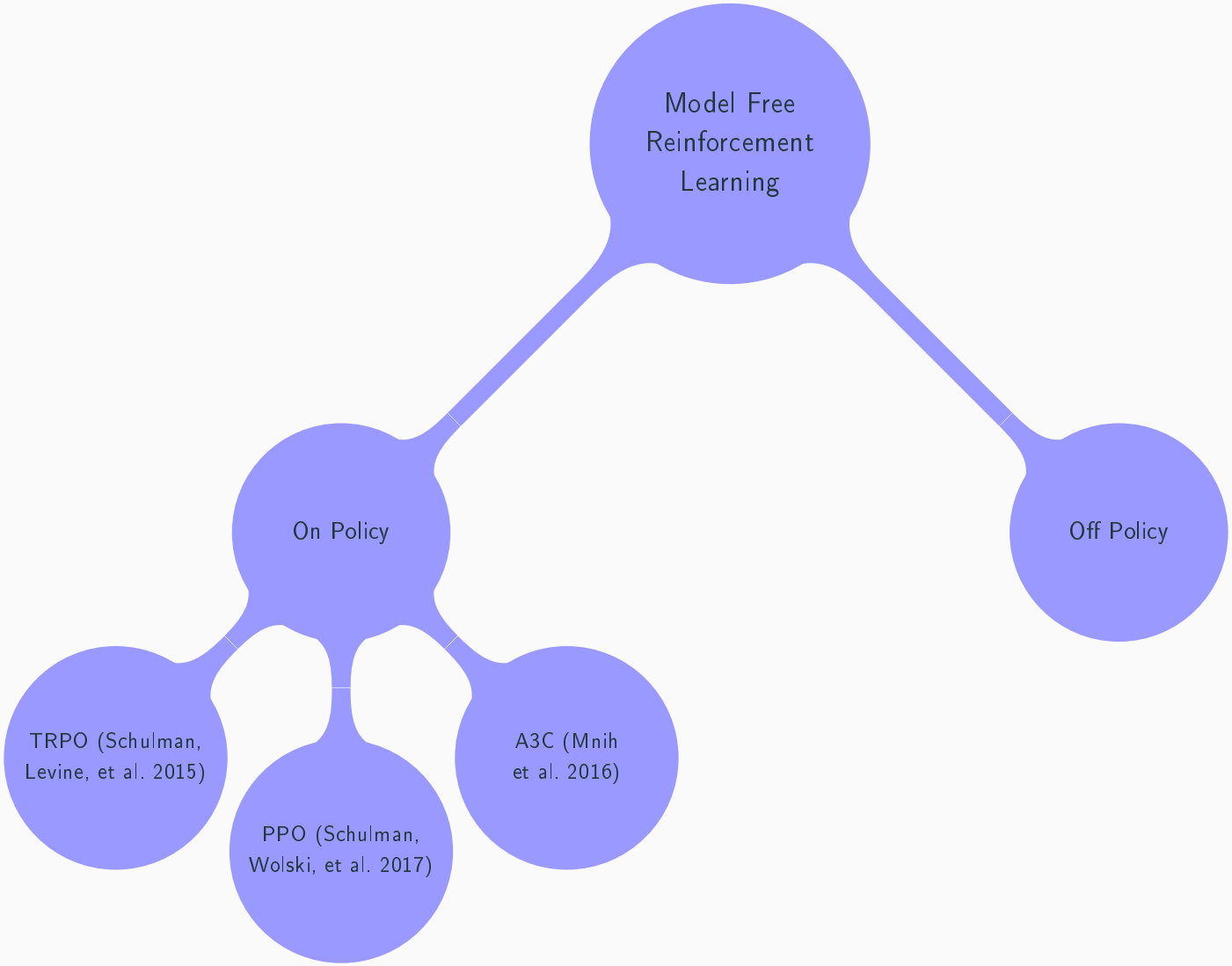
Accurate Recap of the context



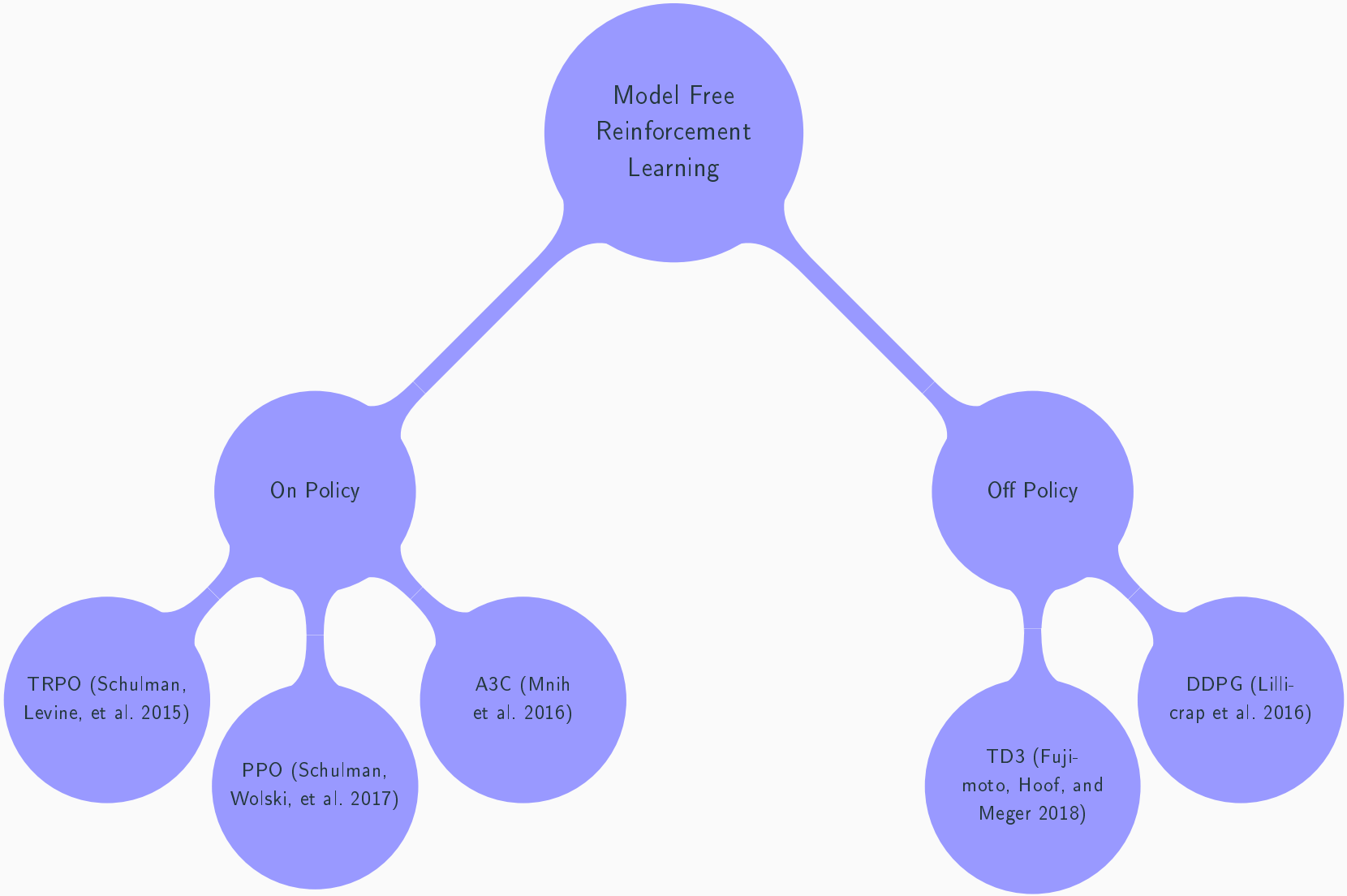
Accurate Recap of the context



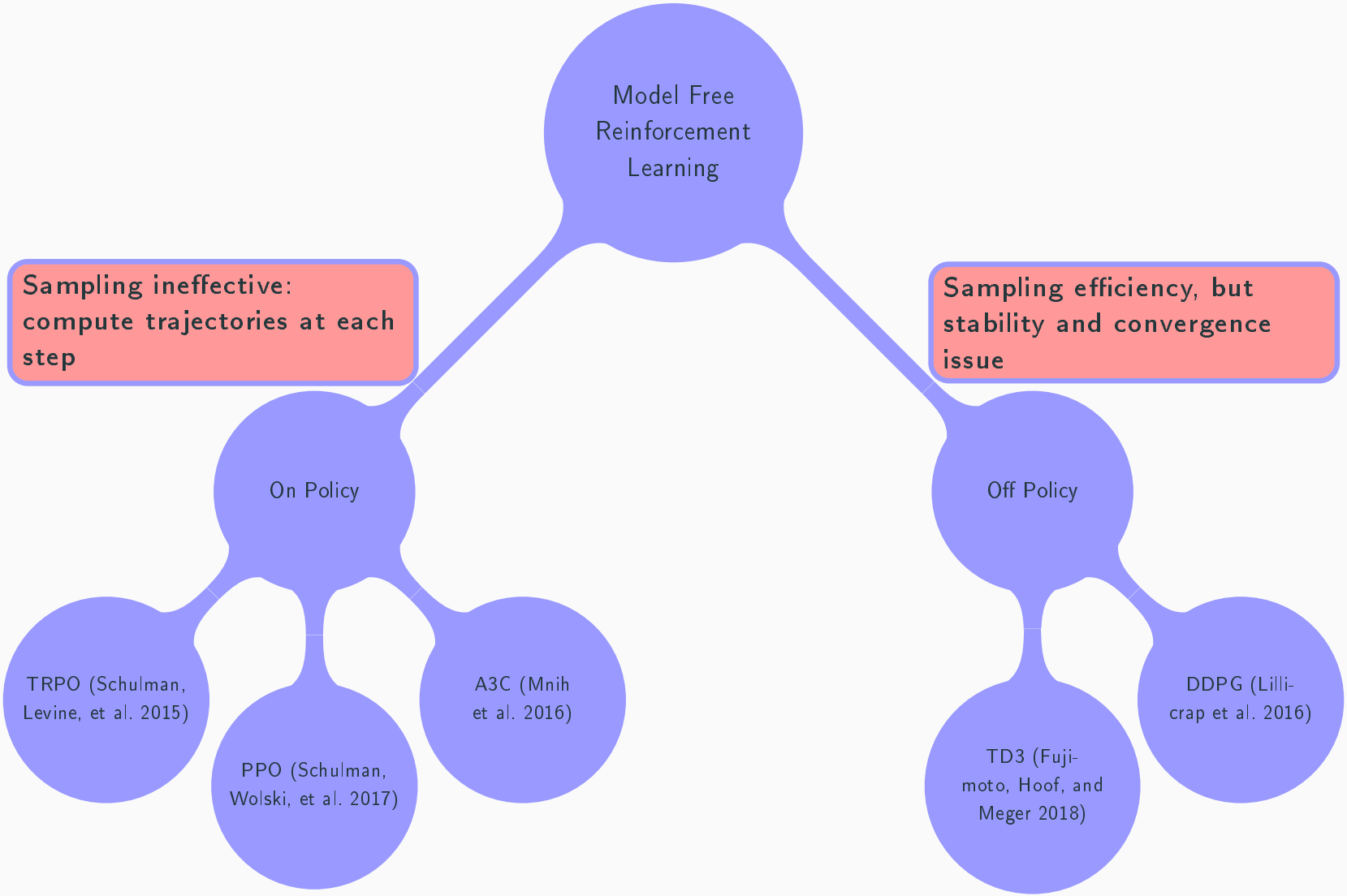
Accurate Recap of the context



Accurate Recap of the context



Accurate Recap of the context



Soft Policy Iteration

Augmented reward

The objective function is defined as:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{s_t \sim \mu, a_t \sim \pi}(r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))) \tag{1}$$

Where the entropy is the following:

$$\mathcal{H}(\pi(\cdot|s_t)) = -\mathbb{E}_{a_t \sim \pi}\big[\log \pi(a_t \mid s_t)\big] \tag{2}$$

Introducing a reward *augmented* reward $r_\pi(s, a) = r(s, a) + \mathbb{E}_{s' \sim \mu}(\mathcal{H}(\pi(\cdot|s')))$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$, one can rewrite the objective function as the well-known finite horizon objective function for MDPs: $J(\pi) = \mathbb{E}_{s \sim \mu}(v_\pi(s))$.

Soft Policy evaluation: classic policy evaluation but modified reward

One can derive the policy evaluation presented in the paper using the Bellman equation (Sutton and Barto 1998).

$$
\begin{aligned}
q(s,a) &= r_\pi(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v(s') \text{ (Bellman equation)} \\
&= r(s,a) - \gamma \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} p(s'|s,a) \pi(a'|s') \log(\pi(a'|s')) \\
&+ \gamma \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} p(s'|s,a) \pi(a'|s') q(s',a')
\end{aligned}
$$

One deduces:

$$
q(s,a) = r(s,a) + \gamma \mathbb{E}_{s \sim \mu}(\bar{v}(s)) \tag{3}
$$

with the *augmented* value function defined as:

$$
\bar{v}(s) = \mathbb{E}_{a \sim \pi}(q(s,a) - \log(\pi(a|s))) \tag{4}
$$

Soft Policy improvement

The policy improvement step is the following for each state $s \in \mathcal{S}$

$$\pi_{\text{new}}(a|s) = \arg\min_{\pi' \in \Pi} D_{\text{KL}} \left(\pi'(\cdot|s) \middle\| \frac{\exp(q^{\pi^{\text{old}}}(s, \cdot))}{Z^{\pi^{\text{old}}}(s)} \right) \tag{5}$$

Where the Kullback-Leibler divergence ensures a projection of the improve policy over a desired set of policies Π. One can easily show that:

Proposition (Soft Policy improvement)
If a policy $\pi^{new}(\cdot|s)$ fulfilled 5, and Π is the set of distribution over $\mathcal{A}$, then:

$$\pi_{new}(a|s) = \frac{\exp(q^{\pi^{old}}(s, a))}{\sum_{a \in \mathcal{A}} \exp(q^{\pi^{old}}(s, a))} \tag{6}$$

Proof.
See in appendix A (page 23).

□

Convergence of the algorithm

The policy evaluation algorithm converges due to two facts:

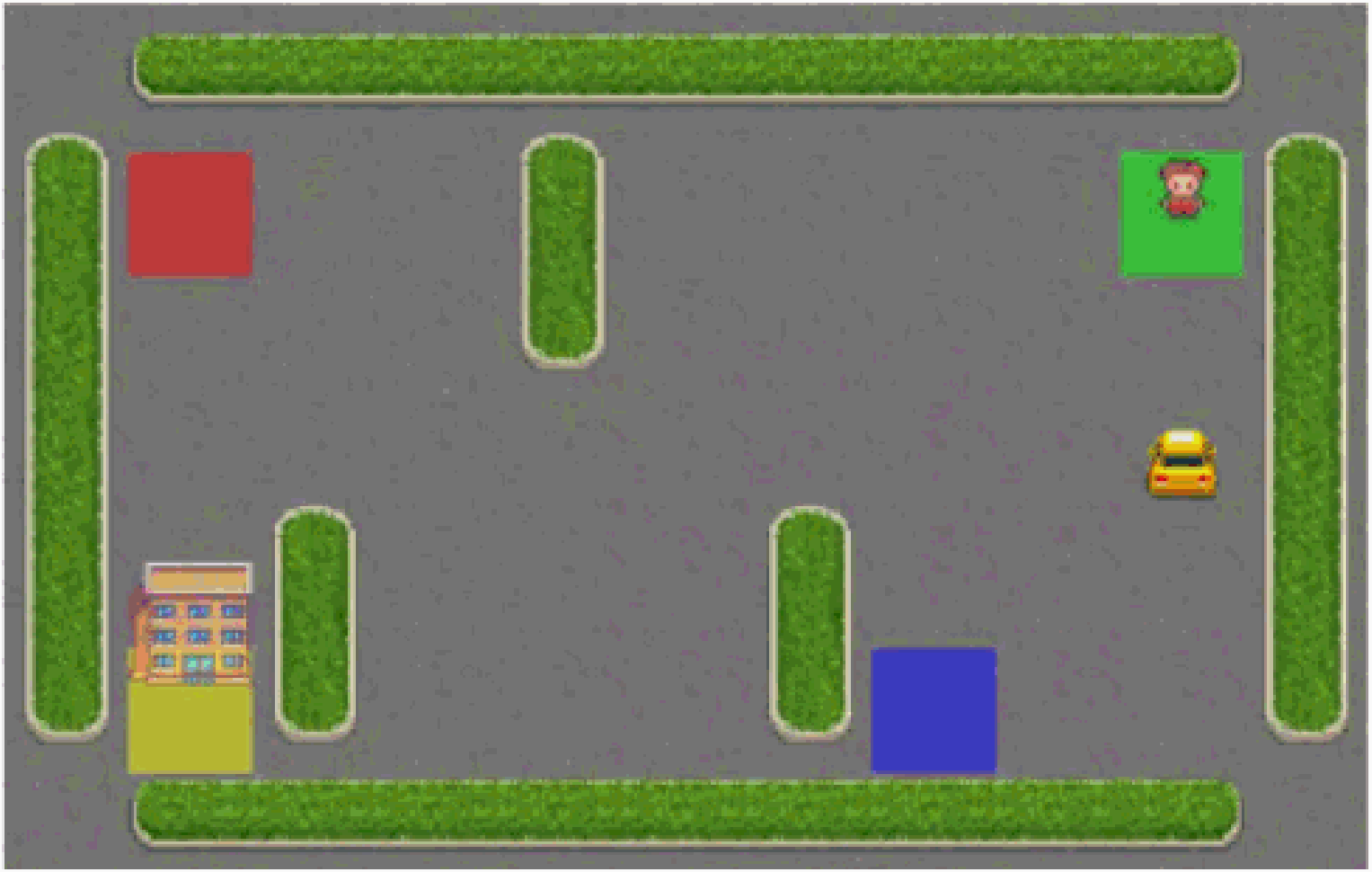
1. Contraction of the *soft Bellman operator* which ensures the convergences of the q-function.
2. The soft policy improvement which ensures that we improve the q-function at each step.

Algorithm 1: Soft Policy Iteration

Input: Initialise q and π

foreach i *in* $[\![1, N_{iter}]\!]$ **do**
- **Policy evaluation**
- **foreach** j *in* $[\![1, N_q]\!]$ **do**
  - **foreach** s *in* $\mathcal{S}$ **do**
    - **foreach** a *in* $\mathcal{A}$ **do**
      - $\text{acc} \leftarrow 0$
      - **foreach** s' *in* $\mathcal{S}$ **do**
        - **foreach** a' *in* $\mathcal{A}$ **do**
          - $\text{acc} \leftarrow p(s'|s,a)\pi(a'|s')q(s',a') + p(s'|s,a)\pi(a'|s')\log(\pi(a'|s'))$
      - $q(s,a) \leftarrow r(s,a) + \gamma \times \text{acc}$
- **Policy improvement** $\pi(a|s) \leftarrow \dfrac{\exp(q^{\pi}(s,a))}{\sum_{a \in \mathcal{A}} \exp(q^{\pi}(s,a))}$

Numerical results with `env = gym.make("Taxi-v3")`



Soft Policy iteration only works with finite states and actions space

=> need to developp more robust methods for continuous spaces

Soft Actor Critic

Soft Actor Critic

For continous spaces, one need to introduce specific function approxiators:

- Value function:

$$J_{\bar{v}}(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\frac{1}{2} \left(\bar{v}_\phi(s_t) - \underbrace{\mathbb{E}_{a_t \sim \pi_\phi}[q_\theta(s_t, a_t) - \log(\pi_\phi(a_t|s_t))]}_{\text{Approximation of the augmented value function given by 4}} \right)^2 \right] \tag{7}$$

- q function:

$$J_q(\theta) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(s_t, a_t) - \underbrace{(r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \mu}[\bar{v}_{\bar{\psi}}(s_{t+1})])}_{\text{Approximation of the real } q \text{ value given by 3}} \right)^2 \right] \tag{8}$$

- π:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[D_{\mathrm{KL}} \left(\pi_\phi(\cdot|s_t) \middle\| \frac{q_\theta(s_t, \cdot)}{Z_\theta(s_t)} \right) \right] \tag{9}$$

One can now perform a stochastic gradient descent.

Algorithm

Algorithm 2: Soft Actor-Critic (SAC)

Initialize:

Parameters $\psi, \theta_1, \theta_2, \phi$

Replay buffer $\mathcal{D} \leftarrow \emptyset$ `// used for Off-Policy learning`

for *each iteration* **do**

for *each environment step* **do**

$a_t \sim \pi_\phi(a_t|s_t)$ `// Sample action`

$s_{t+1} \sim p(s_{t+1}|s_t, a_t)$ `// Step environment`

$\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$

for *each gradient step* **do**

`// Neural Networks over a batch` B `selected over` $\mathcal{D}$ `and compute gradient` *via* `backpropagation`

$\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$

$\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ `// Update Policy`

$\theta_i \leftarrow \theta_i - \lambda_q \hat{\nabla}_{\theta_i} J_q(\theta_i)$ for $i \in \{1, 2\}$ `// Update q-networks`

$\bar{\psi} = \tau\psi + (1 - \tau)\bar{\psi}$

Pratical implementation of the code

The code involves several ideas:

- Use function approximation to deal with continuous functions.
- Two differents Neural Networks for the policy (actor) and the q function (Critic).
- Perform Stochastic gradient descent.
- Replay buffer to perform off policy learning.

as well as keeping this idea of entropy to help for stabilization.

Practical Implementation of the Code

- `next_values = next_q_values - entropy_scale * next_log_pis`
  and `q_targets = rewards + discounts * next_values`
  corresponds respectively to the estimation of the value and of the estimated $\hat{q}$: $v(s) = \mathbb{E}_{a\sim\pi}(q(s,a) - \log(\pi(a|s)))$ (4),
  $\hat{q}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p}[v_{\bar{\phi}}(s_{t+1})]$
- **Update of θ (Critic):**
  `q_losses = 0.5*(tf.losses.MSE(y_true=q_targets, y_pred=q_values))`
  corresponds to: $J_q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(q_\theta(s_t,a_t) - \hat{q}(s_t,a_t)\right)^2\right]$
- **Update of ϕ (Actor):**
  `policy_losses = entropy_scale * log_pis - q_log_targets`
  corresponds to:
  $J_\pi(q) = \mathbb{E}_{s_t\sim\mathcal{D},\epsilon_t\sim\mathcal{N}}[\log \pi_\phi(f_\phi(\epsilon_t; s_t)|s_t) - q_\theta(s_t, f_\phi(\epsilon_t; s_t))]$.

In fact, a gradient descent on ϕ is unuseful.

Difficulties Cloning the GitHub Repository

Creating the SAC environment using `conda env create -f environment.yml` was impossible for the SAC repository (even for the most recent version), due to outdated dependencies. That results in the following errors:

```
Could not solve for environment specs.
The following packages are incompatible:
- joblib ==0.10.3
  - joblib 0.10.3 requires
    - python =3.5 *, which can be installed;
  - joblib 0.10.3 requires
    - python =3.4 *, which does not exist
      (possibly due to a missing channel);
  - joblib 0.10.3 requires
    - python =2.7 *, but no viable options are available:
      - python [2.7.13|2.7.14|...|2.7.18] requires
        - vc =9 *, which can be installed;
      - python 2.7.12 conflicts with all previously
        installable versions...
```

Results and conclusion

Results presented in the article: Comparaison between different algorithms

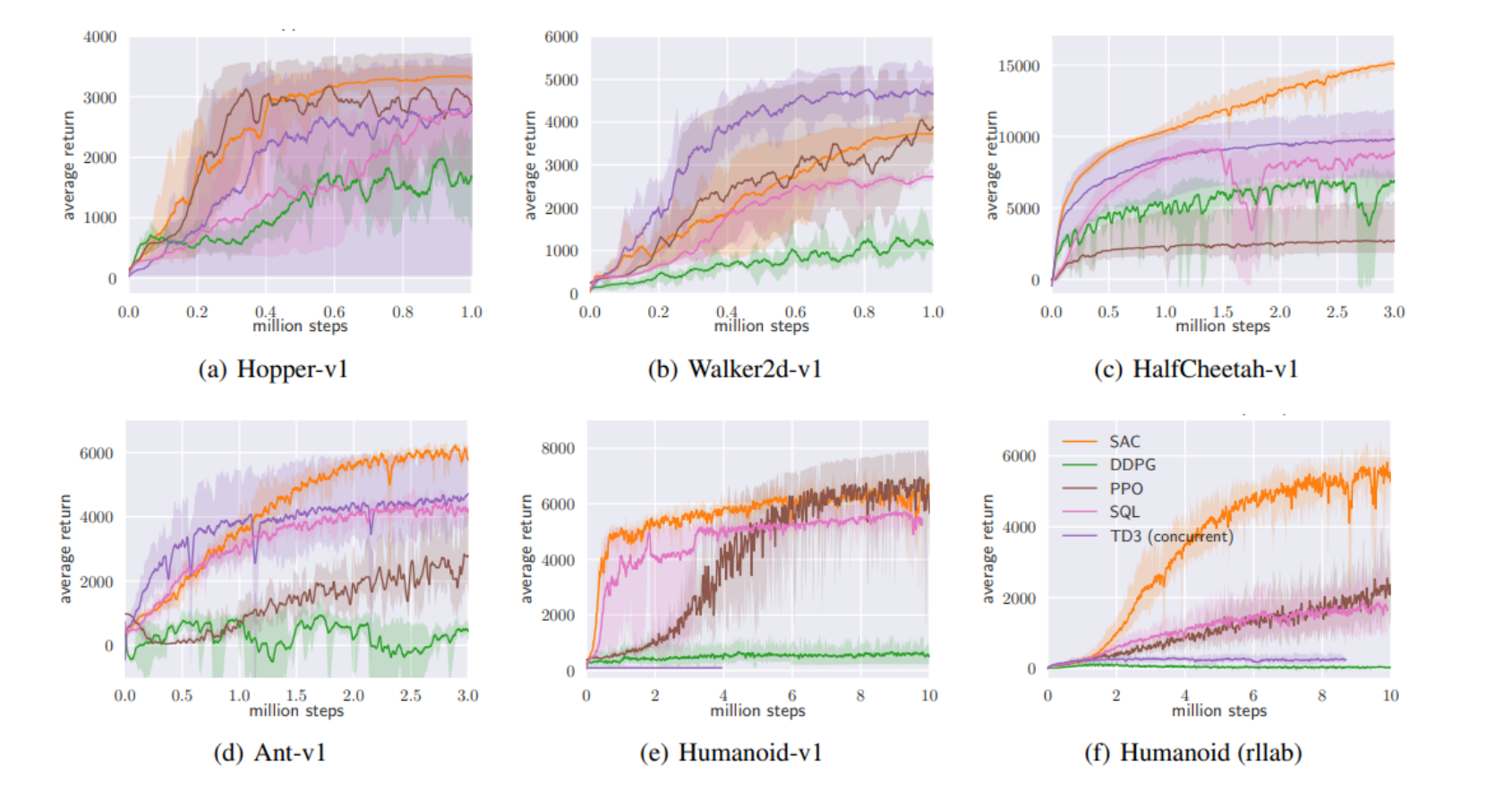


Figure 1: Results: Comparaison between different algorithms on the Mujoco environment

Results presented in the article: Stochastic VS deterministic

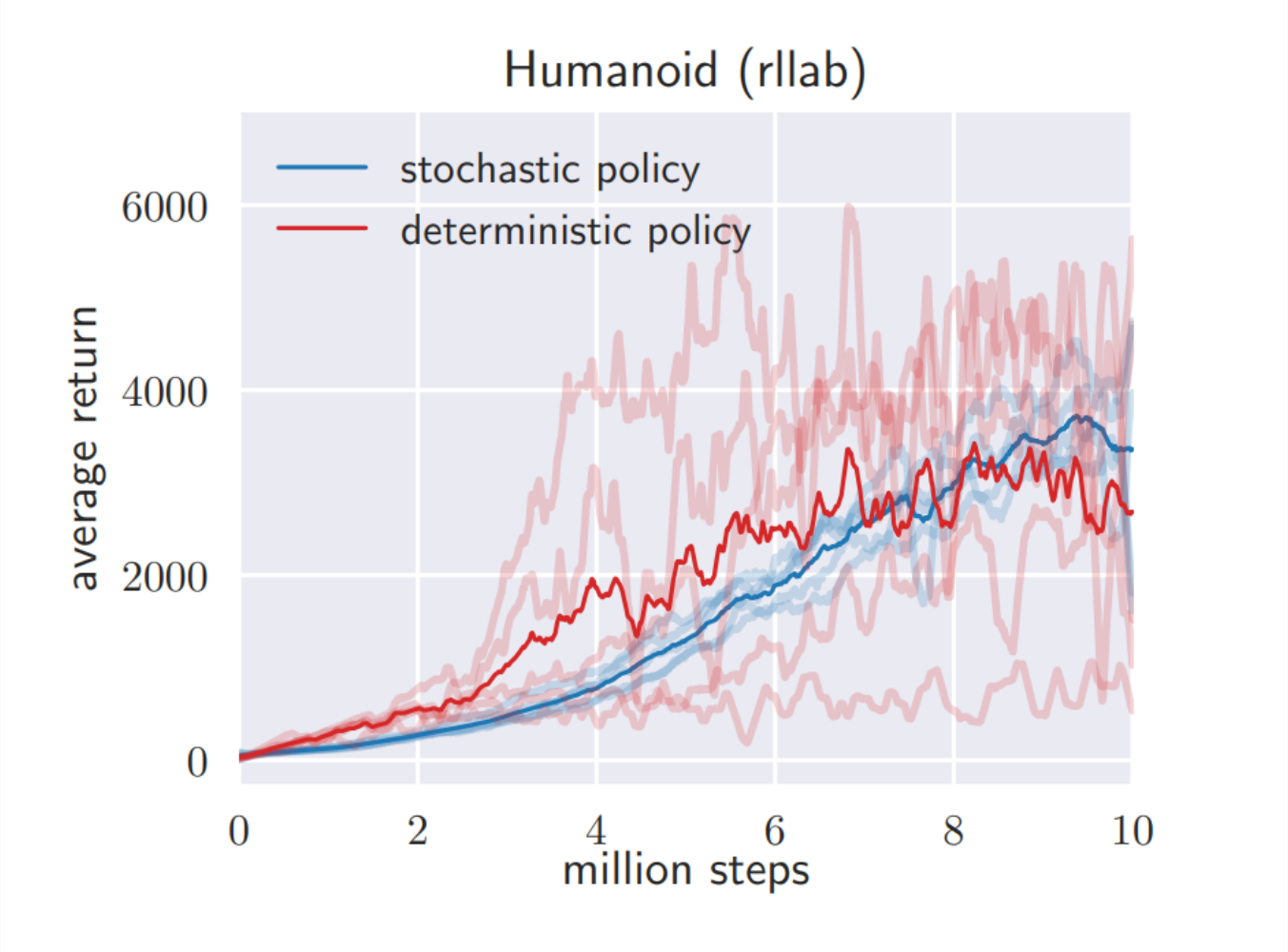


Figure 2: Results: Comparaison between different stochastic policy and deterministic policy for the SAC algorithm

Results presented in the article: Parameters influence

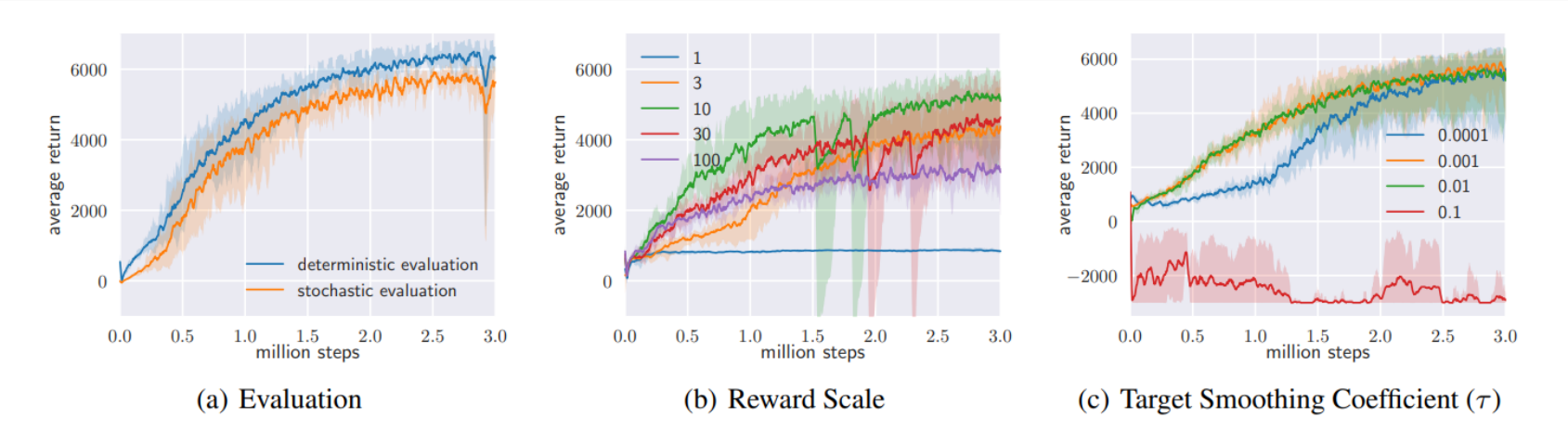


Figure 3: Results: Parameters influence of SAC

Our attempt to reproduce the results

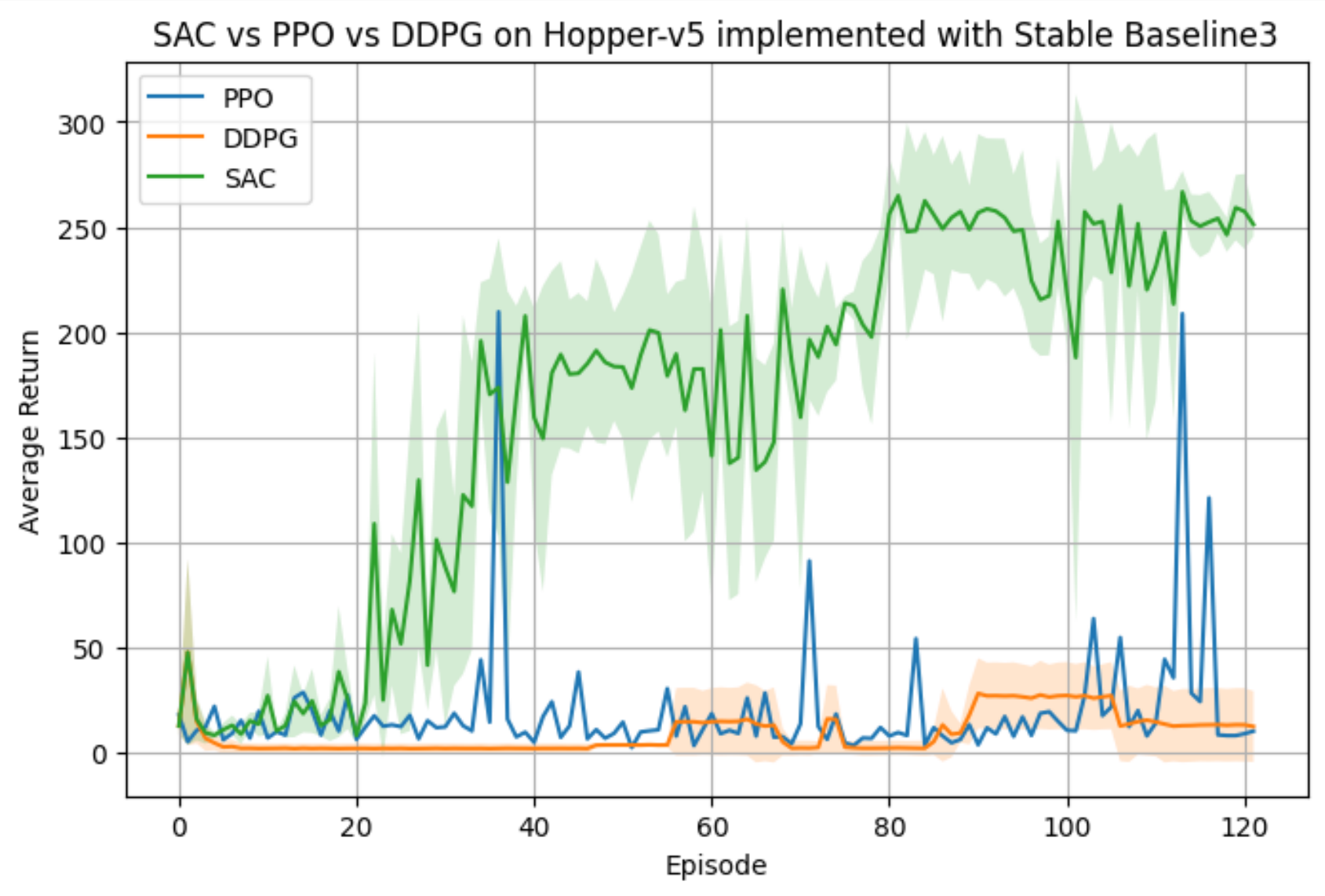


Figure 4: Personnal result implemented with the Stable Baseline3 library

Appendix A: Proof for proposition 1

We want to compute 5 $\arg\min_{\pi' \in \Pi} D_{\mathrm{KL}}\left(\pi'(\cdot|s) \middle\| \frac{\exp(q^{\pi^{\mathrm{old}}}(s,\cdot))}{Z^{\pi^{\mathrm{old}}}(s)}\right)$ One has:

$$
\begin{aligned}
D_{\mathrm{KL}}\left(\pi'(\cdot|s) \middle\| \frac{\exp(q^{\pi^{\mathrm{old}}}(s,\cdot))}{Z^{\pi^{\mathrm{old}}}(s)}\right) &= \sum_{a\in\mathcal{A}} \pi'(a|s) \log\left(\frac{\pi'(a|s) Z^{\pi^{\mathrm{old}}}(s)}{\exp(q^{\pi^{\mathrm{old}}}(s,a))}\right) \\
&= \sum_{a\in\mathcal{A}} \pi'(a|s) \log \pi'(a|s) \\
&\quad - \sum_{a\in\mathcal{A}} \pi'(a|s) q^{\pi^{\mathrm{old}}}(s,a) + \log Z^{\pi^{\mathrm{old}}}(s) \\
&:= J(\pi'(\cdot|s))
\end{aligned}
$$

Appendix A: Proof for proposition 1

J is a strictly convex and continuous function over the compact set which is the simplex, so it has a unique minimum. We consider the Lagrangian function $\mathcal{L}(\pi', \lambda) = J(\pi'(\cdot|s)) + \lambda \left(\sum_{a \in \mathcal{A}} \pi'(a|s) - 1 \right)$

The minimum is reached when the gradient w.r.t. π' vanishes:
$\frac{\partial \mathcal{L}}{\partial \pi'(a|s)} = \log \pi'(a|s) + 1 - q^{\pi^{\text{old}}}(s,a) + \lambda = 0, \quad \forall a \in \mathcal{A}.$

Taking into account the condition $\sum_{a \in \mathcal{A}} \pi'(a|s) = 1$ helps to find λ

which leads to: $\pi^*(a|s) = \frac{\exp(q^{\pi^{\text{old}}}(s,a))}{\sum_{b \in \mathcal{A}} \exp(q^{\pi^{\text{old}}}(s,b))}$.

Appendix B: TRPO and PPO

Using like a Taylor-expansion, one get that:

$$V(\pi') = V(\pi) + (1-\gamma)\,\mathbb{E}_{S\sim d_\pi,\, A\sim\pi}\left[\frac{\pi'(A|S)}{\pi(A|S)}\, a_\pi(S,A)\right] + O\left(\mathbb{E}_{S\sim d_\pi}\left[D_{\mathrm{KL}}\left(\pi'(\cdot|S)\,\|\,\pi(\cdot|S)\right)\right]\right)$$

One aims to achieve directly a policy update:

$$\theta_{k+1} \in \arg\max_\theta L_{\pi_{\theta_k}}(\pi_\theta) \;\triangleq\; \mathbb{E}_{\pi_{\theta_k}}\left[\sum_{t=0}^{\infty}\gamma^t \frac{\pi_\theta(A_t|S_t)}{\pi_{\theta_k}(A_t|S_t)} a_{\pi_{\theta_k}}(S_t,A_t)\right]$$
$$\text{s.t.}\quad \mathbb{E}_{S\sim d_{\pi_{\theta_k}}}\left[D_{\mathrm{KL}}(\pi_\theta(\cdot|S)\,\|\,\pi_{\theta_k}(\cdot|S))\right] \le \delta \tag{10}$$

This is the main idea in PPO (Schulman, Levine, et al. 2015), while in TPO achieves gradient clipping over (Schulman, Wolski, et al. 2017).

Appendix C : DDPG and TD3

Let's recap the optimal Bellman equation for the q function.

$$q^*(s,a) = \mathbb{E}_{s' \sim \mu}\left[r(s,a) + \gamma \max_{a'}(q^*(s',a'))\right] \tag{11}$$

From this idea, one can derive a squared Bellman error (MSBE):

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} q_\phi(s',a')\right)\right)^2\right] \tag{12}$$

The $(d-1)$ is set for computation reasons. $\mathcal{D}$ is a replay-buffer which helps to perform Off-policy learning.
TD3 is based on this idea but introduces Double-q Learning to reduce positive bias.

References

[FHM18] Scott Fujimoto, Herke van Hoof, and David Meger. **"Addressing Function Approximation Error in Actor-Critic Methods".** In: (2018). URL: https://arxiv.org/abs/1802.09477.

[Haa+18] Tuomas Haarnoja et al. **"Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor".** In: (2018). arXiv: 1801.01290 [cs.LG]. URL: https://arxiv.org/abs/1801.01290.

References ii

[Lil+16] Timothy P. Lillicrap et al. **"Continuous control with deep reinforcement learning".** In: *International Conference on Learning Representations (ICLR)*. 2016.

[Mni+16] Volodymyr Mnih et al. **"Asynchronous Methods for Deep Reinforcement Learning".** In: Proceedings of Machine Learning Research 48 (2016), pp. 1928–1937. URL: http://proceedings.mlr.press/v48/mniha16.html.

[Sch+15] John Schulman, Sergey Levine, et al. **"Trust Region Policy Optimization.".** In: 37 (2015), pp. 1889–1897.

References iii

[Sch+17] John Schulman, Filip Wolski, et al. **"Proximal Policy Optimization Algorithms".** In: (2017). arXiv:1707.06347 [cs]. DOI: 10.48550/arXiv.1707.06347. URL: http://arxiv.org/abs/1707.06347.

[SB98] R. S. Sutton and A. G. Barto. ***Reinforcement Learning: An Introduction.*** Sections 4.1 and 4.2 (Policy Evaluation/Improvement). MIT Press, 1998.