Reinforcement Learning Implementation Project: Soft Actor-Critic applied to Ant-v2 environment

Leandro Maglianella – 1792507

Lorenzo Nicoletti – 1797464

Contents

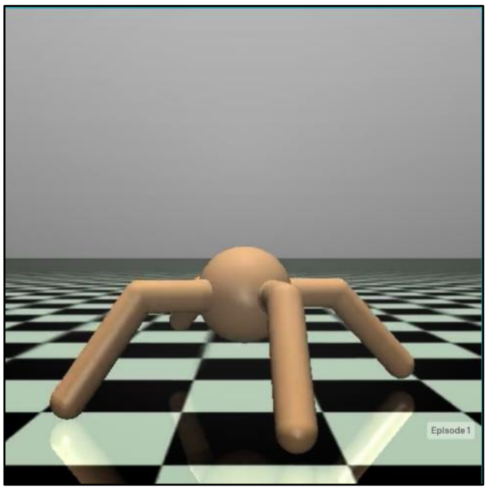
Abstract

The proposed work is an implementation of Soft Actor-Critic, which is one of the major state-of-the-art algorithms in the field of Reinforcement Learning. This application has been implemented, trained and tested on the ‘Ant-v2’ environment that belongs to the OpenAI Gym MuJoCo tasks family. The results achieved in this benchmark will be the main focus of this paper and they will be discussed in the following sections. The framework used for the experiments is Python v3.7 with Tensorflow v2.4.1 and Gym v0.18.0.

Introduction

The environment that has been adopted for the training, the testing and the evaluation of our implementation is ‘Ant-v2’, a four-legged creature characterized by a:

- 111-dimensional observation space, resulting as the sum of the height of the torso, four quaternions for the orientation, eight joint angles, two three-dimensional direction and angular velocities, eight joint velocities and a total of eighty-four external six-dimensional forces applied on each of the fourteen links.
- 8-dimensional action space, given by the torque value (between -1 and +1) of each joint.
- Reward function, that considers the task completed with an average reward of 6000.0 over 100 consecutive trials.



The creature’s goal is obtaining the ability of walking forward as fast as possible: the chosen algorithm to accomplish this learning task is Soft Actor-Critic, which is based on an off-policy maximum entropy learning method.

In the ‘Related work’ section, the origins and design of this method will be formally analyzed in a mathematical way to understand the reasons that favored its spread. Then, in the ‘Experiments and Results’ section, we will see in particular the code we developed, with our adaptations and additions, and finally the results we were able to achieve will be shown and evaluated.

Related work

Maximum Entropy Reinforcement Learning: Usually, in standard Reinforcement Learning algorithm the task is to learn a policy $\pi^*(a_t|s_t)$ that maximize the expected sum of rewards $J(\pi)$. Since we are applying stochastic policy, we have to consider also the consistent presence of entropy that modifies the formula as follows:

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

where α is the temperature parameter that determines the importance of the Entropy H term, consequently also controlling the stochasticity of the procedure. The entropy is commonly defined as:

$$H(P) = \mathbb{E}_{x\sim P}[-\log(P(x))] \tag{2}$$

where x is a random variable with distribution P.

Soft Policy Iteration: In this algorithm, the goal is to compute the optimal maximum entropy policy. First of all, the value of policy π is estimated with reference to (1): considering any function $Q\colon \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ and the fixed policy π, we can derive the soft Q-value iteratively by applying multiple times a modified Bellman backup operator $\mathcal{T}^\pi$:

$$\mathcal{T}^\pi \mathrm{Q}(s_t,a_t) \triangleq \mathrm{r}(s_t,a_t) + \gamma\, \mathbb{E}_{s_{t+1}\sim \mathrm{p}}\,[\mathrm{V}\,(s_{t+1})] \tag{3}$$

where

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

is the soft state value function.

Secondly, we improve the policy using the following equation, for which it is demonstrable that it has a higher value than the old policy with respect to the maximum entropy objective:

$$\pi_{new} = \arg\min_{\pi'\in\Pi} D_{KL}\left(\pi'(\cdot\,|s_t)||\frac{\exp(\frac{1}{\alpha}Q^{\pi_{old}}(s_t,\ \cdot))}{Z^{\pi_{old}}(s_t)}\right) \tag{5}$$

These evaluation and improvement phases are repeated until the optimal maximum entropy policy is found. This algorithm is particularly robust but has a downside: it can be executed in its exact form only in a tabular case. Because of this, we will need to approximate the algorithm for continuous domains, to rely on a function approximator to represent the Q-values and computing the two steps until convergence is computationally too expensive. The approximation caused a new practical algorithm to come about, called Soft Actor-Critic.

Soft Actor-Critic: As mentioned before, our implementation of SAC algorithm uses function approximators for both the soft Q-function and the policy: these are structured as neural networks and the learning alternates between them, optimizing with stochastic gradient descent. We consider a parametrized soft Q-function $Q_\theta(s_t,a_t)$ and a tractable policy $\pi_\phi(a_t|s_t)$. For the first one's parameters, the training aims to minimize the soft Bellman residual:

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}\Big(Q_\theta(s_t,a_t) - \big(r(s_t,a_t) + \gamma\, \mathbb{E}_{s_{t+1}\sim \mathrm{p}}\,[\mathrm{V}_{\bar{\theta}}\,(s_{t+1})]\big)\Big)^2\right] \tag{6}$$

where $\mathrm{V}_{\bar{\theta}}\,(s_{t+1})$ is parametrized using (4).

Instead, the policy parameters are trained by minimizing the expected KL-divergence in (5):

$$J_\pi(\theta) = \mathbb{E}_{s_t\sim\mathcal{D}}\left[\mathbb{E}_{a_t\sim\pi_\phi}\left[\alpha \log\left(\pi_\phi(a_t|s_t)\right) - Q_\theta(s_t, a_t)\right]\right] \tag{7}$$

An important feature used here to optimize the policy is the reparameterization trick that consists in a sampling from $\pi_\phi(\cdot\,|\mathrm{s})$ according to some deterministic function, policy parameters and noise. Following the paper, we have used a squashing function to sample as follows:

$$\tilde{a}_\theta(s, \xi) = \tanh(\,\mu_\theta(s) + \sigma_\theta(s) \odot \xi), \qquad \xi \sim \mathcal{N}(0, I) \tag{8}$$

The $tanh$ function ensures that actions are bounded to a finite range. Then, it is possible to rewrite (7) by substituting every occurrence of a_t with the reparametrized version described in (8).

Finally, Soft Actor-Critic applies the so-called min-double-Q trick to get the policy loss by considering the minimum of the two Q approximators estimated by the algorithm that in our implementation correspond to the output of the Critic Neural Networks.

Experiments and Results

The proposed implementation of Soft Actor-Critic contains all the key aspects that define the algorithm: the separation between the policy and the value function networks, an off-policy design to re-use previously collected data samples stored in a replay buffer filled at training time, and the maximization of the entropy in order to ensure exploration and robustness. In particular, we have modeled the policy network with the Actor class and the value function network with the Critic class. Both Actor and Critic has been abstracted in Neural Networks with a predefined number of hidden units and layers. The idea behind this approach is very intuitive: the two components, although separate from each other, are able to interact with each other in such a way that Actor can exploit Critic's indications to improve the efficiency of the policy and that Critic can subsequently evaluate the actual improvement dictated by these changes.

In our work we have tried different combinations of hyperparameters values but, at the end, our experiments have indicated that the best combination is the one summarized in the following table:

Parameters	**Values**
Optimizer	Adam
Learning rate	$1 \cdot 10^{-3}$
Gamma	0.99
Alpha	0.3
Replay buffer size	10^6
Nr. of hidden layers	2
Nr. of hidden units per layer	256
Non-linearity	ReLU
Polyak coefficient	0.99

Our implementation's main task is to show consistently how Soft Actor-Critic has improved the performances of the agent by comparing a randomized action-selector agent against a trained agent using the algorithm. To support our thesis, we have run the first agent and observed its behavior, noticing how it objectively failed if it tried even to make a simple step forward with a final result of an overturn of the creature. To improve this performance, we have experimented two types of trainings inspired by the provided implementations of the paper and of the SpinningUp Documentation whose pseudocodes are attached to the Appendix at the end of this report. We have noticed that the first training is relatively fast and simple, while the second one is very expensive but produces more robust results.

Since 'Ant-v2' is a particularly difficult environment, characterized by a high-dimensional action and observation spaces, the results we were able to obtain are not optimal, but they show a consistent improvement in the behavior of the agent. Then, to prove the correctness of our implementation of Soft Actor-Critic, we have tested the same algorithm on a different and simpler environment: 'MountainCarContinuous-v0'. It is an OpenAI Gym Classic control environment consisting in a one-dimensional car-truck with the goal of driving up a mountain and reaching the uppermost position. We have tested our implementation combined with the first type of training based on the paper and we have reached incredibly high performances, factor that makes us believe in the quality of the algorithm. To testify it, we report below in Figure 1 and 2 the evolutions of the learning of the agent obtained in the two adopted environments, 'MountainCarContinuous-v0' and 'Ant-v2', in terms of episode lengths, episode rewards and losses collected over time.

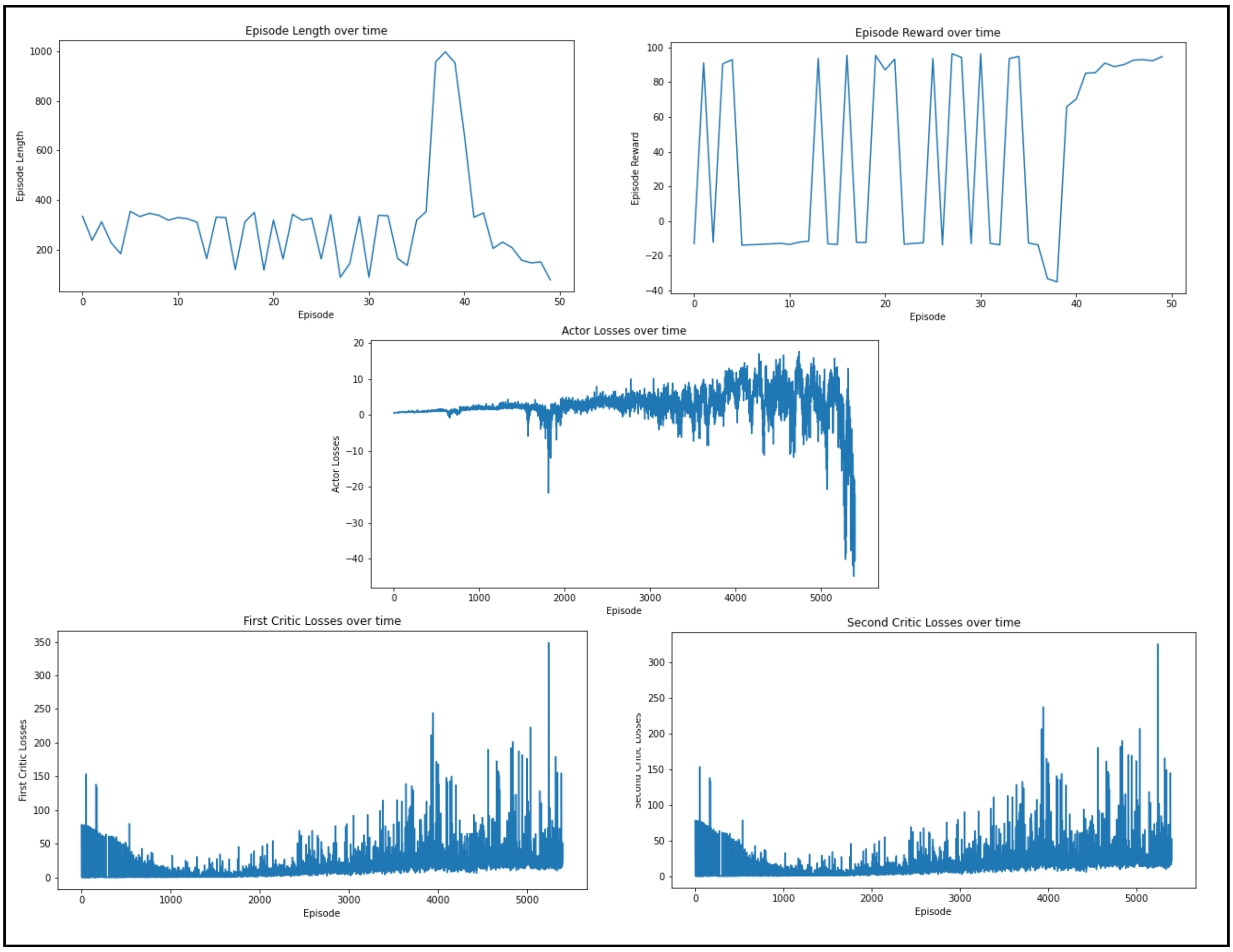


Figure 1) Episode Lengths/Rewards over time, Actor loss and the two Critics losses in 'MountainCarContinuos-v0'.

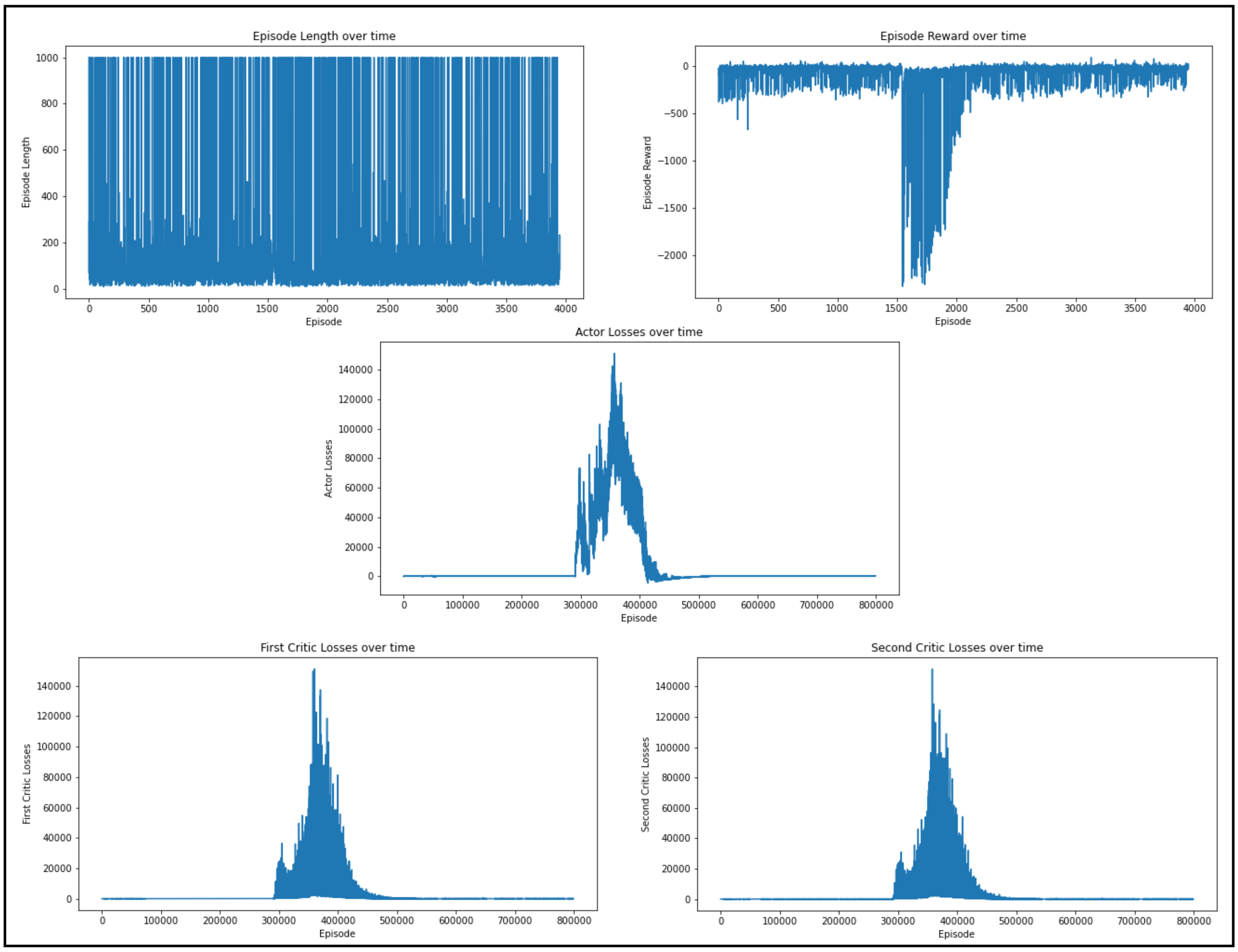


Figure 2) Episode lengths/Rewards over time, Actor loss and the two Critics losses in 'Ant-v2'.

In ‘MountainCarContinuous-v0’, we have reached the expected results even with a relatively small (≈ 15) number of training episodes. As registered by the first set of plots, the cumulative reward is characterized by a final “almost-constant” region that suggests that the learning seems to be accomplished, also the required episode lengths over time graph suggests how the learning agent can complete the task is an incredibly low number of steps. This conclusion is also proved by the output at the testing stage of our experiment, where the car is able to reach the goal effortlessly by building up a sufficient momentum to overcome the hill.

A similar reasoning can be considered for ‘Ant-v2’ environment. The plots testify that the rewards collected over time are not so positive but, on the other hand, the most interesting result is achieved for the losses: they all converge to a zero-value, reaching a stability for our network models. The testing stage has suggested some exciting results too. In fact, the creature is now able to proceed forward on the same direction, the motion is not perfect, but it seems that the agent has acquired the capability of performing consecutive steps with higher velocity and self-confidence and a superior equilibrium than before (it manages also to understand it is about to overturn and to correct its trajectory), without the risk of falling for at least a reasonable number of steps.

We are conscious that these performances are still hugely improvable but at the end we have demonstrated how Soft Actor-Critic is able to improve the behavior of the creature in its environment, creature that has proved the ability to learn and to adjust its motion.

Conclusions

In conclusion, after having described the environment in which we have worked and exposed the origins and general behavior of Soft Actor-Critic, we have discussed our take on its implementation and showed our results that have demonstrated the improvements that a powerful algorithm like this is able to introduce in the training of a Reinforcement Learning agent. Although our outcomes are not optimal, they are reasonably sufficient to prove our starting thesis: how good a state-of-the-art method like Soft Actor-Critic can improve the performances of an agent. In the end, we have been capable to provide a consistent implementation of the original paper and some practical results based on the theory and the reasoning the whole algorithm is built on.

References

- Berkeley Artificial Intelligence Research, University of California, Berkeley, USA. (2018) Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. Available at: https://arxiv.org/pdf/1801.01290.pdf (Last Accessed: 3 February 2021).

- Capobianco R. (2020) Reinforcement Learning: SAC lesson. Available at: https://www.youtube.com/watch?v=HaZiurNC3Fc&t=3240s (Last Accessed: 31 January 2021).

- Capobianco R. (2020) Q-learning with Neural Networks Colab Notebook. Available at: https://colab.research.google.com/drive/1FJwiI3l7iYUqVWNIb4-qAy8CYHOAy7z3 (Last Accessed: 2 February 2021).

- Haarnoja’s Soft Actor-Critic implementation. (2018) Available at: https://github.com/haarnoja/sac (Last Accessed: 28 January 2021).

- OpenAI. (2018) Soft Actor-Critic. Available at: https://spinningup.openai.com/en/latest/algorithms/sac.html# (Last Accessed: 2 February 2021).

- OpenAI Spinningup’s Soft Actor-Critic implementation. (2020) Available at: https://github.com/openai/spinningup/tree/master/spinup/algos/tf1/sac (Last Accessed: 2 February 2021).

- UC Berkeley & Google Brain. (2019) Soft Actor-Critic Algorithms and Applications. Available at: https://arxiv.org/pdf/1812.05905.pdf (Last Accessed: 28 January 2021).

Appendix

The following pseudocodes describe the two Soft Actor-Critic implementations provided respectively by the paper we have studied for the project and by the SpinningUp Documentation.

Algorithm 1 Soft Actor-Critic

```
Initialize parameter vectors $\psi$, $\bar{\psi}$, $\theta$, $\phi$.
for each iteration do
    for each environment step do
        $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
        $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
    end for
    for each gradient step do
        $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
        $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ 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 for
end for
```

Algorithm 1 Soft Actor-Critic

```
1:  Input: initial policy parameters $\theta$, Q-function parameters $\phi_1$, $\phi_2$, empty replay buffer $\mathcal{D}$
2:  Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
3:  repeat
4:      Observe state $s$ and select action $a \sim \pi_\theta(\cdot|s)$
5:      Execute $a$ in the environment
6:      Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
7:      Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
8:      If $s'$ is terminal, reset environment state.
9:      if it's time to update then
10:         for $j$ in range(however many updates) do
11:             Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:             Compute targets for the Q functions:
                $$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$
13:             Update Q-functions by one step of gradient descent using
                $$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$
14:             Update policy by one step of gradient ascent using
                $$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)\middle| s\right)\right),$$
                where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt $\theta$ via the
                reparametrization trick.
15:             Update target networks with
                $$\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$
16:         end for
17:     end if
18: until convergence
```