SUCCESSOR OPTIONS REPRODUCED

Matthias Hutsebaut-Buysse, Ruifeng Ma

ABSTRACT

This report is our submission to the 2019 ICLR Reproducibility Challenge. In this report we present our findings in reproducing work presented in the paper *Successor Options : An Option Discovery Algorithm for Reinforcement Learning*. This paper presents a novel approach on solving the issue of option discovery. We were able to find similar results for the claims we examined, however we did not replicate all presented results.

1 INTRODUCTION

In this report we reproduce a subset of the experiments presented in (Tomar et al., 2018) in the context of the 2019 ICLR Reproducibility Challenge [1].

The research we have chosen to partially reproduce situates itself in the area of Hierarchical Reinforcement Learning (HRL). This area of Reinforcement Learning (RL) attempts to learn goal-achieving behavior using only a feedback signal provided by the environment (the reward). This behavior is learned by only using experience gained from interaction with the environment. In an hierarchical system this behavior is typically a composition of multiple sub-behaviors.

These sub-behaviors are often formalized using the options framework (Sutton et al., 1999). An option $(w \in \Omega)$ consists of an initiation set I_w, consisting of all states from which the option can be invoked. A termination set β_w consisting of states which halt the execution of the option. And an option policy π_w decides what action to take given a state.

Automatically discovering options is currently an ongoing challenge in RL. A popular approach to discover options is to focus on the termination-set β_w, and learn a corresponding π_w which is able to efficiently navigate to a goal-state.

The selected research presents a novel approach in tackling this challenge. The authors claim the following main contributions:

- Introduction of a new model called *Successor Options* (SO) that leverages a learned *Successor Representation* (SR) to detect states that are suitable as termination states β_w in the options framework
- A new pseudo-reward for learning the intra-options policies π_w
- An incremental approach that is able to explore states where using primitive actions is inadequate to form a SR.

In this report we present our findings in reproducing the experiments that support these claims. The code used in this report can be found on the following link: https://github.com/mhtsbt/SuccessorOptions

2 SUCCESSOR REPRESENTATION LEARNING

The first step in the proposed algorithm, consist of learning the successor representation (SR). A successor representation (Dayan, 1993) represents a state s as a vector of its successor-states. The successor representation can be calculated following Equation 1:

$$\psi_\pi(s, s') = E_{s' \sim P, a \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}_{(s_t = s')} | s_0 = s\right] \tag{1}$$

[1] https://www.cs.mcgill.ca/ jpineau/ICLR2019-ReproducibilityChallenge.html

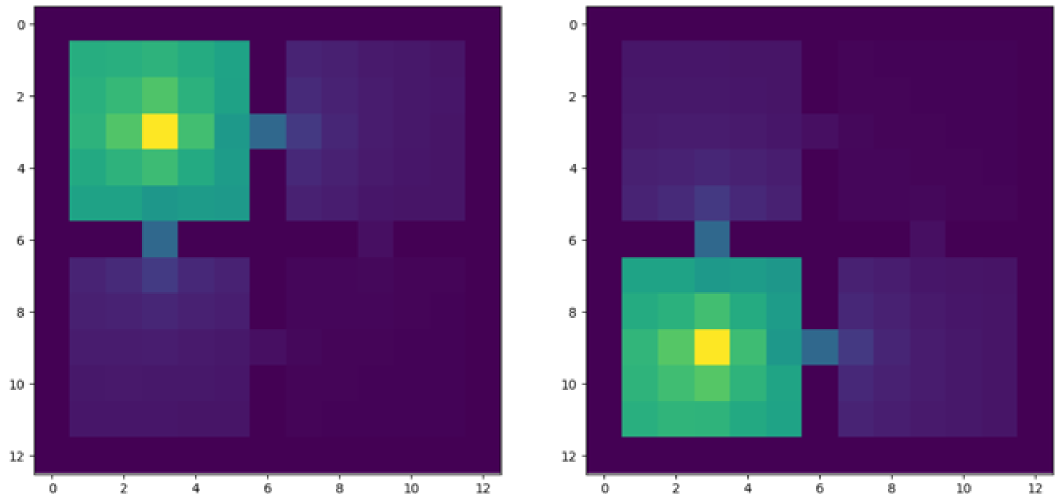

Figure 1: Learned SR for two different states marked in yellow ($\alpha = 0.0025$, $\gamma = 0.99$)

Where $\psi_\pi(s, s')$ defines the successor representation starting from state s to state s' following policy π. $\mathbb{I}_{(s_t=s')}$ is a binary indicator that returns 1 if state s' is visited at time t, and otherwise returns 0. γ is the discount factor which discounts the value for states that are visited in a more distant future.

This representation can be learned in a temporal difference (TD) fashion when gathering experience in the environment (Equation 2) following a policy π. Let $\hat{\psi}(s, :)$ denote the estimated SR of state s, updated by $\alpha \left[1_s + \gamma \left[\hat{\psi}(s', :)\right] - \hat{\psi}(s, :)\right]$ in each iteration. Where $\mathbb{I}(s)$ is the indicator-function which is a zero-vector with the exception of 1 in state s.

$$\hat{\psi}(s, :) \leftarrow \hat{\psi}(s, :) + \alpha \left[1_s + \gamma \left[\hat{\psi}(s', :)\right] - \hat{\psi}(s, :)\right] \tag{2}$$

Learning the SR in a TD-style requires a learning-rate α, which was not provided in the original research. We conducted a hyperparameter search, and presented some qualitative results in 2. We used a discounting factor of $\gamma = 0.99$ which has provided in the research. Figure 1 visualizes the SR for two different states (marked in yellow).

In the first iteration the policy will act completely random, exploring as much of the state space as possible. A random sampling policy with a uniform action-selection distribution was run for 5×10^6 steps.

As demonstrated in our small qualitative analysis visualized in Figure 2, the quality of the learned SR, and the resulting sub-goals heavily depends upon selecting a good α learning-rate. Setting α too high will render the SR unsuitable to be used as a pseudo-reward used in learning different option-policies in the next step. Setting α too low will not allow the SR to become developed enough to select good canidate subgoals. In our fourroom-environment setting $\alpha = 0.0025$ seems to result in the most spread-out sub-goals, and provided us with a dense reward-map (pictured in the bottom part of Figure 2)

We also experimented with the amount of random steps that the agent needs to take in order to learn the SR efficiently. If we lower the amount proposed by the original paper(5×10^6 steps), we cannot learn any useful sub-goals from the SR.

3 SUB-GOAL DISCOVERY

In the second stage of the successor options framework the learned SR is used to find sub-goals. Navigating to these sub-goals should make learning to navigate to environmental-goals faster for a RL agent.

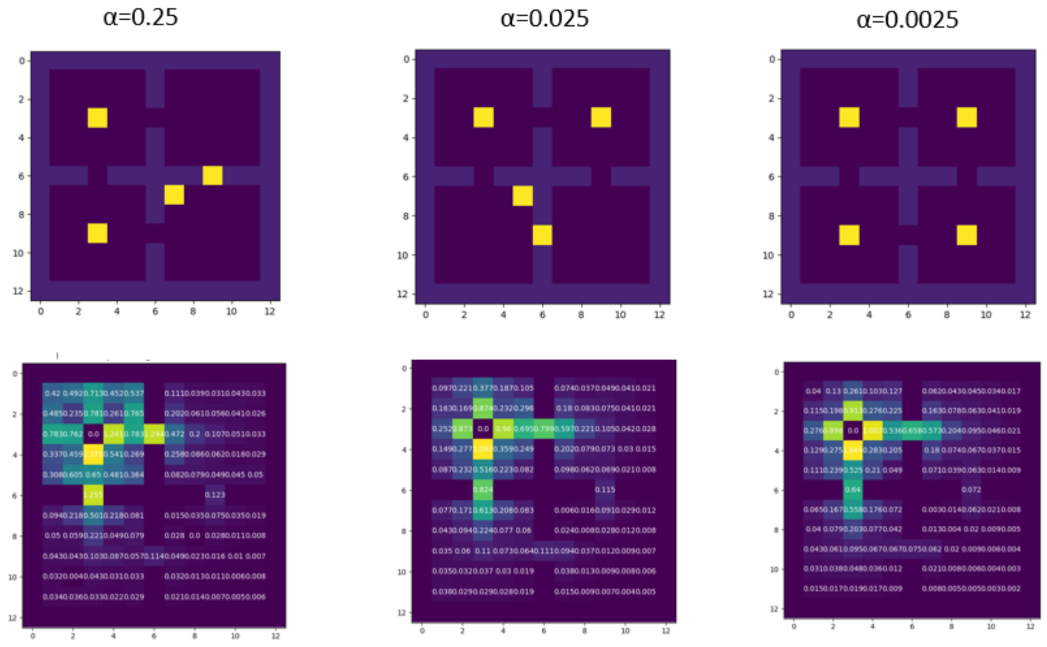

Figure 2: Different values for α compared (5×10^6 rollout samples used). The top-pane shows the found sub-goals, the bottom-pane shows a reward map for the sub-goal discovered in the top-right corner.

The examined research proposes using K-Means++ to cluster states with a similar SR. The cluster centers are translated to sub-goals (using cosine distance) that have vastly different successor states, meaning different sub-goals provide access to different regions of the state-space. The amount of clusters is equal to the amount of desired options. This is a typical hyper-parameter in a options context. As visualized in Figure 2 this clustering technique assigns the centers of the rooms as sub-goal states. We also tested this clustering technique on different environments (Figure 3), with varying amounts of options.

The proposed algorithm suggest filtering out states which do not have a good enough developed SR. We however did not used this filtering as with the amount of samples used, our SR seemed sufficiently developed to find suitable sub-goals.

We found that the presented method seems to be able to find sub-goals which are spread-out across the environment. It however is difficult in this stage to assess whether these sub-goals will allow our agent to better navigate the environment.

4 INTRA-OPTION POLICY LEARNING

Once we have found suitable sub-goal states which we can use as termination-states for options, we can now learn a policy that is able to navigate our agent to these different sub-goals. In the SO-algorithm each option is allowed to train for 1×10^6 steps in the four-room environment. Options terminate when $Q(s, a) \leq 0$ (there is no improvement possible anymore by taking an action)

As a reward signal we use:

$$r_{\psi_s}\left(s_t, a_t, s_{t+1}\right) = \psi_s\left(s_{t+1}\right) - \psi_s\left(s_t\right) \tag{3}$$

Where $r_{\psi_s}\left(s_t, a_t, s_{t+1}\right)$ denotes the reward signal at state s_t, taking action a_t and turning into result state s_t. The reward signal is given by the difference between the SR of state s_{t+1} and s_t.

This reward-signal provides a dense reward as visualized in Figure 4. Which can encourage the agent to move to the expected target. Therefore standard Q-Learning can be applied to learn a

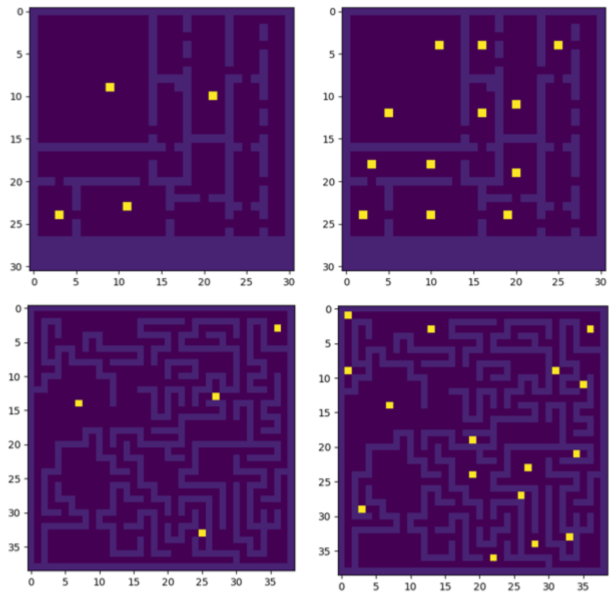

Figure 3: Sub-goals found in different gridworld environments (5×10^6 rollout samples used)

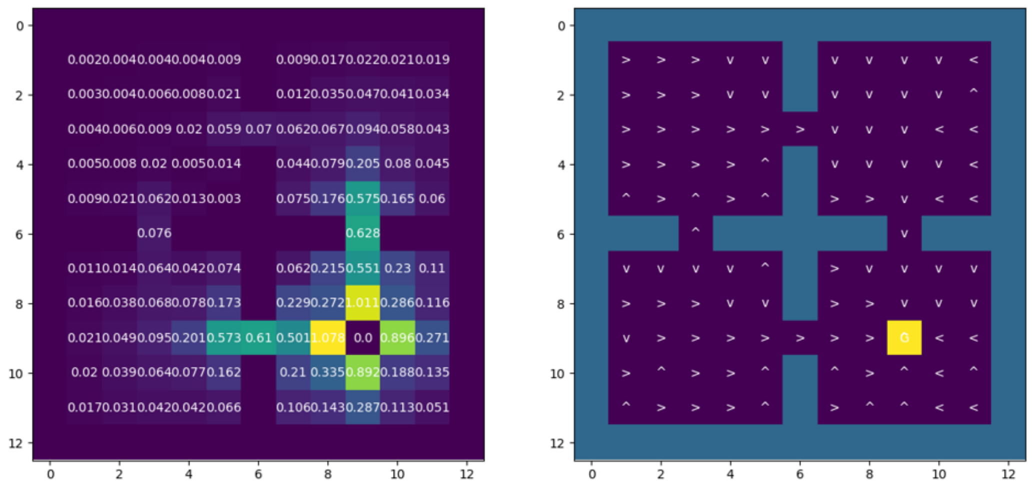

Figure 4: Left reward-map for sub-goal. Right: corresponding option-policy after 10^6 training steps ($\epsilon = 0.3, \gamma = 0.99, learningrate = 0.25$)

policy to reach the sub-goal in an efficient way. In our experimentation we used a fixed exploration rate $\epsilon = 0.1$

5 Policy over options

In order to truly measure the benefit of using SO as a way of navigating an environment we test this claim by averaging performance on 100 randomly generated scenarios in the four-room gridworld. Each scenario has a random start-position and a random goal-position. The environment is modelled as an SMDP-problem: the primitive action-space (up, down, left, right) is extended with the options. This entails that the agent is able to both utilize primitive actions and options.

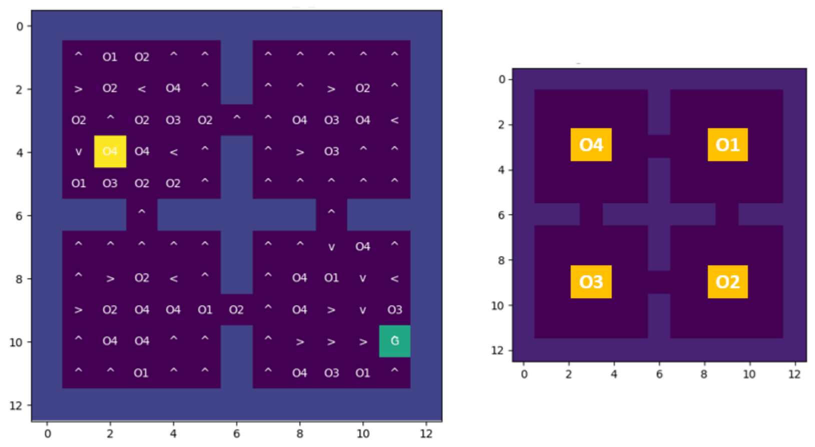

Figure 5: Left: an example scenario, the starting position in marked in yellow, the goal is marked in green. Right: labeled termination-states of the different options ($\epsilon = 0.1$, $\gamma = 0.99$, learning rate=0.25)

In order to solve these scenarios a separate Q-table is learned for each scenario during 5×10^5 training steps. We used the same fixed exploration rate ($\epsilon = 0.1$) as during the training of different option-policies. An example scenario is displayed in Figure 5.

We measure the performance of our learning agent in a similar way as presented in the research: we measure the performance averaged over 100 random scenarios. For each episode we measure the cumulative reward gathered in 200 consecutive training stages (each consisting of 2500 training steps). Our results are visualized in Figure 6

We compare the performance of different option-action sampling ratios (1=use only primitive actions, 0.5=uniformly sample options and actions, 0=use only options). This sampling ratio is important for the agent to efficiently try out options to navigate to different parts of the environment. The primitive actions are a ”last-mile” solution that allow the agent to navigate from a sub-goal to the environment-goal.

We found that the SO method on average is able to solve random scenarios faster than action-only Q-learning, and furthermore that the SO algorithm is able to solve more scenarios. Figure 7 e.g. shows two scenarios which are difficult to solve using primitive actions only. However using well-developed options this scenario can easily be solved.

6 Conclusion

In this report we reproduced a subset of experiments using the successor options architecture. We successfully found that the presented architecture is capable of finding well separated and suitably located sub-goals in a four-room gridworld, using a pre-learned successor representation. We also demonstrated that these options are able to speed up training sparsely rewarded tasks compared to flat Q-Learning.

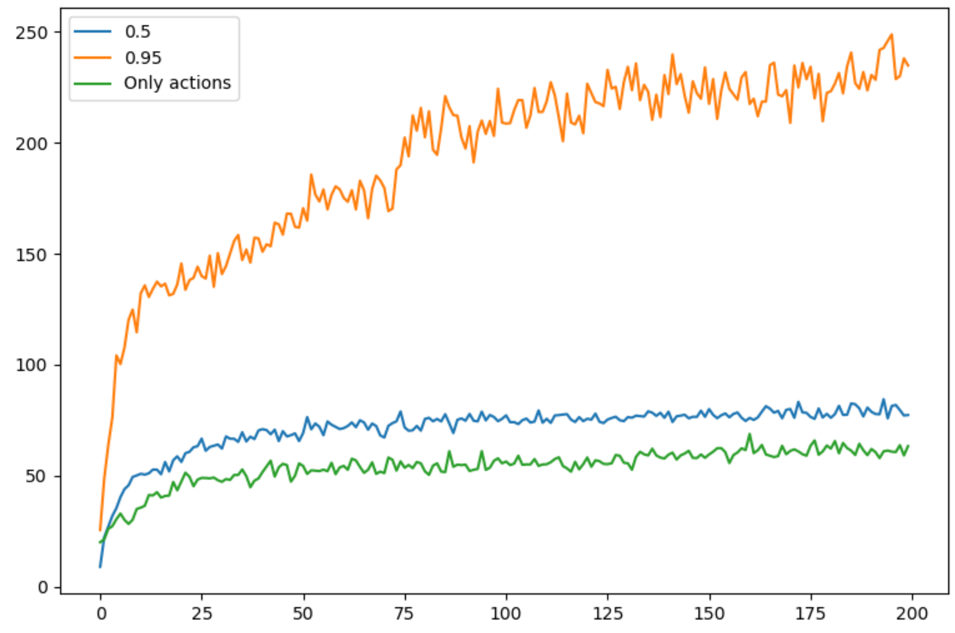

Figure 6: Performance averaged over 100 different scenarios

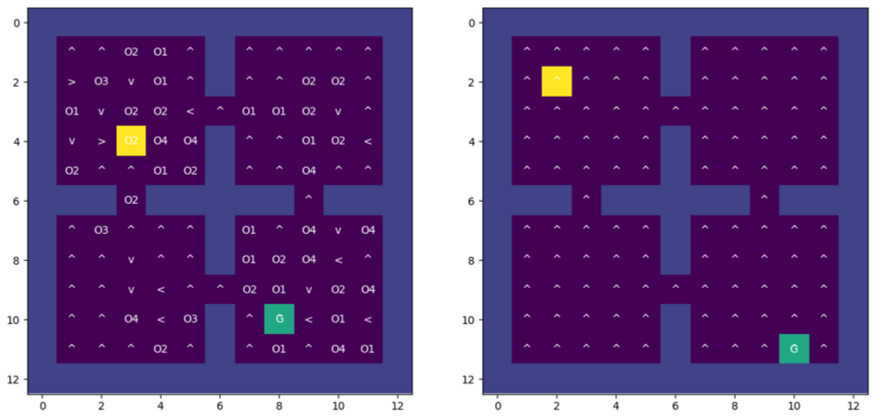

Figure 7: Policies for similar scenarios. The right (action-only) policy failed to learn goal-satisfying behavior, the left policy (action-option sampled in an 1/19 ratio) finds a good policy.

References

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Manan Tomar, Rahul Ramesh, and Balaraman Ravindran. Successor options : An option discovery algorithm for reinforcement learning. Technical report, 2018.