Soft Actor Critic and its variants

Mattia Bertè
Department of Computer Science
University of Bologna
mattia.berte@studio.unibo.it

Abstract

This project contains a comparison between Soft Actor Critic method and two of its variant, the first which include the automatic temperature parameter tuning while the second is the Averaged-SAC. The experiments for testing the performances are conducted using two MuJoCo gym environments. Link to GitHub repository with code and demo [1].

1 Introduction

OpenAI (Brockman et al. 2016) gym presents multiple environments for testing reinforcement learning algorithms, among them we can find MuJoCo (Todorov et al. 2012), a physics simulator engine for facilitating development in robotics and biomechanics.
Standard DQN (Mnih et al. 2013) algorithms become useless when we have to deal with agents which operate in continuous action space. So we need to resort to Actor Critic algorithm in which the actor learns the policy (i.e how to chose the best action) while critic learns the state-value function.

1.1 HalfCheetah-v4

The HalfCheetah (Wawrzynski, 2009) is a 2-dimensional robot composed of 9 links and 8 joints. The goal is to make the cheetah move as fast as possible.
The torso and head of the cheetah are fixed, and the torque can only be applied on the other 6 joints, which is the dimension of the action space. Each action can assume values in $[-1, 1]$.
The observation space is a 17 dimensions array containing position and velocity of the body parts. There's no limitation on the value they can assume.
The starting state is an array of 0 plus a random noise. The noise comes from an uniform distribution $U(-0.1, 0.1)$ for the position values while from a normal distribution $\mathcal{N}(0, 0.1)$ for the velocity values. Episodes end after 1000 steps.
The reward is made of two terms: a reward for moving forward minus a cost that prevent the cheetah to take too big actions.

1.2 Humanoid-v4

The Humanoid (Tassa et al., 2012) is a 3D bipedal robot whose goal is to walk as fast as possible without falling over. It presents 17 joints and so it's also the dimensions of the action space whose values are in $[-0.4, 0.4]$.
The observation space is much bigger than the previous one, 376 dimensions, indeed not only it is a 3D environment but also more information are provided such as the ones related to center of mass and inertia.
The starting state is a fixed array of value (so that the humanoid starts standing up and facing forward) plus some random noise from $U(-0.01, 0.01)$ to the value related to joints positions and velocity.
The episode ends or after 1000 steps or if the z coordinate of the torso is outside $[1, 2]$ that means the

[1] https://github.com/TiaBerte/rl-soft-actor-critic

robot fell over.
The reward is composed of four terms: two positive rewards, one for moving forward and one for being alive, minus two negative reward, one which prevents too big action and one that takes in account external forces.

2 Soft Actor Critic

Soft actor-critic (Haarnoja et al., 2018) is an off-policy actor-critic algorithm based on the maximum entropy framework. In this framework the optimal policy is the one that optimizes at the same time both the expected value of cumulative reward and the entropy of the policy. A parameter α is introduced to regulate the importance of the entropy with respect to the reward.

$$\pi^* = \arg\max_{\pi} \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[\gamma^t(r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{1}$$

Off-policy means that the learning process isn't depend on the current action/state of the systems but instead it exploits previous experience which is stored in a replay buffer and from which training batches are randomly sampled.
The systems is composed by two soft Q-functions, two target soft Q-functions and a policy. The policy is defined by a Gaussian whose mean and variance are computed by a neural network with parameter ϕ. The soft Q-functions are defined by two neural networks with parameters θ_1 and θ_2, while the target networks are parametrized with an exponential moving average of the parameters of the soft Q-functions (eq. 2).

$$\bar{\theta}_i \leftarrow (1-\tau)\bar{\theta}_i + \tau\theta_i \tag{2}$$

The presence of two soft Q-function helps to mitigate the effect of positive bias in the policy improvement step and moreover it helps to speed up the training. However only the minimum between the two soft Q-value is used for computing the gradients. The soft Q-function networks are trained to minimize eq. 3.

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(s_t,a_t) - \left(r(s_t,a_t) + \gamma\,\mathbb{E}_{s_{t+1}\sim p}[V_{\bar{\theta}}(s_{t+1})]\right)\right)^2\right] \tag{3}$$

In the first version of this algorithm was present a value-function approximator network which was then abandoned, since the value function is parametrized trough soft Q-function parameters using eq. 4

$$V(s_t) = \mathbb{E}_{a_t\sim\pi}\left[Q(s_t,a_t) - \alpha\ log\ \pi(a_t|s_t)\right] \tag{4}$$

Policy network is trained minimizing eq. 5

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim\mathcal{D}}\left[\mathbb{E}_{a_t\sim\pi_\phi}[\alpha\ log\ (\pi\phi(a_t|s_t)) - Q_\theta(s_t,a_t)]\right] \tag{5}$$

For helping the optimization phase, made by backpropagation, a reparametrization trick (Kingma et al., 2013) is applied.

3 Soft Actor Critic with automatic temperature tuning

Choosing the optimal temperature α parameter is non-trivial and it need to be tuned for each task. Reward and entropy can vary a lot for each task and so also the temperature parameter that establish the relative importance of the two. Moreover entropy and rewards vary during training while the policy improve. (Haarnoja et al., 2018b) proposed an automated process for tuning this parameter formulating a maximum entropy reinforcement learning objective, where the entropy is treated as a constraint.

$$\max_{\pi 0:T} \mathbb{E}_{\rho_\pi}\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t)\right] \tag{6}$$

$$s.t.\ \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[-log(\pi_t(a_t|s_t))\right] \geq \mathcal{H},\ \forall t \tag{7}$$

where $\mathcal{H}$ is the desired minimum expected entropy.
Since the policy at time t can only affect the future objective value, a sort of dynamic programming approach can be used, solving for the policy backward through time. Exploiting then the duality theorem, it's possible to formulate eq. 8 for obtaining the optimal α.

$$\alpha_t^* = \underset{\alpha_t}{argmin}\, \mathbb{E}_{a_t \sim \pi_t^*} \left[-\alpha_t log \pi_t^*(a_t|s_t, \alpha_t) - \alpha_t \bar{\mathcal{H}} \right] \tag{8}$$

where $\bar{\mathcal{H}}$ is the target entropy.

4 Averaged Soft Actor Critic

Soft Q-function tends to overestimate the value which will result in an "excessive" estimated sampled action. SAC will tend to choose the current action with the best performance, although this action is overestimated.
(Ding et al., 2021) proposed a simple but effective idea, instead of using the last state-value obtained using eq. 4, they proposed to use the average of the last K ones. So eq. 3 becomes eq. 9. This technique reduces the variance, improving training stability and agent performance.

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(s_t, a_t) - \left(r(s_t, a_t) + \gamma \frac{1}{K} \sum_{k=1}^{K} V_{\bar{\theta}}(s_{t+1}) \right) \right)^2 \right] \tag{9}$$

In the paper they used the old version of soft actor critic, so the one which used also the value function network. I tried to experiment with the new one which uses the 2 soft Q-functions.

5 Experiments

I tested these 3 algorithms using the HalfCheetah-v4 and the Humanoid-v4 environments. Since training these systems can be computational expensive, I didn't trained them till the convergence. In the first case I established a fixed maximum number of training steps equal for the 3 algorithms. Since in HalfCheetah we have 1000 steps for each episode, fixing the number of training steps is equal to choose a maximum number of training episode. Instead of updating the networks parameters every steps I choose to update them every N steps with $N = 5$, so 200 updating steps for each episode. The maximum number of training steps was equal to $1.2 \cdot 10^6$ equivalent to 6000 episodes.
In the second case since the episode can end before the 1000^{th} step, there's no relation between number of steps and episodes. Here I increased the maximum number of steps to $1.5 \cdot 10^6$ and I fixed the maximum number of episodes to 10^4, moreover I set a sort of early stopping after 750 episodes without improvement in the test reward.
In both cases the model was evaluated every 10 episodes and the test reward was computed as the averaged of the reward of the model in 20 episodes.
The hyper-parameters used for training (Tab. 1) are the same used by (Haarnoja et al., 2018b).
The replay buffer size was reduced from 10^6 samples to $2 \cdot 10^5$, moreover before starting the training I filled the buffer with a minimum number of samples obtained with random actions.
In the HalfCheetah environment both the variants achieved better results. The tuning of the α parameter seems to provide the best improvement even if in the first phase it's the one that performs worse due to the too big initial value of α chosen. Probably a value closer to the optimal one, 0.25 would have make the initial settling phase quicker. The averaged version instead provide way better performances in the first phase while this gap becomes smaller toward the end, this is probably due to the fact that in the initial phase the Q-Network generates value with a great standard deviation since the initial value are more or less randomic. This issue is reduced by the averaging technique proposed. Which is the best variant is difficult to say, however from this analysis seems that with a limited amount of time/resources, averaged-SAC achieve faster good enough performances. Probably in the long run the tuning of temperature closes this gap and achieves the best performances.
Humanoid is a way harder environment, due to the bigger action and observation spaces the number of variable increase a lot. Here the situation changed, both SAC with fixed α and with its tuning ended their training respectively at episode 6670 and 8490 after 750 episodes without any improvement, while averaged-SAC reached $10k$ episode. This fact seems to confirm the previous hypothesis. Averaged-SAC performs better at the beginning and it allowed it to learn faster to have the possibility to survive till the end. The other two algorithm instead weren't enough stable to survive. However

Table 1: Hyper-parameters value

Hyper-parameter	Value
Optimizer	Adam
Learning rate	$3 \cdot 10^{-4}$
Discount (γ)	0.99
Replay buffer size	$2 \cdot 10^5$
Minimum buffer size	10^4
Number of hidden layers (all networks)	2
Number of hidden units per layer	256
Number of samples per mini-batch	256
Entropy target	-dim($\mathcal{A}$)
Non-linearity	ReLU
Target smoothing coefficient (τ)	0.005
α HalfCheetah (not applicable for SAC sec. 3)	0.2
α Humanoid (not applicable for SAC sec. 3)	0.1
K (only applicable for Averaged-SAC sec. 4)	10

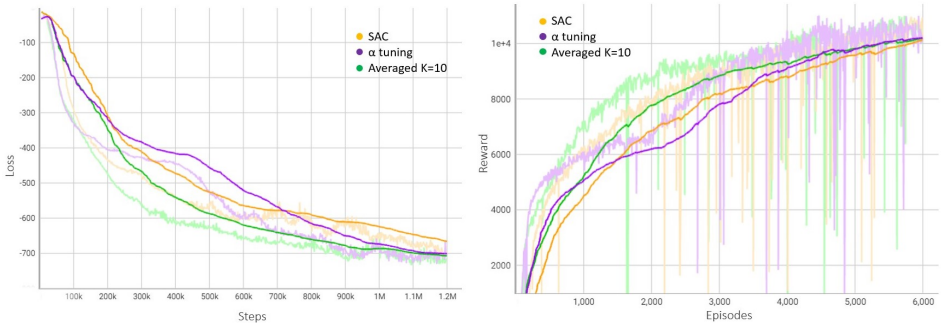
Table 2: Best test reward achieved with each algorithm in the different environments.

	Test reward		
Environment	SAC	SAC α tuning	Averaged-SAC
HalfCheetah-v4	10182.9	11348.2	11141.6
Humanoid-v4	5885.8	6522.6	7972.3

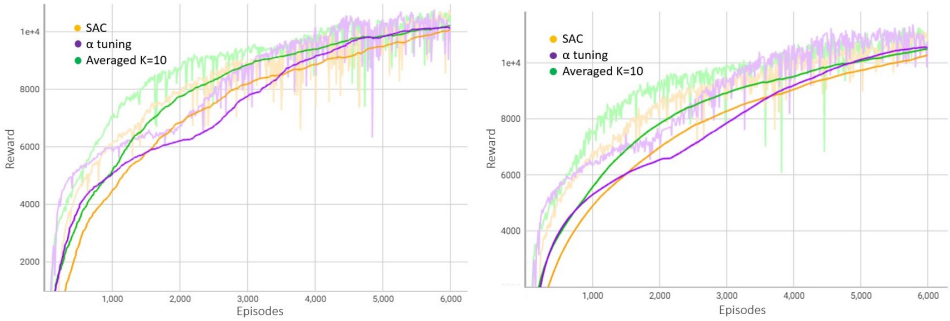
tuning the temperature improve the performance. As in the previous case, the initial α value was too far from the optimal one 0.065, this required many episodes for settling while in the averaged case the initial value was already close enough to provide good results.

6 Conclusion

SAC and in particular its variants provided good results even with limited resources. In the first case SAC appears more stable due to the easier task and seems that toward the end all 3 algorithms tend to achieve similar results. In the second case the convergence requires a greater number of steps, since the task is harder the differences become more evident. With tuning of α, first episodes are mostly related to find a good value of it before the system start to really learn (The purple plots are shifted with respect to the orange ones). Averaged-SAC instead present a similar behaviour to SAC but the introduction of the average makes the the learning process of the soft Q-networks smoother, it makes the systems to improve a bit at each episode but it keeps to learn instead of having great improvement at random position as in SAC.
A possible future experiments would be to combine both technique to obtain the benefit of averaged-SAC in the first phase and the benefit of α tuning in the second ones.
Another possible experiments would be to modify the factor that regulates the importance of the various components of the reward in the Humanoid-v4, to make it more similar to the humans in the first phase we should give more importance to the capability of standing up and only after that increasing the importance of the moving forward ability.

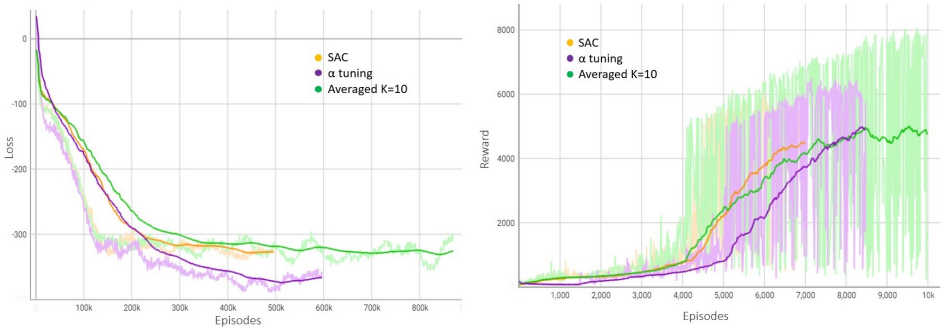


(a) Policy loss (b) Training reward

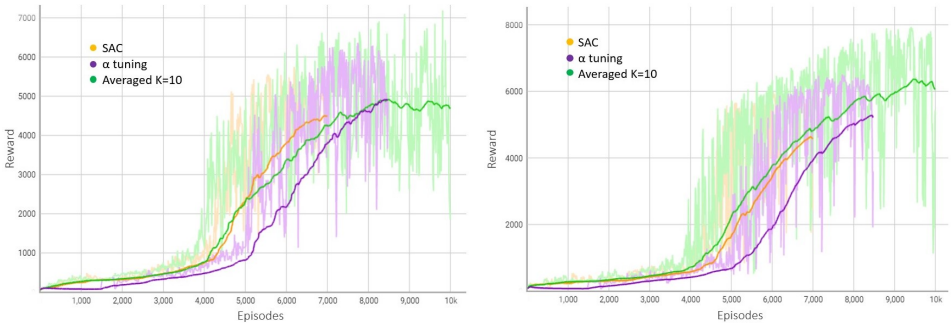


(c) Average training reward (d) Test reward

Figure 1: Plots related to HalfCheetah-v4 environment.



(a) Policy loss (b) Training reward



(c) Average training reward (d) Test reward

Figure 2: Plots related to Humanoid-v4 environment.

References

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W., "OpenAI Gym". *ArXiv Preprint ArXiv:1606.01540*, 2016

Ding, F., Guanfeng, M., Zhikui, C., Jing, G. , & Peng, L., Averaged Soft Actor-Critic for Deep Reinforcement Learning. *Complexity* vol. 2021, Article ID 6658724, 16 pages, 2021.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *International Conference on Machine Learning*, 2018a.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., & Levine, S. Soft Actor-Critic Algorithms and Applications. *ArXiv, abs/1812.05905*, 2018b

Kingma, D., & Welling, M. Auto-Encoding Variational Bayes. *International Conference on Learning Representations*, 2013.

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

Tassa, T., Erez, T. & Todorov, E., Synthesis and stabilization of complex behaviors through online trajectory optimization, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4906-4913, 2012

Todorov, E., Erez, T., & Tassa, Y.,, MuJoCo : A physics engine for model-based control". *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026-5033, 2012

Wawrzynski, P., A Cat-Like Robot Real-Time Learning to Run. 380-390. 10.1007/978-3-642-04921-7$_3$9, 2009

A Reparametrization trick

In SAC algorithm we are interested in computing the gradient of the expected value of a function.

$$\nabla_\theta \, \mathbb{E}_{x \sim p_\theta(x)}[f(x)] \tag{10}$$

The problem in evaluating eq. 10 is the fact that the expectation is taken wrt a distribution with parameters θ and we can't compute the derivative of this quantity unless we have an analytic solution. Thanks to reparameterization trick we can rewrite the samples of the distribution p_θ in terms of a noise variable ϵ independent of θ.

$$\epsilon \sim q(\epsilon) \tag{11}$$

$$x = g_\theta(\epsilon) \tag{12}$$

$$\nabla_\theta \, \mathbb{E}_{x \sim p_\theta(x)}[f(x)] = \mathbb{E}_{\epsilon \sim q(\epsilon)}[\nabla_\theta f(g_\theta(\epsilon))] \tag{13}$$

The variable x is reparamterized as a function of ϵ and the stochasticity is assumed by the distribution $q(\epsilon)$, where q can be any random noise distribution. As always the Gaussian distributions the most used and it is also the distribution chosen by SAC. We can assume x is sampled from a Gaussian, $x \sim \mathcal{N}(\mu, \sigma)$. The function $g_\theta(\epsilon)$ then can be defined as the following:

$$g_\theta(\epsilon) = \mu_\theta + \epsilon \sigma_\theta \tag{14}$$

where $\epsilon \sim \mathcal{N}(0, 1)$.
The trick changed the expectation to a distribution independent of θ and if $f(g_\theta(\epsilon))$ is differentiable with respect to θ, we can compute it using Monte Carlo (eq. 15).

$$\nabla_\theta \, \mathbb{E}_{x \sim p_\theta(x)}[f(x)] \approx \frac{1}{N} \sum_{i=0}^{N} \nabla_\theta f(g_\theta(\epsilon_i)) \tag{15}$$

Reparametrization was preferred because present a lower variance than REINFORCE.