Reinforcement Learning, Master MVA, 2017/2018

Successor Features in Deep Reinforcement Learning and Transfer Learning

Erwan Bourceret - Matthieu Doutreligne

erwan.bourceret@ens-paris-saclay.fr
matthieu.doutreligne@ens-paris-saclay.fr

Abstract

Barreto et al. 2016 present Successor Features as a model allowing to decouple the dynamic transitions from the rewards. Thus, it can easily handle Transfer Learning when the environments of the different tasks are similar. In this project, we review this setup and its development for Deep Reinforcement learning exposed in Lehnert, Tellex, and Littman 2017, Zhang et al. 2017 Kulkarni et al. 2016.

Introduction : The transfer problem

We are interested in the transfer learning problem in the framework of Reinforcement Learning. This is motivated by the need for an agent in RL problems to adapt to new tasks by making use of the knowledge gained on the previous tasks.

More formally, let consider two tasks T, T'. The agent first learns a policy π on the first task T. When the task changes from T to T', we would like the policy π to transfer well to the new task. In other words, we want that π', the new policy, scores well on T' by making use of the knowledge it gained on the first task. Note that we don't want to loose the performance gained on T either. So, π' should perform better or equally on T while also learning the task T'.

What do we mean by a change of task exactly ? In a Grid World example, it could mean that the cells with negative or positive rewards are moved to other

locations from T to T'. For a more visual example, consider a bot navigating in a maze: a change of task corresponds simply to a change of location of the walls in the maze.

In the setting that we consider, the change of task is carried out most of the time by a change of the reward function $r(s, a)$.

1 The Markov Decision Process framework for RL

We briefly recall the Markov Decision Process (MDP) framework which is heavily used in the literature and for the Successor Features development.

We consider an agent interacting with a completely observed random environment. At each time step, we perfectly observed the state occupied by the agent $s \in \mathcal{S}$. The agent chooses an action $a \in \mathcal{A}$, which conditions the probability to enter the next state $s' : \ p(s'|a, s)$. He is then rewarded for this action and according to the current state by $r(s, a)$. Beginning at time t, the goal of the agent is to maximize the cumulative sum of reward discounted by a factor γ: $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$. We want to find the optimal policy π^* which selects the actions maximizing R_t.

We define the MDP $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma\}$ where:

- $\mathcal{S}$ is the state space,
- $\mathcal{A}$ is the action space,
- $\mathcal{P}$ are the transition probabilities, also called the dynamics of the MDP:

$$P^a_{s,s'} = \mathbb{P}(s_{t+1} = s'|s_t = s, a_t = a)$$

- $r(s, a)$ is the reward function.
- γ, the actualization rate.

A well known approach to address the maximal cumulative reward problem comes from Dynamic Programming (DP) and make use of the action-value function defined by:

$$Q^\pi(s, a) = \mathbb{E}[R_t|s_t = s, a_t = a]$$

which follows the Bellman equation,

$$Q^\pi(s, a) = r_{t+1} + \gamma \mathbb{E}_{s',a'}[Q^\pi(s, a)|s_{t+1} = s', a_{t+1} = a']$$

Beginning with a policy π we process in two steps to recover a better policy:
First, we evaluate π with the previous Bellman equation.
Then, we derive a new policy π' greedily with respect to $Q^\pi(s, a)$: $\pi'(s) = \underset{a}{\text{argmax}} Q^\pi(s, a)$.

Alternating between these two steps leads to the desired optimal policy π^*.

We introduce here the example of the "puddle world" presented in Sutton 1996 and Barreto et al. 2016 which is a subset of the Grid World examples. It will provide a recurrent support to illustrate propositions and definitions. The environment is a grid where the agent can navigate step by step and must achieve the goal where the reward is 1 while avoiding two puddles of size 3×1 and 1×3 (one horizontal and one vertical) where the reward is equal to -1. The goal and puddle are absorbing states. See Fig. 1

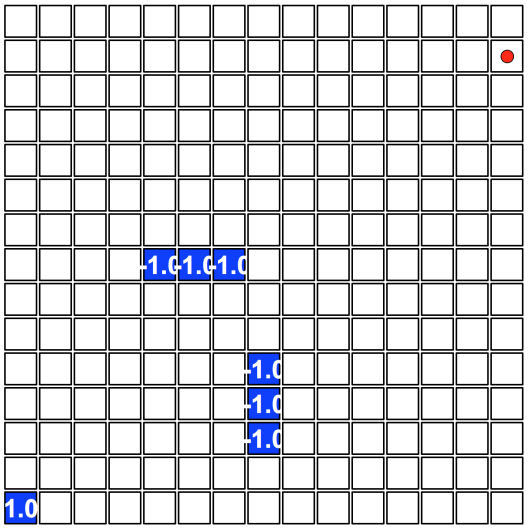

Figure 1: Puddle World. The red point is the agent, actions available are *right*, *left*, *up* and *down* except for borders where the agent cannot exit the grid.

Thus, the goal is to compute the best action for each cell of the grid (state) in order to maximize the cumulative reward, which is equivalent to minimize the path to the goal avoiding puddles.

2 Successor Features (SF)

2.1 SF background and origination

The main idea of *Successor Features* (SF) model originated in Dayan 1993 *Successor Representations* (SR) where each state is represented trough the expected future occupations of all other states with a vector $\mathbf{\Psi}^\pi$ given a policy π. Barreto et al. 2016 present SF as an embedded model of the SR where the vector $\mathbf{\Psi}^\pi$ belong to a much smaller space. In doing so, the method is no longer consistent but has a faster convergence.

2.2 Definition

Before introducing SFs, Barreto et al. 2016 assume a linear model for the reward. In other words, for the state-action combination (s, a) and the reward function r, it exists Φ and w such that :

$$r(s, a) = \Phi(s, a)^T \cdot w$$

with :

- $\Phi(s, a) \in \mathbb{R}^d$ a representation of (s, a)
- $w \in \mathbb{R}^d$ the weighted representation of the reward function associated to Φ

Note that this representation is not restrictive. If we take the representation $\Phi(s, a)$ to be a one-hot vector with a one at the (s, a) position and 0 elsewhere, we can reconstruct exactly the reward function by taking w the weighted vector to have $r(s, a)$ at the position (s, a). Doing so, the dimension d is equal to $|\mathcal{S}| \times |\mathcal{A}|$. This model is called Successor Representation.

In the SF model, we assume that the space of vectors Φ^π can be reduced in order to trim down recurrent descriptions of the model. In other words, we could embed (s, a) in $\Phi(s, a) \in \mathbb{R}^{d'}$ where $d' << |\mathcal{S}| \times |\mathcal{A}|$. In this case the reward model is an approximated linear relation: $r(s, a) \approx \Phi(s, a) \cdot w$.

In both model, one may remark that this representation isolates environment characteristics from the rewards. State of the environment is described by Φ (which is a function of (s, a)) while rewards rely on w which is constant across the environment.

We can then rewrite the Q-function and use this compact form for the reward function:

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a\Big] \\ &= \mathbb{E}\Big[\sum \gamma^t \Phi_t^T \cdot w | s_0 = s, a_0 = a\Big] \\ &= \mathbb{E}\Big[\sum \gamma^t \Phi_t^T | s_0 = s, a_0 = a\Big] \cdot w \\ &= \Psi^\pi(s, a)^T \cdot w \end{aligned}$$

Where we introduced the successor features Ψ^π. These features are the discounted sum of the state-action representations under the current policy π. If we consider the previous example of Φ being the tabular representation of the state-action space (with a one-hot vector), $\Psi^\pi(s, a)$ is simply the discounted sum of occurrences of (s, a) under π and belongs to $\mathbb{R}^d$.

Recalling our primary goal of transfer learning, this expression of the Q-function has a major advantage : it decouples the dynamics of the MDP under the current policy (captured by Ψ^π) from the task (represented in the weights w). Doing so allows us to learn separately these two components. Once the model learned, a change of task (trough the w vector) could be addressed by re-learning only the task component. We would hope in this case a shorter learning period because the environment dynamics would have been transfered directly. Inversely, if the MDP dynamics change, we could relearn Ψ^π while retaining the informations on the reward (task) contained in w.

To illustrate this purpose, consider again the case of the puddle-world. We assume that we have prior information about the number of the puddle and their possible position. Suppose, such as in Barreto et al. 2016, that we have 2 puddles (one vertical 3×1, one horizontal 1×3) in a 15×15 grid and that their center are in $[3, 5, 7, 9, 11] \times [3, 5, 7, 9, 11]$.

In the case of SR, we consider $\Phi(s, a)$ in $\mathbb{R}^{|S|\times|A|} = \mathbb{R}^{900}$ and as one-hot vector for the couple (s, a). The reward function vector describing the environment is very sparse and takes values only in a small subset : couples (s, a) that lead to a puddle or a goal. In addition to this loss of memory, each update of this method does not take into consideration the prior information of the puddle-world.

Considering the SF model, we can embed a grid as a vector of length 54 : 25 possible positions for each puddle and 4 possible positions for the goal. In doing so, $\Phi(s, a)$ is equal to zero if in any possible configuration, the couple (s, a) leads automatically to a cell which is neither the goal or a puddle. By doing so, we reduce the dimension of Ψ^π without increasing error.

2.3 Learning

Learning the parameters is done by successively alternating between the two components of SFs:

- **Learning the reward w:** Considering the approximated relation $r(s, a) \approx \Phi(s, a) \cdot w$, learning w is a supervised learning problem which can be performed with any of the many algorithms developed for this question (e.g. gradient descent).
- **Learning the successor features Ψ:** we note that Ψ follows a similar Bellman equation as Q:

  $$\Psi^\pi(s, a) = \Phi(s_{t+1} = s', a_{t+1} = a') + \gamma \mathbb{E}\big[\Psi^\pi(s_{t+1} = s', a_{t+1}a')|s_t = s, a_t = a\big]$$

  Then, any algorithm for dynamic programming such as Q-learning could be used to learned the successor features.

Alternating these two learning methods, we can compute an optimal policy for every grid with and a better initial policy for each new grid.

3 Learning architectures for Deep Successor Features

Reducing the state representations to a smaller dimension could be very interesting in many setups where this space is too large and the computation of both components in SF learning is intractable. In the Successor features framework presented above, Barreto et al. 2016 suggests that this could be done easily with an approximation of the state-space representation. As an example, think of a navigation task where the state space are images perceived successively by the agent. In such setups, approximation methods with convolutional neural networks (CNN) are popular. Here, we will present the method developed in Kulkarni et al. 2016 and augmented in Zhang et al. 2017 for adapting Successor features to large state-spaces.

We will detail here the implementation in Zhang et al. 2017, which builds on that of Kulkarni et al. 2016 and seemed more natural to us in terms of its formulation and application.

In their experiment, a robot navigates in maze-like environment where the target is a green sphere lying somewhere in the maze among cubic objects. The state space is the set of images (x_t) received by the robot over a history H: $s_t = (x_{t-H}, ..x_{t-1}, x_t)$.

They first extract relevant features $\Phi^k_{s_t}$ from s_t with a CNN θ_Φ and make sure that they are representative of the state with a decoder θ_d. Then, they perform successor features as described in 2: that means that they are learning separately the reward vector w and the successor features $\Psi_(a_n)$ for $a_n \in \mathcal{A}$, the action space. These different components are learned as follows:

- The successor features are learned with deep Q-learning with a two fully connected network θ_Ψ on the cost function derived from the Bellman equation:

$$L(\theta_\Psi) = \mathbb{E}\Big[(\Phi_s + \gamma\Phi_{s'}(\theta, a^*; \theta^-_\Psi) - \Phi_s(\theta, a; \theta_\Psi))^2\Big]$$

  where $a^* = argmax_{a'}Q(s', a'; \pi^*)$ is obtained with the previous value of Ψ and the reward vector w.

- The reward vector w is learned thanks to a linear regression of the observed reward R_{s_t} over Φ_{s_t}, the extracted features of the space.

- The approximations of the states are learned trough the encoder-decoder (θ_Φ, θ_d). These two components (Φ_{s_t}, w) are learned together with respect to the following objective function which ensures that the reward is regressed

and the features Φ are representative of the state s_t:

$$\underset{(s,R(s))\in\mathcal{D}_R}{\mathbf{L}(\theta_\Phi, \theta_\mathrm{d}, \mathrm{w})} = \mathbb{E}\Big[\big(R(s) - \Phi_s^T w\big)^2 + \big(s - d(\Phi_s; \theta_d)\big)^2\Big]$$

The learning of the total set of parameter $(\Phi_{s_t}, \Psi(\Phi_s, a), w)$ is obtained by alternatively optimizing these two loss functions.

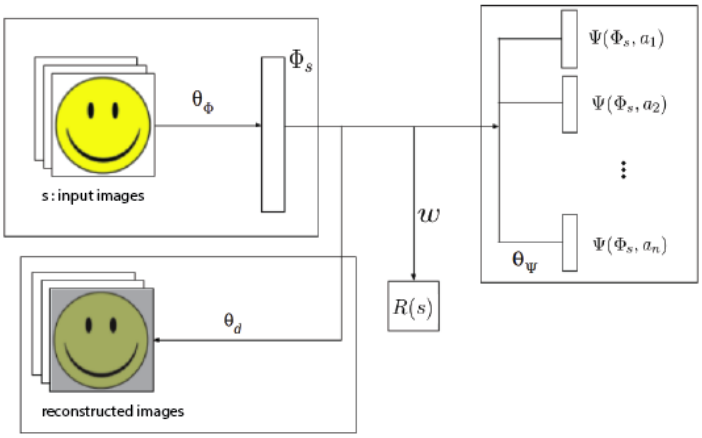

Figure 2: Zhang et al. 2017 architecture for Deep Successor Features.

Zhang et al. 2017 distinguishes two types of transfer in reinforcement learning (TRL):

- **TRL1:** The K different tasks to solve share the same environment and differ only in the reward function. In the navigation framework, it corresponds to a K-objective goals search in a single maze. It is sufficient to relearn only the w vector. It is the same experiment as the one in Lehnert, Tellex, and Littman 2017 and Kulkarni et al. 2016.

- **TRL2:** The more general question of transfering knowdlege between tasks in different environments is also discussed. They assume some linear mapping between each task features, embodying the idea of a shared structure between the various environments. Considering one task k that we already have learned,

$$\Phi_s^i = \mathcal{B}^i \Phi_s^k \quad \forall i \leq k \;\; \text{with} \;\; \mathcal{B}^k = \mathbf{Id}$$

We can then rewrite the Q function of the task i making use of the previously learned task k:

$$Q^i(s, a, \pi^i) = \Psi^{\pi^i}(\mathcal{B}^i \Phi_{s_t}^k; a)^T w^i$$

They take as an example that the extracted features $\Phi(.)$ could be the relative distance of a set of objects to the agent. In this case, the successor features $\Psi(.)$ would be the discounted sum of these relative distances given the actual policy. A transfer in a new environment would result in the adaptation of Φ. But, since we suppose a shared structure between environments, the first layers of the CNN resulting in Φ could be transfered to the new features. This common base could be capture by the linear mapping $\mathcal{B}$. Then, Φ should adapt quickly. And w could be re-learned if we change the rewards-states association or transfered directly if no changes in reward is assumed (in the previous example of relative distance, we could penalize the proximity to a specific set of objects in both environments).

4 Results

Both papers experiments with maze-based agents, which goal is to reach a point in the maze, with negative rewards when stepping on penalty states. We will now briefly describe the setup and the results in each paper, before presenting our implementation in a a puddle-word problem, inspired by Barreto et al. 2016.

4.1 Experiments and results in Zhang et al. 2017 and Kulkarni et al. 2016

4.1.1 Kulkarni

Experiments : They experiment in MazeBase (a gridworld environment). The environment is presented as a sequence of sentences to the agent (e.g. block at [-1;3]). The agent have to reach a goal state located randomly on the map.

Results : They compare their deep successor representation (as described in 3) to a Deep Q-Network baseline (from Mnih et al. 2015) for solving the game. When changing the reward scalar value (from 1.0 to 3.0) the DSR adapts quickly and outperforms the DQN.
They are also able to detect subgoals (or key states) that structure the environment as well as to partition the environment. For example in the MazeBase setup, the agent identifies states as key moments when it is located near a wall that separates the map into two parts. They intend to develop such interesting approach in a hierarchical framework.

4.1.2 Zhang

Experiments : Zhang et al. 2017 experiment in maze-like environments where the target is a green sphere, which lies somewhere in the maze among cubic objects. The agent has 4 available actions (move forward, stand still, rotate by 90^{o} right and left).

They take as baseline methods a CNN trained with supervised learning to predict the actions computed by a A^* planner (optimal path) and a DQN (see Mnih et al. 2015). They train the agent in a base maze (*Map1*) from scratch and then transfer it to different mazes to evaluate the performance of SF-RL compared to the baselines.

They also experiment in a real world-setup, constructing a maze and training the agent on simulated kinect depth images before transferring it to the real maze.

Results : In both simulation transfers and real-world experiments, the SF-RL showed a speed up compared to the the DQN and achieved the same magnitude of precision as the A^* planner. Note that, thanks to the mapping $\mathcal{B}$ between learned representations of the different environments, the SF-RL transfers well from one maze to another and keeps in memory the previous maze that it learned.

Moreover the learned features Φ_s^k contains the localization of the robot in the maze : It is possible to train a supervised model on the positions of the bot to reconstruct the raw position from these features. The agent is able to locate itself.

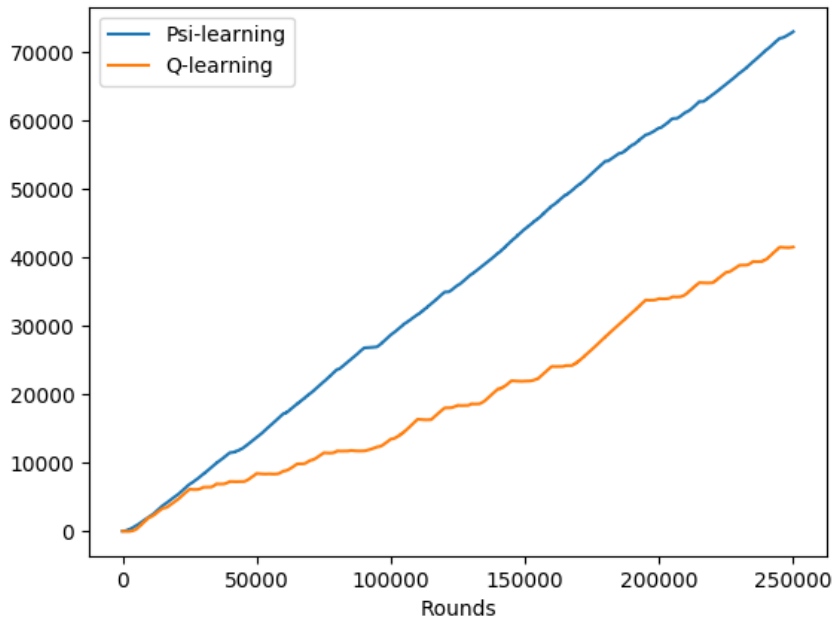

Figure 3: Cumulative rewards for both methods. We change the task (i.e. new round) every 2000 rounds.

4.2 Our results in the puddle world implementation

We implemented the successor features presented in Barreto et al. 2016 with the puddle world setup. The environment is a 15×15 grid with two different types of terminal states: the two 3x1 cells puddles (reward -1), and the goal cell (reward 1). The centers of the puddles are located in a subset of the grid defined by the Cartesian product [3,5,7,9,11][3,5,7,9,11]. The goal cell is at one corner of the grid. We experiment the ability of SF to adapt to different tasks by transferring the agent between various environments, i.e. environments with different locations for the puddles and the goal.

We compared the SF method to a Q-learning baseline with ϵ-greedy policy ($\epsilon = 0.15$). SF method takes more time to learn the optimal policy in the first tasks but after some changes of task, when transferred in a new environment it outperforms clearly the Q-learning. We see this transferring capability in the reward plot (Fig. 3), where the SF method does not present the horizontal levels at each new task.

References

Barreto, André et al. (2016). "Successor Features for Transfer in Reinforcement Learning". In: *arXiv*, pp. 1–13. arXiv: arXiv:1606.05312v1.

Dayan, Peter (1993). "Improving generalization for temporal difference learning: The successor representation". In: *Neural Computation* 5.4, pp. 613–624. ISSN: 0899-7667. DOI: 10.1162/neco.1993.5.4.613. URL: http://discovery.ucl.ac.uk/85073/.

Kulkarni, Tejas D et al. (2016). "Deep Successor Reinforcement Learning". In: *arXiv*. arXiv: arXiv:1606.02396v1.

Lehnert, Lucas, Stefanie Tellex, and Michael L. Littman (2017). "Advantages and Limitations of using Successor Features for Transfer in Reinforcement Learning". In: *arXiv*. arXiv: 1708.00102. URL: http://arxiv.org/abs/1708.00102.

Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, pp. 529–533. ISSN: 0028-0836. DOI: 10.1038/nature14236. URL: http://dx.doi.org/10.1038/nature14236.

Sutton, Richard S (1996). "Generalization in reinforcement learning: Successful examples using sparse coarse coding". In: *Advances in neural information processing systems*, pp. 1038–1044.

Zhang, Jingwei et al. (2017). "Deep Reinforcement Learning with Successor Features for Navigation across Similar Environments". In: *arXiv*. arXiv: 1612.05533. URL: http://arxiv.org/abs/1612.05533.