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

Abdelkarim Eljandoubi
abdelkarim.eljandoubi@ensta-paris.fr

Géraud Faye
geraud.faye@student-cs.fr

Orso Forghieri
orso.forghieri@gmail.com

April 1, 2022

Abstract

Soft Actor-Critic method is a model-free deep reinforcement learning algorithm that aims to explore wisely large environments. It is a combination of multiple methods of reinforcement learning like entropy maximization, an actor-critic structure and an off-policy framework that allow to reuse collected experience. In this context, we implemented the Soft Actor-Critic method in Python and applied it to the PendulumV0 and ReacherV1 environments. The algorithm is supposed to learn an near-optimal policy in this complex environment.

1 Introduction

Reinforcement learning methods made significant progress when model-free methods in Deep RL were developed. However, model-free method faces high complexity and convergence issues. In this context, [1] proposes an off-policy actor critic algorithm base on the maximum entropy principle.

1.1 Context

Model-free Deep Reinforcement Learning methods expensiveness mainly comes from the space state exploration and the hyperparameters tuning. Soft Actor-critic process aims to bring stability and efficiency to the policy research process. It focuses on the maximum entropy framework and augments the standard maximum reward with an entropy maximization term.

Thus, the article presents a method that combines three ingredients : an actor-critic architecture which separates the network for value function estimation and the one for policy distribution, an off-policy formulation with data collection and entropy maximization for stability.

1.2 Work

We implemented the Soft Actor-Critic algorithm described in the article. We then focused on two environments :

- Pendulum : A rotative pendulum to which we can apply a torque belonging in a continuous range (see Figure 1). The environment returns high reward when the pendulum is upward.
- Reacher : An arm composed of two rotative articulations has to reach a specific point in the environment. The actions are now composed of two torques instead of one, but the dynamics of the system is different as no gravity affects the arm (see Figure 2).

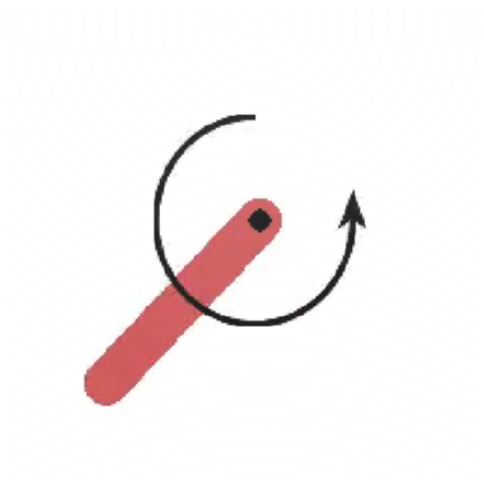

Figure 1: Pendulum environment : a bar subject to gravity

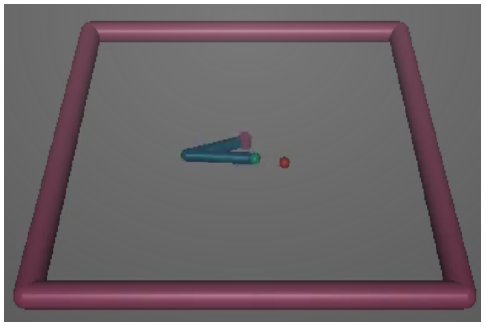

Figure 2: Reacher environment : control two hinges to reach a point

For both problems, we then trained an agent using Soft Actor-Critic to address it.

2 Method

2.1 Entropy maximization

First of all, we consider a maximum entropy framework, i.e. we aim to maximize a combination of the reward and the entropy :

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \alpha \mathcal{H}\left(\pi\left(\cdot \mid \mathbf{s}_t\right)\right) \right] \tag{1}$$

where α is a temperature parameter which determines the relative important of the entropy with respect to the reward. This objective allow to explore widely and can capture multiple modes of positive behaviors.

2.2 Soft-policy Iteration

Using equation 1, the soft Q-value can be extracted using a modified Bellman Operator

$$\mathcal{T}^\pi Q\left(\mathbf{s}_t, \mathbf{a}_l\right) \triangleq r\left(\mathbf{s}_t, \mathbf{a}_\ell\right) + \gamma \mathbb{E}_{\mathbf{s}_{\ell+1} \sim p}\left[V\left(\mathbf{s}_{t+1}\right)\right] \tag{2}$$

with the soft state value function

$$V\left(\mathbf{s}_t\right) = \mathbb{E}_{\mathbf{a}_t \sim \pi}\left[Q\left(\mathbf{s}_t, \mathbf{a}_t\right) - \log \pi\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right] \tag{3}$$

The convergence of Q to the soft Q-value is then proven in the article.

The policy improvement step consists in using :

$$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} \mathrm{D}_{\mathrm{KL}}\left(\pi'\left(\cdot \mid \mathbf{s}_\ell\right) \| \frac{\exp\left(Q^{\pi_{\text{old}}}\left(\mathrm{s}_t, \cdot\right)\right)}{Z^{\pi_{\text{old}}}\left(\mathbf{s}_t\right)}\right) \tag{4}$$

Here, the partition function $Z^{\pi_{\text{old}}}$ normalizes the distribution and does not contribute to the gradient. The article then proves that the projected policy improves the objective. Finally, the soft policy iteration is composed of the policy evaluation and the improvement. From now, we obtain the proof that using these steps make the policy converge to the optimal policy in the sense of Q. In order to make this algorithm not too expensive, it is necessary to make an approximation of both the Q-function and the policy.

2.3 Soft Actor-Critic

Thus, we consider parameterized state value function $V_\psi\left(\mathbf{s}_t\right)$, soft Q-function $Q_\theta\left(\mathbf{s}_t, \mathbf{a}_t\right)$ and policy $\pi_\phi\left(\mathbf{a}_t \mid \mathbf{s}_t\right)$. These function can models the value function and the policy. In this case, it is convenient to train simultaneously the approximation networks.

The soft value function is trained using

$$J_V(\psi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}}\left[\frac{1}{2}\left(V_\psi\left(\mathbf{s}_t\right) - \mathbb{E}_{\mathbf{a}_t \sim \pi_\phi}\left[Q_\theta\left(\mathbf{s}_t, \mathbf{a}_t\right) - \log \pi_\phi\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right]\right)^2\right]$$

where $\mathcal{D}$ is the distribution of the previouslu sampled states. The gradient can be approximated by the following equation :

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi\left(\mathbf{s}_t\right)\left(V_\psi\left(\mathbf{s}_t\right) - Q_\theta\left(\mathbf{s}_t, \mathbf{a}_t\right) + \log \pi_\phi\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right)$$

The soft Q-function can be determined using the Bellman residual :

$$J_Q(\theta) = \mathbb{E}_{\left(\mathbf{s}_t, \mathbf{a}_t\right) \sim \mathcal{D}}\left[\frac{1}{2}\left(Q_\theta\left(\mathbf{s}_t, \mathbf{a}_t\right) - \hat{Q}\left(\mathbf{s}_t, \mathbf{a}_t\right)\right)^2\right]$$

with

$$\hat{Q}\left(\mathbf{s}_t, \mathbf{a}_t\right) = r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim p}\left[V_{\bar{\psi}}\left(\mathbf{s}_{t+1}\right)\right]$$

which can optimized with

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta\left(\mathbf{a}_t, \mathbf{s}_t\right)\left(Q_\theta\left(\mathbf{s}_t, \mathbf{a}_t\right) - r\left(\mathbf{s}_t, \mathbf{a}_t\right) - \gamma V_{\bar{\psi}}\left(\mathbf{s}_{t+1}\right)\right)$$

Also, this update uses $\bar{\psi}$ that is a moving average of the network weights to stabilize training.

Finally, the policy can be learned minimizing the KL-divergence :

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\mathrm{D}_{\mathrm{KL}} \left(\pi_\phi \left(\cdot \mid \mathbf{s}_t \right) \| \frac{\exp\left(Q_\theta \left(\mathbf{s}_t, \cdot \right) \right)}{Z_\theta \left(\mathbf{s}_t \right)} \right) \right]$$

To approximate the best policy, we reparameterize the policy using

$$\mathbf{a}_t = f_\phi \left(\epsilon_t ; \mathbf{s}_t \right)$$

where ϵ_t is an input noise. We rewrite the policy loss as :

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}, \epsilon_t \sim \mathcal{N}} \left[\log \pi_\phi \left(f_\phi \left(\epsilon_t ; \mathbf{s}_t \right) \mid \mathbf{s}_t \right) - Q_\theta \left(\mathbf{s}_t, f_\phi \left(\epsilon_t ; \mathbf{s}_t \right) \right) \right]$$

with π_ϕ define in function of f_ϕ. Finally, we approximate the gradient :

$$\begin{aligned} \hat{\nabla}_\phi J_\pi(\phi) = & \nabla_\phi \log \pi_\phi \left(\mathbf{a}_t \mid \mathbf{s}_t \right) \\ & + \left(\nabla_{\mathbf{a}_t} \log \pi_\phi \left(\mathbf{a}_t \mid \mathbf{s}_t \right) - \nabla_{\mathbf{a}_t} Q \left(\mathbf{s}_t, \mathbf{a}_t \right) \right) \nabla_\phi f_\phi \left(\epsilon_t ; \mathbf{s}_t \right) \end{aligned}$$

We finally obtain the algorithm 1 :

Initialize parameter vectors $\psi, \bar{\psi}, \theta, \phi$ **for** *each iteration* **do**
  for *each environment step* **do**
    $\mathbf{a}_t \sim \pi_\phi \left(\mathbf{a}_t \mid \mathbf{s}_t \right)$
    $\mathbf{s}_{t+1} \sim p \left(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t \right)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \left\{ \left(\mathbf{s}_t, \mathbf{a}_t, r \left(\mathbf{s}_t, \mathbf{a}_t \right), \mathbf{s}_{t+1} \right) \right\}$
  end
  for *each gradient step* **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q \left(\theta_i \right)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau \psi + (1 - \tau) \bar{\psi}$
  end
end

Algorithm 1: Soft Actor-Critic

3 Implementation

We implemented the SAC algorithm using the frameworks and modules seen in class :

- Jax : useful for array treatment, jitting functions and computing the gradients automatically
- Haiku : to make the neural networks more easily using jax, as they remember the weights implicitely, keeping jax functions pure at all time
- dm_env : to create wrappers around the gym and mujoco environments

The SAC agent was then written from scratch, as well as the loss functions used for learning.

4 Results

We implemented and performed training on trajectories of length 50 to 200 iterations instead of the 100,000 that the paper proposes.

The loss curves obtained show significant variations that can be observed in figures 3 and 4.

The only loss to actually reduce is the policy loss, but it reaches very low values. The losses in the pendulum environment are erratic, with values oscillating between very high and very low.

We also plot the average returns over the 50 iterations, which aren't increasing, despite the change in the networks.

The first return curve is erratic and the second one is clearly decreasing over and over. We think it may be because of a bad implementation of the losses.

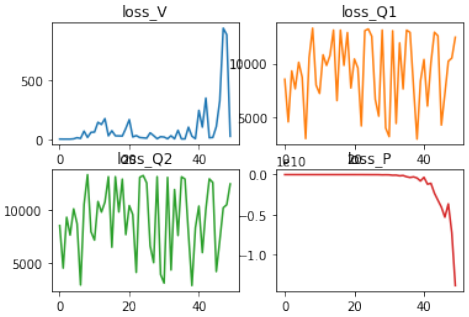

Figure 3: Losses in the pendulum environment (50 learning steps)

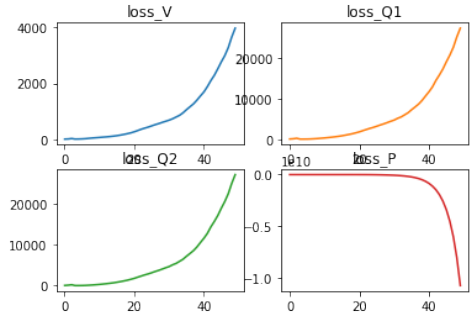

Figure 4: Losses in the reacher environment (50 learning steps)

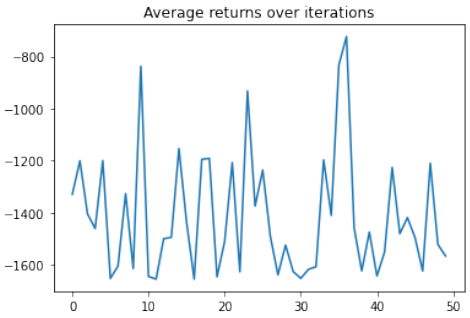

Figure 5: Average returns in the pendulum environment

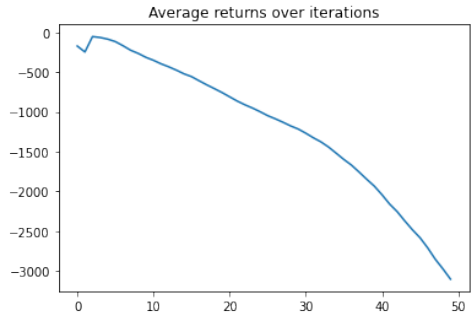

Figure 6: Average returns in the reacher environment

5 Conclusion

In this project, we are interested in the Soft Actor-Critic method . We have implemented the proposed algorithm and neural network learning. The algorithm seems to learn very slowly and has to explore the environment for a long time. We obtain the last curve by tuning appropriate parameters (learning rate, length of the learning phase). Thus, we faced different issues of convergence and exploration of the environment. We have also faced problems of convergence of the losses implemented. This project allowed us to address an algorithm that is fundamental for reinforcement learning in the model-free context.

References

[1] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.