A Case for Representation-Based Successor Features for Transfer in Reinforcement Learning

Louis Bagot[1 [0000−0001−6300−6993]], Kevin Mets[1 [0000−0002−4812−4841]], Tom De Schepper[1 [0000−0002−2969−3133]], and Steven Latré[1 [0000−0002−8267−9955]]

University of Antwerp - imec;
IDLab - Department of Computer Science;
Sint-Pietersvliet 7, 2000 Antwerp, Belgium
{firstname.lastname}@uantwerpen.be

Abstract. Successor Features stand at the boundary between model-free and model-based Reinforcement Learning. By predicting a sum of features instead of a sum of rewards, they enable very efficient transfer learning through the General Policy Improvement Theorem. Recent work has shifted the focus of the feature space from learnt features to a well-chosen set of base rewards. While this framework greatly improves stability, it discards the flexibility to generalize outside the base reward space. In this paper, we aim to rekindle interest in "representation-based" Successor Features for transfer learning, by clarifying the possible design choices and providing simple cases where they prevail. In a robot arm scenario, we find that they more easily transfer to unseen tasks without suffering from instabilities during training. We provide visual interpretation of the learnt features to explain this performance.

Keywords: Reinforcement Learning · Successor Features · Transfer Learning · Auxiliary Tasks.

1 Introduction

In Reinforcement Learning (RL, Sutton and Barto 2018), an *agent* interacts with an *environment*, performing an *action* to gather *rewards* in a sequential task, using *states* as input. Transfer Learning then refers to the ability to extract and re-use knowledge from previously learnt tasks for new environments.

RL methods are often divided into model-free and model-based RL. *Model-free* RL refers to a direct mapping of states-action to values or probabilities, predicting either a sum of rewards or a policy. This "reflexive" behavior is contrasted by *model-based* RL, which attempts to model the environment, and therefore learn and plan mainly in said model.

The Successor Representation (GPI, Dayan 1993; Kulkarni et al. 2016), later generalized to Successor Features (SFs), is sometimes seen as a middle-ground between model-based and model-free RL (Lehnert and Littman 2020). Indeed, instead of learning a sum of rewards or a model, SFs attempt to predict a sum of *features*, leading to a form of generalized Policy Evaluation over all the reward

functions that can be modelled by the feature space. From there, very efficient transfer learning can be achieved through the Generalized Policy Improvement Theorem (Andre Barreto, Dabney, et al. 2017) over a set of skills. Follow-up work in the field (Andre Barreto, Borsa, et al. 2018) has shifted the focus from learning a set of features to choosing a good set of *base rewards* to act as features, thus explicitly executing Policy Evaluation over this reward basis. This choice leads to a greater stability of the features, and therefore an easier prediction of their sum. On the other hand, its transfer potential relies entirely on the ability of the reward basis to cover as many downstream tasks as possible.

In this paper, we make a case for "representation-based" Successor Features, where the set of features to predict is learnt through Representation Learning methods -here, auxiliary tasks- rather than a fixed reward basis ("reward-based" SFs). To our best knowledge, the GPI has never been applied with general-purpose feature-extraction methods before, nor have representation-based SFs and reward-based SFs been identified and compared. Through an overview of the theory and literature, we motivate certain design choices for the architecture and learning of representation-based SFs, that are later justified through experiments. We empirically show that transfer to unseen tasks of different natures is possible with the more general features learnt by representation-based SFs; tasks on which reward-based SFs fail to transfer. Finally, we provide visual intuition for our results - indeed, we can interpret the learnt features and weights, and understand how the inclusion of auxiliary tasks allows for efficient transfer.

We start by providing a study-like approach to Successor Features, with the necessary background in Section 2, followed by an overview of the literature with respect to several key design choices in SFs in Section 3. We then introduce the general framework for "representation-based" SFs in Section 4 and a comparison of representation-based and reward-based approaches in a simple robot arm scenario in Section 5, with visual interpretations of the learnt features.

2 Background

2.1 Reinforcement Learning and Transfer

The RL interactions are generally formalized with a *Markov Decision Process* (MDP) $M = (\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$, respectively containing the state space, action space, reward space, dynamics function and discount factor. Of particular interest here, the dynamics function $p(s', r \mid s, a)$ dictates the next state and reward, and therefore generates the *reward function* $r(s, a) = \sum_{s',r} p(s', r \mid s, a)\, r$, which can also be expressed as $r(s, a, s')$.

In RL, the notion of Transfer Learning (TL) can have several interpretations. The formalization we use (Z. Zhu, Lin, and Zhou 2020) is to consider a learning in several phases. In an initial phase, the agent learns from a first set of MDPs. In subsequent phases, the agent applies the learnt knowledge or minimally relearns in new MDP sets. What we are interested in SFs is mainly the transfer in the reward function, i.e. the MDPs will mainly differ in the reward space $\mathcal{R}$ and the reward dynamics $r(s, a, s')$.

2.2 Successor Features & General Policy Improvement

Transition Features and Task Vector The Successor Representation was originally introduced by Dayan 1993 in a founding work, and first applied in Deep Learning by Kulkarni et al. 2016. However, arguably the cornerstone paper for us is in the generalization to Successor Features (SFs) and the introduction of the General Policy Improvement Theorem (GPI) by Andre Barreto, Dabney, et al. 2017. All background presented in this section was introduced in this work, unless mentioned otherwise. Successor Features start from the main assumption

$$\phi\left(s, a, s'\right)^{\top} w = r\left(s, a, s'\right) \tag{1}$$

with $\phi\left(s, a, s'\right)$ and $w \in \mathbb{R}^d$. The vector $\phi\left(s, a, s'\right)$ represents features of that transition; in particular, control-oriented features if possible, i.e. that contain information that will be relevant for downstream tasks. We refer to ϕ as the *transition features* and d as the *feature dimension*.

w indicates which features ϕ_i are desirable or not for our particular task. It therefore contains task information: ϕ is task-independent, simply describing the elements of the transition, while w provides a notion of preference in ϕ. We therefore call w the *task vector*, and write it $w^{(j)}$ when associated with an MDP $M^{(j)}$ and reward function $r^{(j)}$.

Successor Features From Equation (1), Andre Barreto, Dabney, et al. 2017 rewrite the definition for the *action-value function* Q:

$$\begin{aligned} Q_\pi\left(s, a\right) &\doteq \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right] \\ &\overset{(1)}{=} \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k \phi\left(S_{t+k}, A_{t+k}, S_{t+k+1}\right) \mid s, a\right]^{\top} w \\ &\doteq \Psi_\pi\left(s, a\right)^{\top} w \end{aligned} \tag{2}$$

This naturally leads to the definition of the *Successor Features* Ψ_π, representing the expected sum of transition features ϕ observed when following π. A very interesting property of Successor Features is that they behave like a value function, and can therefore be learnt efficiently using TD learning:

$$\begin{aligned} \Psi_\pi\left(s, a\right) &\doteq \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k \phi_{t+k} \mid S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi\left[\phi_t + \gamma \Psi_\pi\left(S_{t+1}, A_{t+1}\right) \mid S_t = s, A_t = a\right] \end{aligned} \tag{3}$$

General Policy Improvement Note that Ψ_π is task-independent, it only describes the interactions of π and p in terms of ϕ. More formally, if we focus on a given task $M^{(j)}$ with reward function $r^{(j)}$, we obtain $Q_\pi^{(j)}\left(s, a\right) = \Psi_\pi\left(s, a\right)^{\top} w^{(j)}$.

This means that, for any task $M^{(j)}$, we can directly compute the Q-value of our policy π, $Q_\pi^{(j)}$, as long as we know $w^{(j)}$. This, in turn, means that we can improve on π for a new task by applying the Policy Improvement Theorem. Taking this further still, if we assume that we have a set of D policies $\{\pi_j\}_j$ that we call a *skill library*, then the General Policy Improvement Theorem (GPI, Andre Barreto, Dabney, et al. 2017; Andre Barreto, Borsa, et al. 2018) allows us to compute the strongest policy available for a new task M' by using

$$\begin{aligned} \pi' &= \arg\max_{a,j} Q_{\pi_j}^{M'}(s,a) \geq \pi_j \, \forall j \\ \pi' &= \arg\max_{a,j} \Psi_{\pi_j}(s,a)^\top w' \end{aligned} \tag{4}$$

In other words, when combining the GPI and SFs, using a skill library we can directly compute the strongest policy available for any new task M' as long as we can fit the linear regression $\phi^\top w' \approx r'$, with some bounds on the uncertainty.

3 Design and Algorithmic Choices in SFs

3.1 Representation- and Reward-based Transition Features

There are essentially two crucial design questions regarding GPI-based Transfer using SFs: (i) the skill library, and (ii) the choice of ϕ. In this paper, we will focus on the second issue. The choice of ϕ inherently controls our generalization power for transfer. Indeed, the objective is to design a ϕ so that Equation 1 holds as closely as possible for as many downstream tasks as possible. Recently, the literature has shifted into two main branches regarding this choice, that we refer to as

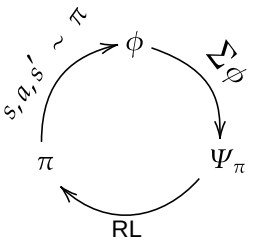

Fig. 1. Instabilities in the SF learning loop: ϕ is estimated by Ψ_π (leading to Q_π), which modifies π through RL, generating new data modifying ϕ

- **Representation-based** ϕ (e.g. Kulkarni et al. 2016): the transition features are learnt through some Representation Learning method, which attempts to include as many potentially interesting features as possible. While this holds great potential for transfer, the resulting learning loop is a form of non-stationary reward setting, as can be visualized on the graph on the right. This was observed by Andre Barreto, Borsa, et al. 2018 and led to the second branch:
- **Reward-based** ϕ: since we start from a skill library to use the GPI, we can instead assume a *reward library* $\{r_1 \cdots r_D\}$ generating these skills. The proposal is then to join the two critical design questions into the single $\phi = [r_1 \cdots r_D]$. Conceptually, according to Equation 1 this means that all downstream rewards will have to be expressed as a weighted sum of our reward library, which can then also be called *reward basis*. The main

advantages of this method is that it gets rid of the stability issues, since the reward library can be assumed stationary (therefore breaking the $\pi \rightarrow \phi$ link); and it solves the dilemma of the ϕ design.

Following the work by Andre Barreto, Borsa, et al. 2018, we observe a switch in paradigm from representation-based to reward-based SFs (Borsa et al. 2018; André Barreto, Hou, et al. 2020; André Barreto, Borsa, et al. 2019; Machado, Andre Barreto, and Precup 2021). While reward-based SFs may seem attractive, we introduce a simple thought experiment to showcase their main weakness: imagine an environment populated with colored boxes (red, green, blue). The initial tasks are to reach the red and green boxes, and the transfer task is to reach the blue box. If our reward library consists of the initial tasks $\{r_G, r_R\}$, a weighted sum of these can never successfully model the blue box reward r_B. In comparison, with a simple auto-encoder auxiliary task, a representation-based SF scenario will naturally contain blue box information in its representation, and can therefore hope to perform much better than reward-based SFs. In essence, the reward basis can be limiting in ways that a learnt representation can easily cover for, even without expert knowledge. In other words, reward-based SFs put huge pressure on the reward library to cover the skill *and* reward space.

Therefore, in this paper, we argue that representation-based SFs hold sufficiently strong arguments in their transfer potential to make up for their instabilities. In the rest of this section, we describe algorithmic differences sprouting from choices of the ϕ conditioning.

3.2 On the Transition Feature conditioning

The most general form of Equation 1 conditions ϕ on the transition (s, a, s'), however it is generally inefficient and not necessary to take in so much information. In practice, several combinations of s, s' and a are viable and lead to different assumptions and algorithmic ideas. We delve into the most popular choices found in the literature and argue for the one we find most relevant.

State-conditioned $\phi(s)$ Given how closely the choice of ϕ ties with Representation Learning, the most natural and popular choice for conditioning is to simply let ϕ be features of the state, $\phi = \phi(s)$, and let w isolate desirable features of the state (Kulkarni et al. 2016; Zhang et al. 2017; Machado, Rosenbaum, et al. 2018; Hansen et al. 2020). However, this choice means that Equation 1 will only hold if ϕ implicitly contains action information; in other words, $\phi(s)$ needs to be policy-dependent: $\phi = \phi_\pi(s)$. This is a major issue for applying the GPI, since every single policy in the library will need to learn its own ϕ and associated w. This complicates the setup, increases training time, reduces interpretability of the learnt tasks and nullifies potential for downstream generalized Meta-training (by inputting w, André Barreto, Borsa, et al. 2019; Machado, Andre Barreto, and Precup 2021). The arduousness of the process can be observed in Zhang et al. 2017, where new ϕ_π need to be adapted for every single task tackled. Note

that several of the cited papers ignore the policy-dependence entirely, by adopting an environment where the choice makes little difference; however this is not generalizable to any MDP.

Next-state conditioned $\phi(s')$ To encapsulate policy-independent reward information, we can instead turn to next-state conditioning (Ma, Wen, and Bengio 2018). The difference is subtle and might not always be clear in the literature, as the Ψ training can be identical to state-conditioned features, but the reward prediction approximation differs: $\phi(s')^\top w \approx r(s, a, s')$. While this choice lifts the policy-dependence and solves the issues mentioned above, in the general case it is not possible to infer rewards from the next state uniquely. As a simple example, in a gridworld with two states leading to a terminal cell with rewards $+1$ and -1, the terminal state $s' = s_T$ alone doesn't contain enough information to tell the rewards apart.

State-action conditioned $\phi(s, a)$ This choice is also policy-independent, and only requires ϕ to encapsulate some dynamics information. It is theoretically sound compared to next-state conditioning, while being directly scalable to apply the GPI with a single w per task. We can either take a as input or use a different head per discrete action. In the first case, this slows down inference when we need to compute all actions at the same time. In the latter case, it scales the representation by $|\mathcal{A}|$, and we have no guarantee that the learnt features will be consistent over actions. In practice, we opt for the second choice and find that both issues are minor, and that the theoretical and algorithmic benefits of picking state-action conditioning outweigh its drawbacks. This is therefore our preferred choice and the one of several prior works (Y. Zhu et al. 2017; Andre Barreto, Borsa, et al. 2018).

4 Representation-based Successor Features

In this section, we describe our implementation choices to apply GPI with representation-based SFs.

4.1 Training

Forcing the Bellman Equation Note that in the state-action conditioning scenario, Equation 3 becomes

$$\begin{aligned}\Psi_\pi(s, a) &= \mathbb{E}_\pi\left[\phi(S_t, A_t) + \gamma\Psi_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right] \\ &= \phi(s, a) + \gamma\mathbb{E}_\pi\left[\Psi_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]\end{aligned} \tag{5}$$

In other words, since we have access to $\phi(s, a)$ as part of our architecture, we can focus on directly predicting the next state-action SFs $\Psi_\pi(S_{t+1}, A_{t+1})$, and spare our approximator from predicting the first step and applying the discount factor. This is tantamount to having access to the immediate reward in traditional RL. We refer to this as "forcing" the Bellman equation.

Auxiliary Task Learning Most previous methods of learning ϕ can be seen with the lens of Auxiliary Task Learning (Jaderberg et al. 2017; Gelada et al. 2019); in other words, using some alternative loss in conjunction with the real objective to guide the representation in containing generally useful features. In RL, three auxiliary tasks sprout naturally from available information in the MDP: (i) reward prediction $\hat{r} \approx r(s,a)$, (ii) state reconstruction $\hat{s} \approx s$, and (iii) forward prediction $\hat{s'} \approx s' \sim p(s' \mid s,a)$. Reward prediction is already a core element of the SF learning loop through Equation 1, and can be identified as the "main objective" for ϕ. On top of reward prediction, we use state reconstruction and forward prediction tasks, which can both be implemented easily with state-action conditioned features. Previous work has implemented both, usually independently (reconstruction: Kulkarni et al. 2016; Zhang et al. 2017; forward model: Machado, Rosenbaum, et al. 2018; Machado, Bellemare, and Bowling 2020). We combine both in an auto-encoder-style architecture for our ϕ training and train all three tasks to learn ϕ. We provide more architectural details in the following subsection.

Loss computation Although there might seem to be several moving parts to the SF training of the skill library, it generally winds down the the same major three components regardless of ϕ conditioning, summed up in the following losses:

$$\begin{aligned} \mathcal{L}_r &= \left|\left|\phi^\top w - r(s,a,s')\right|\right| \\ \mathcal{L}_{SF} &= \left|\left|\phi + \gamma \max_{a^*} \Psi_\pi(s',a^*) - \Psi_\pi(s,a)\right|\right| \\ \mathcal{L}_{aux} &= ||\hat{x}(\phi) - x|| \end{aligned} \tag{6}$$

where $\mathcal{L}_r$ simply represents Equation 1, $\mathcal{L}_{SF}$ is a control version of Equation 3 (here, using QLearning), and $\mathcal{L}_{aux}$ represents any chosen auxiliary tasks on prediction target x (e.g. the state or next state, as seen in the previous paragraph). This is the general idea we follow; all our specific choices are mentioned in the experiments, Section 5.2. Previous work sometimes also optimizes w to predict the Q-value through $\mathcal{L}_Q = \left|\left|\Psi(s,a)^\top w - \hat{Q}(s,a)\right|\right|$ (e.g. Y. Zhu et al. 2017; Janz et al. 2019) where $\hat{Q}(s,a)$ is any Q-value target – we do not incorporate this loss, to keep the framework simple by underlining the difference in training between modules. In order to use the GPI, the $\mathcal{L}_r$ and $\mathcal{L}_{SF}$ losses in Equation 6 simply need to be computed for each reward or skill in the library.

The most basic form of SF transfer learning is achieved by computing a new task vector for the reward of the new task: $\phi^\top w' \approx r'(s,a,s')$, and applying the GPI on the skill library. It is also possible to retrain a new policy through any RL means, using the GPI policy as an initial behavior to gain jump-start performance.

4.2 Neural Network Architecture

The literature has seen a very wide variety of architectures depending on feature-conditioning, specific tasks or transfer scenarios – as such, virtually all papers

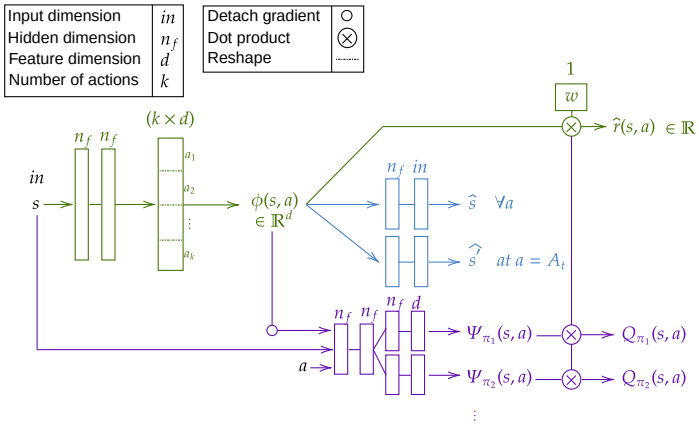

Fig. 2. SF architecture. The ϕ extraction module appears in green, with two auxiliary tasks in blue. The SF network is in purple and displays a shared Ψ body over policies. All layers are Fully Connected, with output dimension above the layer. $\phi(s,a)$ is simply reshaped from a concatenated $k \times d$ layer, leading to action-dependent features.

cited thus far have used a unique architecture. However, the core is generally the same: a Neural Network with heads for ϕ, Ψ and maybe some auxiliary tasks. One of the main design choice in the architecture generally revolves around the parameter sharing between ϕ and Ψ: one might decide to backpropagate both losses through a shared body (Y. Zhu et al. 2017; Andre Barreto, Borsa, et al. 2018) or instead decide to keep ϕ totally independent from Ψ by detaching the gradient (Kulkarni et al. 2016; Machado, Bellemare, and Bowling 2020). In order to isolate the ϕ training and dampen the SF instability loop, we opt for the latter, and study this choice in the experiments in Section 5. The entire architecture we use (ϕ, Ψ and the task and auxiliary task heads w, $\hat{s}$, $\hat{s'}$) can be seen in Figure 2. Note that we share parameters between the different Ψ_{π_j} by using a shared body, but detach their gradient from flowing into ϕ.

An important point to underline from our architecture is that the $\phi(s,a) \in \mathbb{R}^d$ are different vectors for all actions, in a similar fashion as the several action heads $Q(s,a)$ in a regular DQN (Mnih et al. 2015), and as mentioned in Section 3.2. While our choice allows for rapid computation of all action features, we need to be aware that there is no reason the transition features should be similar between actions – they could be encoding information in vastly different ways. However, they all need to pass through the same $w \in \mathbb{R}^d$ to approximate the reward – we can expect that this acts as a form of regularization over the vectors. To further stress on this idea, the $\hat{s}$ and $\hat{s'}$ modules also take as input a d-dimensional vector, which means the $\phi(s,a)$ have yet more implicit motivation to coordinate their representations. We study this idea in Section 5.5.

5 Experiments

5.1 Environment & Tasks

We replicate the Reacher Domain environment used in Andre Barreto, Dabney, et al. 2017, based on the the MuJoCo Reacher environment (Todorov, Erez, and Tassa 2012) and adapted as a discrete-action, goal-reaching transfer task. It simulates a two-jointed robot arm moving in a 2D plane, as represented in Figure 3.

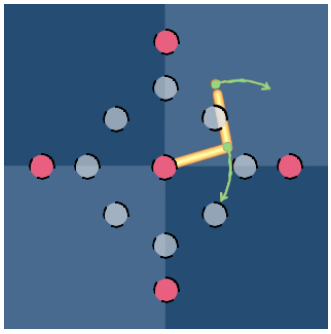

Fig. 3. Reacher Domain adapted from Andre Barreto, Dabney, et al. 2017. Tasks: in red, the 5 training tasks (skill library); in grey, the 8 goal-reaching test tasks; in green, a representation of two speed-maximization tasks.

Following the paper, we use a state $s = \left[\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2\right] \in \mathbb{R}^4$ with θ_i the angle (in radians) of the i^{th} arm starting from the center, and $\dot{\theta}_i$ its angular speed, which we divide by 10 for normalization purposes. The action space is discrete with $|\mathcal{A}| = 9$ encoding all combinations of the $[-1, 0, 1]$ possible torques for both arms. We define two possible tasks: the *goal-reaching* task is to reach a given 2D position g, with reward $r_g(s, a, s') = 1 - ||g - s'||_2$. The *speed-maximizing* task is to maximize or minimize the speed of the arms. We use 8 speed transfer tasks, all combinations of maximizing, ignoring or minimizing $\dot{\theta}_j$ for both arms, except the task to ignore both. The resulting rewards are therefore $r_{speed}(s, a, s') = \sum_{j:1,2} m_j s' \left[\dot{\theta}_j\right]$ with $m_j \in [-1, 0, 1]$. We choose this set of speed-maximization tasks as transfer tasks that do not contain any direct link with goal-reaching tasks, in order to study generalization capacities of the learnt features beyond solved tasks. Differently from Andre Barreto, Dabney, et al. 2017, we do not provide any form of goal as input or part of ϕ, and instead assume the agent only has access to the reward to maximize – this is a much more general scenario that does not make any assumption on the type of transfer we will be tackling. We train our skill library on $D = 5$ goal-reaching tasks, using the goals in red[1] on Figure 3, which define the reward-based transition features $\phi(s, a) = [r_1(s, a), \cdots, r_5(s, a)]$. Our Transfer tasks are 8 additional goals presented in grey, and the 8 mentioned speed maximizing tasks.

5.2 Agent Training

As mentioned above, we use state-action conditioning with $d = 32$ and force the Bellman Equation on a DQN-style algorithm (with target network Ψ_π^-). We use

[1] Note that our training goals differ from the original paper: we found them to be more expressive, able to transfer to more downstream behaviors.

both reconstruction and forward prediction auxiliary tasks in an auto-encoder setting, resulting in the final auxiliary loss

$$\mathcal{L}_{aux} = (1 - \lambda_{fw}) \left|\left|\hat{s} - s\right|\right|_2 + \lambda_{fw} ||\hat{s'} - s'||_2$$

We combine our losses through $\mathcal{L} = \mathcal{L}_r + \lambda_{aux}\mathcal{L}_{aux} + \mathcal{L}_{SF}$; though we should note that the SF branch is completely independent from the ϕ extraction and might as well be thought of as optimizing a different network entirely.

We train the reward library for $100K$ steps and the transfer tasks for $10K$ steps; all other hyperparameters are provided in Appendix A. Each episode of a learning phase (library training, goal transfer, speed transfer) randomly samples an objective from the given phase, and the agent therefore observes its reward within the time limit $T = 100$. To simulate a continual learning scenario from the agent's perspective, we do not include the time-based terminal transitions in the buffer. All quantitative experiments are run over 10 seeds; qualitative visualizations are selected on a random seed.

5.3 Main results

We compare the representation-based SF agent with a reward-based agent, i.e. using $\phi(s, a) = [r_1(s, a), \cdots, r_D(s, a)]$ with $\{r_j\}$ our reward library (red tasks in Figure 3). Our results in training the library and transferring to both goal-reaching and speed-maximization tasks can be found in Figure 4.

Training performance One of the main motivations for switching from representation-based to reward-based SFs was the instability loops (Figure 1) leading to a difficult training of the skill library. While our environment is simple and our objective is not to contradict this, we find that representation-based SFs are just as efficient than reward-based SFs in learning a skill library. As can be seen in Figure 4 (top left), their performances are very similar, showing that learning a single ϕ representation with auxiliary tasks can scale to several skills in a SF setting. Note that in our network architecture (Figure 2, Section 4.2) we decided to detach Ψ to prevent its updates from affecting the transition features ϕ, to dampen the instability loop. We now report the importance of this choice: we attach Ψ to the ϕ network and let its gradient flow through the transition features. In Figure 4 in red, we observe that detaching is crucial, as otherwise the network fails to obtain proper performance on the skill library. We conclude from this that the instability loop can be partly mitigated by making the ϕ network independent from the Ψ network. This matches with intuition on SFs, since ϕ simply extracts relevant transition features, and Ψ predicts a sum of ϕ when following π. Backpropagating the Ψ gradients through ϕ means that we are constraining our transition features to contain policy information, but also to somehow contain information about their own sum.

Transfer to target positions We now turn to goal-reaching transfer learning: a random goal with the 8 provided in grey in Figure 3 is sampled at the start of

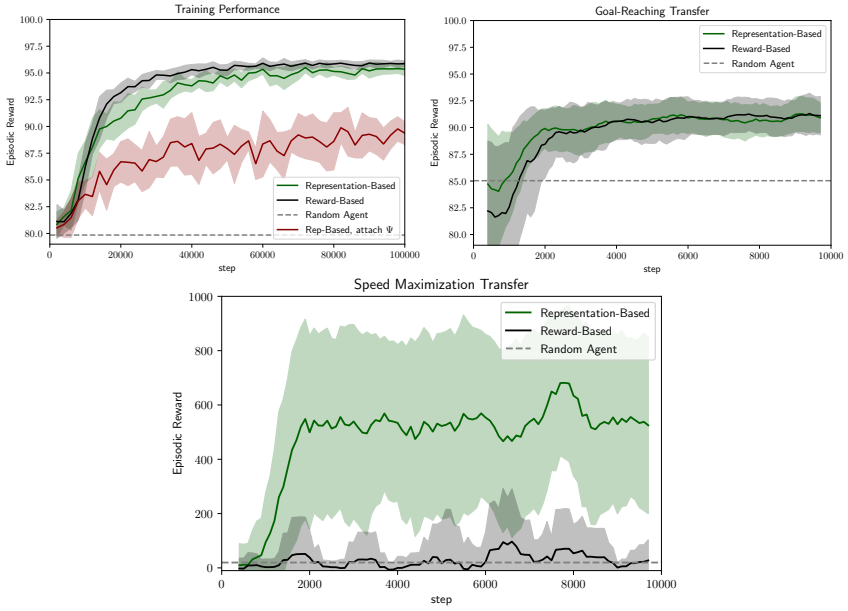

Fig. 4. Comparing training and transfer performances of a feature-based SF representation versus a reward-based one. [top right] training performance; [top left] performance for goal transfer; [bottom] performance for speed transfer(right).

each episode, and the agent needs to maximize the reward[2]. The only transfer information provided is the index of the task. We learn the new task vectors $w^{(j)}$ in both reward-based and representation-based SFs, and apply the GPI. We collect $1K$ samples to fill up the memory buffer before starting learning, and train for $10K$ steps. The results can be observed in Figure 4 (top right). We again observe very little difference between the reward- and representation-based approaches. Indeed, the reward library is sufficient in this scenario to approximate the rewards with good precision. In particular, note that while training performance peaked around 95, transfer performance only peaked around 90: this is due to the relatively weak skill library we used – in general, state-reaching skills struggle to generalize their behavior beyond the training set; however finding stronger skill libraries is beyond the scope of this paper. What is of interest to us here is that both methods can generalize to a new set of goal-reaching tasks when their transition features were trained on goal-reaching tasks. Both methods start learning from $t = 1K$ steps and reach their plateau as soon as $t = 2K$ while learning 8 tasks at the same time. In comparison, the methods reach their plateau in around $20K$ steps (starting to learn from $t = 2K$ steps) when learning the skill library over 5 tasks, which confirms that SF transfer is very efficient.

[2] Note that since the goals are closer to the center, it is natural that the Random Agent performance is stronger.

Transfer to target speeds The main result of our paper is presented on the bottom row of Figure 4, comparing the speed-maximization transfer of reward-based and representation-based SFs. Following our intuition from Section 3.1, reward-based SFs are unable to model the new rewards based on the provided reward library, leading to a failure of the GPI. Even though the skill library seems to be covering for a lot of behaviors, and certainly for these speed maximization tasks, using the reward library as features does not allow sufficient expressiveness to transfer efficiently in this scenario. In comparison, simple auxiliary tasks are sufficient for representation-based SFs to approximate the speed-based rewards and achieve strong performance through GPI. We will now provide more intuition into this difference and the learnt features.

5.4 Visualizations of Learnt Features and Task Vectors

In this section we provide visual intuition for the performance of representation-based SFs. For all of these experiments, we use $d = 8$ to obtain clearer plots; the performance difference between $d = 32$ and $d = 8$ is negligible for our purposes.

It was clear from Figure 4 that a weighted sum of the reward library was not enough to model the speed-maximization reward. This is perfectly intuitive, since this reward basis does not contain any information about speed. What were the transition features learnt by the representation-based method? In order to fully visualize them, we make use of the low input and feature dimensions to display all features for varying values of the state values $s = \left[\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2\right]$ starting from $s_0 = [0, 0, 0, 0]$. The key factor for ϕ to generalize for speed-based rewards is the introduction of the auxiliary tasks. To demonstrate this, we visualize the variations of $\phi(s, \cdot)$, with and without auxiliary tasks on Figure 5 (top row; features averaged over actions). It is very clear that the learning of auxiliary tasks leads to more expressiveness with respect to the arm speeds: without auxiliary tasks, the features are nearly independent of $\dot{\theta}$.

In Figure 5 we display the whole reward-approximation process $\phi^\top w = r$. With auxiliary tasks, we can see that the task vector (middle row) puts a high negative weight on feature ϕ_0 and a high positive weight on feature ϕ_6. As expected, ϕ_0 inversely correlates with both speeds, while ϕ_6 increases with $\dot{\theta}_0$ (top row). The resulting reward prediction (bottom row, blue) is a good approximation of the ground truth (orange). In comparison, without auxiliary tasks, there are no obvious features to favor to model a speed-maximization reward, therefore the prediction fails.

These qualitative results confirm our intuition that by learning auxiliary tasks and general features, representation-based SFs can better generalize to new tasks than reward-based SFs with the same skill library.

5.5 Action-wise Feature Regularization

In Section 4.2 we have decided to use independent heads for each action in the $\phi(s, a)$ transition features. While this saves time during inference, it comes with

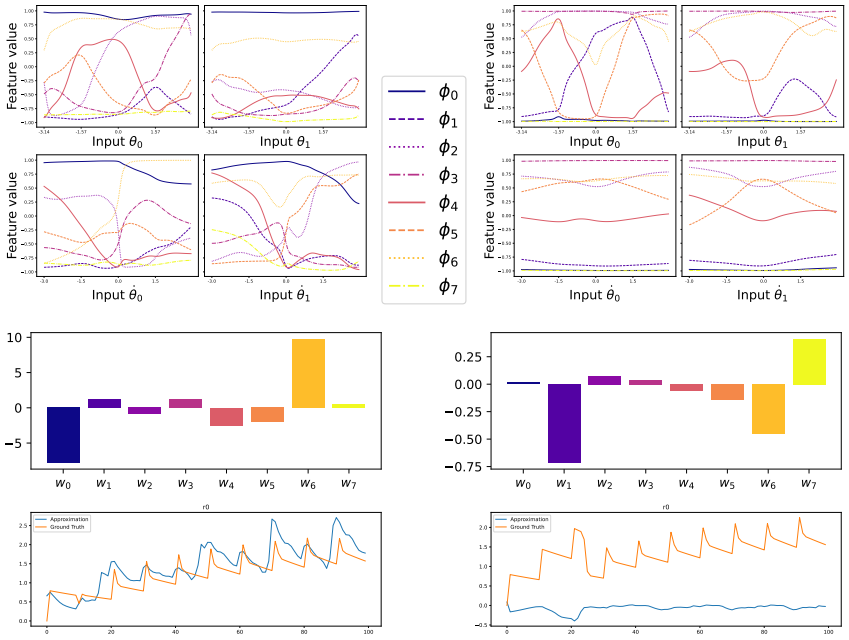

Fig. 5. [top] Variations of the learnt features $\phi(s, \cdot)$ for varying values of the input state. Left and right: with and without auxiliary tasks. The learning of auxiliary tasks adds sensitivity to the velocity of the arms $\dot{\theta}_j$ (bottom rows), which later allows for efficient transfer learning. [middle] Associated task vectors w to maximize both speeds. With auxiliary tasks, the method puts most weight on features $-\phi_0$ and ϕ_6, both of which increase with the velocities. [bottom] Resulting reward prediction $\hat{r} = \phi^\top w$ in blue for a trajectory of the speed-maximizing task (real reward in orange).

the risk that the different heads will encode information differently, and perturb Ψ prediction. We compute the average difference between action features over $10K$ states of random interactions, and plot the resulting distance matrix in the top row of Figure 6. We compare SFs with and without auxiliary tasks.

We observe that the difference in features is slightly higher with auxiliary tasks (maximum of 0.3 versus 0.15). Removing auxiliary tasks means that the features only need to predict the reward library, and goal-reaching rewards are smooth and do not differ much between actions. Therefore, it is natural that adding auxiliary tasks should add flexibility action-wise: different actions lead to quite different outcomes when we include forward prediction. It is interesting to note that the two maximally distant actions in both cases are a_2 and a_6, since they respectively encode torques of $[1, 1]$ and $[-1, -1]$, which are intuitively the most different actions. Given that the features are constrained in $[-1, 1]$ due to the tanh activation, we argue that a maximal difference of 0.3 is quite mild, and evaluate that the learnt features must be coherent between actions. We qualitatively assess this by plotting $\phi(s_0, a)$ at $s_0 = [0, 0, 0, 0]$ (straight arms to

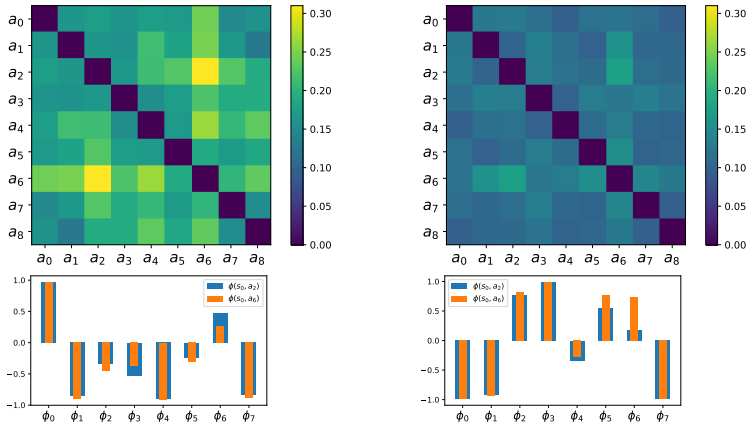

Fig. 6. [top] Difference in $\phi(s,a)$ over $10K$ random states, for (left) representation-based SFs with auxiliary task learning and (right) without auxiliary tasks. A difference in maximum 0.3 of the features shows that the actions are self-consistent: this is verified qualitatively in the [bottom] where we plot $\phi(s_0,a)$ at $s_0 = [0,0,0,0]$ (straight arms to the right with no speed) for the maximally different actions $a = 2$ and $a = 6$. The features are generally close together for both methods.

the right with no speed) for the two maximally different actions, on the bottom row of Figure 6. We observe that, for both methods, the feature vectors for both actions are very close to each other, only differing by small amounts on some features. We therefore conclude that using different action heads for $\phi(s,a)$ is not a practical issue in our case, since the different losses act as regularization to constrain the features to behave similarly.

6 Conclusion

We have identified "reward-based" and "representation-based" features as the two main branches of a critical design choice in Successor Features: the transition features ϕ. We have motivated the usage of $\phi(s,a)$ over other conditioning choices, and discussed the architecture and training template for representation-based Successor Features. In a robot arm environment, we have studied transfer learning to similar tasks (goal-reaching to goal-reaching transfer without goal information) and to completely different tasks (speed maximization). We have argued and empirically shown that, although representation-based SFs are prone to instability issues, they can much more easily transfer to the new task family. This is achieved through the crucial inclusion of standard auxiliary tasks to augment the transition features with general information. While auxiliary tasks have been used in SFs before, to the best of our knowledge this is the first time they are identified as a crucial generalization tool for applying the GPI in Successor Feature scenarios. This is also the first time representation- and reward-based

SFs have been defined and compared. Finally, we have provided visual intuition for the transfer performance of representation-based SFs, displaying their interpretability.

Acknowledgements This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificile Intelligentie (AI) Vlaanderen" programme.

References

[Bar+17] Andre Barreto, Will Dabney, et al. "Successor Features for Transfer in Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[Bar+18] Andre Barreto, Diana Borsa, et al. "Transfer in deep reinforcement learning using successor features and generalised policy improvement". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 501–510.

[Bar+19] André Barreto, Diana Borsa, et al. "The option keyboard: Combining skills in reinforcement learning". In: *Advances in Neural Information Processing Systems* 32 (2019).

[Bar+20] André Barreto, Shaobo Hou, et al. "Fast reinforcement learning with generalized policy updates". In: *Proceedings of the National Academy of Sciences* 117.48 (2020), pp. 30079–30087.

[Bor+18] Diana Borsa et al. "Universal successor features approximators". In: *arXiv preprint arXiv:1812.07626* (2018).

[Day93] Peter Dayan. "Improving generalization for temporal difference learning: The successor representation". In: *Neural computation* 5.4 (1993), pp. 613–624.

[Gel+19] Carles Gelada et al. "Deepmdp: Learning continuous latent space models for representation learning". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2170–2179.

[Han+20] Steven Hansen et al. "Fast Task Inference with Variational Intrinsic Successor Features". In: *International Conference on Learning Representations*. 2020.

[Jad+17] Max Jaderberg et al. "Reinforcement Learning with Unsupervised Auxiliary Tasks". In: *International Conference on Learning Representations*. 2017.

[Jan+19] David Janz et al. "Successor uncertainties: exploration and uncertainty in temporal difference learning". In: *Advances in Neural Information Processing Systems* 32 (2019).

[Kul+16] Tejas D Kulkarni et al. "Deep successor reinforcement learning". In: *arXiv preprint arXiv:1606.02396* (2016).

[LL20] Lucas Lehnert and Michael L Littman. "Successor Features Combine Elements of Model-Free and Model-based Reinforcement Learning." In: *J. Mach. Learn. Res.* 21 (2020), pp. 196–1.

[Mac+18] Marlos C Machado, Clemens Rosenbaum, et al. "Eigenoption Discovery through the Deep Successor Representation". In: *International Conference on Learning Representations*. 2018.

[MBB20] Marlos C Machado, Marc G Bellemare, and Michael Bowling. "Count-based exploration with the successor representation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 5125–5133.

[MBP21] Marlos C Machado, Andre Barreto, and Doina Precup. "Temporal abstraction in reinforcement learning with the successor representation". In: *arXiv preprint arXiv:2110.05740* (2021).

[Mni+15] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[MWB18] Chen Ma, Junfeng Wen, and Yoshua Bengio. *Universal Successor Representations for Transfer Reinforcement Learning*. 2018.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. 2018.

[TET12] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033.

[Zha+17] Jingwei Zhang et al. "Deep reinforcement learning with successor features for navigation across similar environments". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 2371–2378.

[Zhu+17] Yuke Zhu et al. "Visual semantic planning using deep successor representations". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 483–492.

[ZLZ20] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. "Transfer Learning in Deep Reinforcement Learning: A Survey". In: *CoRR* abs/2009.07888 (2020).

A Hyperparameters

Hyperparameter	Value
Discount factor γ	0.9
Feature dimension d	32
Training starts – skill library/transfer	2K / 1K steps
Target network update period	1K steps
Adam learning rate	0.0005
Activation function	tanh
Exploration parameter ϵ	0.1
Auxiliary loss parameter λ_{aux}	0.005
Forward/reconstruction balance λ_{fw}	0.5

Table 1. List of hyperparameters used