Deep Reinforcement Learning with Successor Features for Navigation across Similar Environments

Jingwei Zhang Jost Tobias Springenberg Joschka Boedecker Wolfram Burgard

***Abstract*— In this paper we consider the problem of robot navigation in simple maze-like environments where the robot has to rely on its onboard sensors to perform the navigation task. In particular, we are interested in solutions to this problem that do not require localization, mapping or planning. Additionally, we require that our solution can quickly adapt to new situations (e.g., changing navigation goals and environments). To meet these criteria we frame this problem as a sequence of related reinforcement learning tasks. We propose a successor-feature-based deep reinforcement learning algorithm that can learn to transfer knowledge from previously mastered navigation tasks to new problem instances. Our algorithm substantially decreases the required learning time after the first task instance has been solved, which makes it easily adaptable to changing environments. We validate our method in both simulated and real robot experiments with a Robotino and compare it to a set of baseline methods including classical planning-based navigation.**

I. Introduction

Autonomous navigation is one of the core problems in mobile robotics. It can roughly be characterized as the ability of a robot to get from its current position to a designated goal location solely based on the input it receives from its on-board sensors. A popular approach to this problem relies on the successful combination of a series of different algorithms for the problems of simultaneous localization and mapping (SLAM), localization in a given map as well as path planning and control, all of which often depend on additional information given to the agent. Although individually the problems of SLAM, localization, path planning and control are well understood [1], [2], [3], and a lot of progress has been made on learning control [4], they have mainly been treated as separable problems within robotics and some often require human assistance during setup-time. For example, the majority of SLAM solutions are implemented as passive procedures relying on special exploration strategies or a human controlling the robot for sensory data acquisition. In addition, they typically require an expert to check as to whether the obtained map is accurate enough for path planning and localization.

Our goal in this paper is to make first steps towards a solution for navigation tasks without explicit localization, mapping and path planning procedures. To achieve this we adopt a reinforcement learning (RL) perspective, building on recent successes of deep RL algorithms for solving challenging control tasks [5], [6], [7], [8], [9]. For such an RL algorithm to be useful for robot navigation we desire that it can quickly adapt to new situations (e.g., changing navigation goals and environments) while still preserving the solutions to earlier problems: a prerequisite that is not fulfilled by current state-of-the-art RL-based methods. To achieve this, we employ *successor representation* learning, which has recently also been combined with deep nets [7], [10]. As we show in this paper, this formulation can be extended to handle sequential task transfer naturally, with minimal additional computational costs; its ability of retaining a compact representation of the Q functions of all encountered tasks enables it to cope with the limited memory and processing capabilities on robotic platforms.

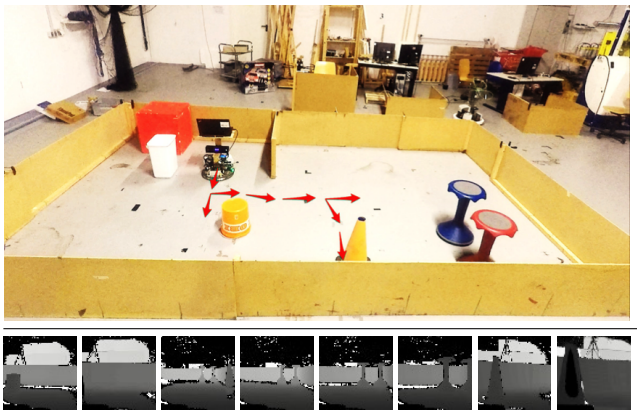


Fig. 1: Exemplary maze-like environment considered in this paper (*Map6*) and the optimal path from a randomly chosen start position to the goal (orange traffic cone) taken by the Robotino robot (top) together with the sensory input captured by the robot's on-board kinect sensor (bottom).

We validate our approach and its fast transfer learning capabilities in both simulated and real world experiments, on both visual and depth inputs, where the agent must navigate different maze-like environments. We compare it to several baselines such as a conventional planner (assuming perfect localization), a supervised imitation learner (assuming perfect localization during training only), and transfer with DQN. In addition, we validate that deep convolutional neural networks (CNNs) can be used to imitate conventional planners in our considered domain.

II. Relations To Existing Work

Our work is related to an increasing amount of literature on deep reinforcement learning. We here highlight the most apparent connections to recent trends with a focus on value

All authors are with the University of Freiburg, Institute of Computer Science, 79110 Freiburg, Germany. This work was partly funded through the State Graduate Funding Program of Baden-Württemberg and by the DFG grant SPP-1597.

{zhang,springj,jboedeck,burgard}@cs.uni-freiburg.de

based RL (which we use as a basis). A more detailed review of the concepts we built upon is then given in Sec. III.

As mentioned, a growing amount of success has been reported for value-based RL in combination with deep neural networks. This idea was arguably popularized by the Deep Q-networks (DQN) [5] approach followed by a large body of work deriving extended variants (e.g., recent adaptations to continuous control [6], [9] and improvements stabilizing its performance [11], [12], [13]).

While the DQN inspired RL algorithms were shown to be surprisingly effective, they also come with some caveats that complicate transfer to novel tasks (one of the key attributes we are interested in). More precisely, although a neural network trained using Q-learning on a specific task is expected to learn features that are informative about both: i) the dynamics induced by the policy of the agent in a given environment (we refer to this as the *policy dynamics* in the following text), and ii) the association of rewards to states; these two sources of information cannot be assumed to be clearly separated within the network. As a consequence, while fine-tuning a learned Q-network on a related task might be possible, it is not immediately clear how the aforementioned learned knowledge could be transferred in a way that keeps the policy on the original task intact. One attempt at clearly separating reward attribution for different tasks while learning a shared representation is the idea of learning a general (or universal) value function [14] over many (sub)-tasks that has recently also been combined with DQN-type methods [15]. Our method can be interpreted as a special parametrization of a general value function architecture that facilitates fast task transfer.

Task transfer is one of the long standing problems in RL. Historically, most existing work in this direction relied on simple task models and explicitly known relations between tasks or known dynamics [16], [17], [18]. More recently, there have been several attempts at combining task transfer with Deep RL [19], [20], [21], [22], [23], [24]. E.g., Parisotto *et al.* [19] and Rusu *et al.* [20] performed multi-task learning (transferring useful features between different ATARI games) by fine-tuning a DQN network (trained on a single ATARI game) on multiple "related" games. More directly related to our work, Rusu *et al.* [21] developed the *Progressive Networks* approach which trains an RL agent to progressively solve a set of tasks, allowing it to re-use the feature representation learned on tasks it has already mastered. Their derivation has the advantage that performance on all considered tasks is preserved but requires an ever growing set of learned representations.

In contrast to this, our approach for task transfer aims to more directly tie the learned representations between tasks. To achieve this, we build on the idea of *successor representation* learning for RL first proposed by Dayan [25] and recently combined with deep neural networks in [7], [10]. This line of work makes the observation that Q-learning can be partitioned into two sub-tasks: 1) learning features from which the reward can be predicted reliably and 2) estimating how these features evolve over time. While it was previously noted how such a partitioning can be exploited to speed up learning for cases where the reward changes scale or meaning [7], [10] we here show how this view can be extended to allow general – fast – transfer across tasks, including changes to the environment, the reward function and thus also the optimal policy.

We also note that the objective we use for learning descriptive features involves training a deep auto-encoder. Learning state representations for RL via auto-encoders has previously been considered several times in the literature [26], [27], [28]. Among these, utilizing the priors on learned representation for robotics from Jonschkowski *et al.* [27] could potentially further improve our model.

III. Background

In this section we will first review the concepts of reinforcement learning upon which we build our approach.

A. Reinforcement learning

We formalize the navigation task as a *Markov Decision Process* (MDP). In an MDP an agent interacts with the environment through a sequence of observations, actions and reward signals. In each time-step $t \in [0, T]$ of the decision process the agent first receives an observation from the environment $\mathbf{x}_t \in \mathcal{X}$ (in our case an image of its surrounding). Together with a history of recent observations $\{\mathbf{x}_{t-H}, \dots \mathbf{x}_{t-1}\}$ – with history length H –, $\mathbf{x}_t$ informs the agent about the true state of the environment $\mathbf{s}_t \in \mathcal{S}$. In the following always define $\mathbf{s}_t$ as $\mathbf{s}_t = \{\mathbf{x}_{t-H}, \dots \mathbf{x}_{t-1}, \mathbf{x}_t\}$.The agent then selects an action $\mathbf{a}_t \in \mathcal{A}$ according to a policy $\mathbf{a}_t = \pi(\mathbf{s}_t)$ [1] and transits to the next state $\mathbf{s}_{t+1}$ following the dynamics of the environment: $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, receiving reward $R(\mathbf{s}_t) \in \mathbb{R}$ and obtaining a new observation $\mathbf{x}_{t+1}$. The agent's goal is to maximize the cumulative expected future reward (with discount factor γ). This quantity uniquely assigns an expected value to each state-action pair. The action-value function (referred to as the Q-value function) of executing action $\mathbf{a}$ in state $\mathbf{s}$ under a policy π thus can be defined as:

$$Q(\mathbf{s}, \mathbf{a}; \pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(\mathbf{s}_t) \Big| \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}, \pi\right], \quad (1)$$

where the expectation is taken over the policy dynamics: the transition dynamics under policy π. Importantly, the Q-function can be computed using the Bellman equation

$$Q(\mathbf{s}_t, \mathbf{a}_t; \pi) = R(\mathbf{s}_t) + \gamma \mathbb{E}\left[Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}; \pi)\right], \quad (2)$$

which allows for recursive estimation procedures such as Q-learning and SARSA [29]. Furthermore, assuming the Q-function for a given policy is known, we can find an improved policy $\hat{\pi}$ by greedily choosing $\mathbf{a}_t$ in each state: $\hat{\pi}(\mathbf{s}_t) = \operatorname{argmax}_{\mathbf{a}_t} Q(\mathbf{s}_t, \mathbf{a}_t; \pi)$.

When combined with powerful function approximators such as deep neural networks these principles form the basis of many recent successes in RL for control.

[1] We restrict the following presentation to deterministic policies with discrete actions to simplify notation. A generalization can easily be obtained.

B. Successor feature reinforcement learning

While direct learning of the Q-value function from Equation (1) with function approximation is possible, it results in a black-box approximator which makes knowledge transfer between tasks challenging (we refer to Sec. II for a discussion). We will thus base our algorithm upon a re-formulation of the RL problem first introduced by [25] called *successor representation* learning which has recently also been combined with deep neural networks [7], [10], that we will first review here and then extend to naturally handle task transfer.

To start, we assume that the reward function can be approximately represented as a linear combination of learned features $\phi(\mathbf{s};\theta_\phi)$ (in our case features extracted from a neural network) with parameters θ_ϕ and a reward weight vector ω as $R(\mathbf{s}) \approx \phi(\mathbf{s};\theta_\phi)^T\omega$. Using this assumption we can rewrite Equation (1) as

$$\begin{aligned} Q(\mathbf{s},\mathbf{a};\pi) &\approx \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t\phi(\mathbf{s}_t;\theta_\phi)\cdot\omega\Big|\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a},\pi\right] \\ &= \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t\phi(\mathbf{s}_t;\theta_\phi)\Big|\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a},\pi\right]\cdot\omega \\ &= \psi^\pi(\mathbf{s},\mathbf{a})^T\omega, \end{aligned} \tag{3}$$

where, in line with [10] we refer to $\psi^\pi(\mathbf{s},\mathbf{a}) = \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t\phi(\mathbf{s}_t;\theta_\phi)|\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a},\pi\right]$ as the *successor features*. Consequently we will refer to the whole reinforcement learning algorithm as successor feature reinforcement learning (SF-RL). As a special case we will assume that the features $\phi(\mathbf{s};\theta_\phi)$ themselves are representative of the state $\mathbf{s}$ (i.e., we can reconstruct the state $\mathbf{s}$ from $\phi(\mathbf{s};\theta_\phi)$ alone) which allows us to explicitly turn $\psi(\cdot)$ into a function $\psi^\pi(\phi(\mathbf{s}_t;\theta_\phi),\mathbf{a}_t)$. In the following we use the short-hand $\phi_\mathbf{s} = \phi(\mathbf{s};\theta_\phi)$ – omitting the dependency on the parameters θ_ϕ – and write $\psi^\pi(\phi_{\mathbf{s}_t},\mathbf{a}_t)$ to avoid cluttering notation.

Interestingly, these successor features can again be computed via a Bellman equation in which the reward function is replaced with $\phi_{\mathbf{s}_t}$; that is we have:

$$\psi^\pi(\phi_{\mathbf{s}_t},\mathbf{a}_t) = \phi_{\mathbf{s}_t} + \gamma\mathbb{E}\left[\psi^\pi\left(\phi_{\mathbf{s}_{t+1}},\mathbf{a}_{t+1}\right)\right]. \tag{4}$$

And we can thus learn approximate successor features using a deep Q-learning like procedure [10], [7]. Effectively, this re-formulation separates the learning of the Q-function into two problems: 1) estimating the expectation of descriptive features under the current policy dynamics and 2) estimating the reward obtainable in a given state.

To show how learning with successor feature RL works let us consider the case where we are only interested in recovering the Q-function of the optimal policy π^*. In this case we can simultaneously learn the parameters θ_ϕ of the feature mapping $\phi_\mathbf{s}$ (a convolutional neural network), the reward weights ω and an approximate successor features mapping $\psi(\phi_\mathbf{s},\mathbf{a};\theta_\psi) \approx \psi^{\pi^*}(\phi_\mathbf{s},\mathbf{a})$ (a fully connected network with parameters θ_ψ) by alternating stochastic gradient descent steps on two objective functions:

$$L(\theta_\psi) = \mathop{\mathbb{E}}_{(s,a,s')\in\mathcal{D}_T}\left[\left(\phi_\mathbf{s} + \gamma\psi(\phi_{\mathbf{s}'},\mathbf{a}^*;\theta_\psi^-) - \psi(\phi_\mathbf{s},\mathbf{a};\theta_\psi)\right)^2\right], \tag{5}$$

$$\begin{aligned} &L(\theta_\phi,\theta_d,\omega) \\ &\quad= \mathop{\mathbb{E}}_{(s,R(s))\in\mathcal{D}_R}\Big[\left(R(\mathbf{s}) - \phi_\mathbf{s}^T\omega\right)^2 + \left(\mathbf{s} - d(\phi_\mathbf{s};\theta_d)\right)^2\Big], \end{aligned} \tag{6}$$

where $\mathcal{D}_T$ and $\mathcal{D}_R$ denote collected experience data for transitions and rewards, respectively, $\mathbf{a}^* = \text{argmax}_{a'}\, Q(\mathbf{s}',\mathbf{a}';\pi^*)$ – computed by inserting the approximate successor features $\psi(\phi_{\mathbf{s}'},\mathbf{a}^*;\theta_\psi^-)$ into Equation (3) – and where θ_ψ^- denotes the parameters of the current target successor feature approximation. To provide stable learning these are occasionally copied from θ_ψ (a full discussion of the intricacies of this approach is out of the scope of this paper and we refer to [5] and [7] for details); we replace the target successor feature parameters every 5000 training steps.

The objective function from Equation (5) corresponds to learning the successor features via online Q-learning (with rewards $\phi(\cdot)$). The objective from Equation (6) corresponds to learning the reward weights and the CNN feature mapping and consists of two parts: the first part ensures that the reward is regressed; the second part ensures that the features are representative of the state $\mathbf{s}$ by enforcing that an inverse mapping from $\phi(\mathbf{s};\theta_\phi)$ to $\mathbf{s}$ exists through a third convolutional network, a decoder $d(\cdot)$, whose parameters θ_d are also learned. After learning, actions can be chosen greedily from $Q(\mathbf{s},\mathbf{a};\pi^*)$ by inserting the approximated successor features into Equation (3).

IV. Transferring successor features to new goals and tasks

As described above, the successor representation naturally decouples task specific reward estimation and the estimation of the expected occurrence of the features $\phi(\cdot)$ under the specific policy dynamics. This makes successor feature based RL a natural choice when aiming to transfer knowledge between related tasks. To see this let us first define two notions of knowledge transfer. In both cases we assume that the learning occurs in K different stages during each of which the agent can interact with a distinct task $k \in [1,K]$. The aim for the agent is to solve all tasks at the end of training, using minimal interaction time for each task. From the perspective of reinforcement learning this setup corresponds to a sequence of K RL problems which have shared structure. Knowledge transfer between tasks can then occur for two different scenarios:

The first, and simplest, notion of knowledge transfer occurs if all K tasks use the same environment and transition dynamics and differ only in the reward function R. In a navigation task this would be equivalent to finding paths to K different goal positions in one single maze.

The second, and more general, notion of knowledge transfer occurs if all K tasks use different environments

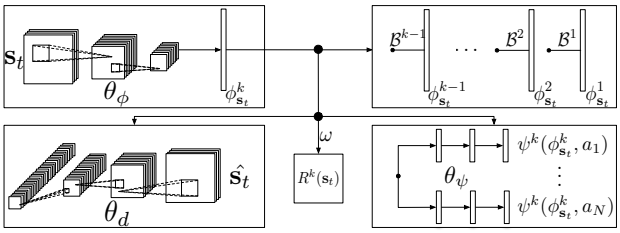


Fig. 2: Visualization of the model architecture: θ_ϕ parameterizes a convolutional network for extracting features $\phi^k_{\mathbf{s}_t}$ (k is the current target task) from $\mathbf{s}_t$ (contains three convolutional layers, with the first layer consisting of 32 8×8 filters with stride 4, the second of 64 4×4 filters with stride 2 and the 3rd of 64 3×3 filters with stride 1, each followed by a rectifying nonlinearity; the last layer is followed by one fully-connected layer with 512 units); θ_d reconstructs $\mathbf{s}_t$ back from $\phi^k_{\mathbf{s}_t}$ (contains five deconvolutional layers, with feature sizes $\{512, 256, 128, 64, 4\}$ and increasing spatial dimensionality in factors of 2); ω regresses the immediate reward $R^k(\mathbf{s}_t)$ out of the state representation $\phi^k_{\mathbf{s}_t}$; θ_ψ computes the successor features $\psi^k(\phi^k_{\mathbf{s}_t}, a_n; \theta_{\psi^k})$ for each $a_n \in \mathcal{A}$ (contains two fully-connected layers); $\mathcal{B}^i$ maps the features of the current task k back to those of the old tasks.

(and potentially different reward functions) which share some similarities within their state space. In a navigation task this includes changing the maze structure or robot dynamics between different tasks.

We can observe that successor feature RL lends itself well to transfer learning in scenarios of the first kind: If the features $\phi(\cdot)$ are expressive enough to ensure that the rewards for all tasks can be linearly predicted from them then for all tasks following the first (i.e., for $k > 1$) one only has to learn a new reward weight vector ω^k (keeping both the learned $\phi(\cdot)$ and $\psi(\cdot)$ fixed), although care has to be taken if the expectation of the features under the different policy changes (in which case the successor features would have to be adapted also). Learning for $k > 1$ then boils down to solving a simple regression problem (i.e., minimizing Eq. (6) wrt. ω) and requires only the storage of an additional weight vector per task. This idea has recently been explored in [10], [7], with [7] showing large learning speedups for a special case of this setting where they changed the scale of the final reward. We here argue that successor feature RL can be easily extended to transfer learning of the second kind with minimal additional memory and computational requirements.

Specifically, to derive a learning algorithm that works for both transfer scenarios let us first define the action-value function for task k using the successor feature notation as

$$Q^k(\mathbf{s}, \mathbf{a}; \pi^k) \approx \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi^k_{\mathbf{s}_t} \middle| \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}, \pi^k\right] \cdot \omega^k,$$

where we used the superscript k to refer to task specific features and policies respectively and where we have again introduced the short-hand notation $\phi^k_{\mathbf{s}_t} = \phi^k(\mathbf{s}_t; \theta_{\phi^k})$ for notational brevity. Additionally, let us assume that there exists a linear relation between the task features, that is there exists a mapping $\phi^i_{\mathbf{s}} = \mathcal{B}^i \phi^k_{\mathbf{s}}$ for all $i \leq k$ and we have $B^k = \mathbf{I}$. We note that such a linear dependency between features does not imply a linear dependency between the observations (since $\phi(\cdot)$ is a nonlinear function implemented by a neural network), and hence this assumption is not very restrictive. Then – again using the fact that the expectation is a linear operator – we obtain for $i \leq k$:

$$\begin{aligned} Q^i(\mathbf{s}, \mathbf{a}; \pi^i) &\approx \mathbb{E}\left[\sum_{t=0}^{\infty} \mathcal{B}^i \gamma^t \phi^k_{\mathbf{s}_t} \middle| \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}, \pi^i\right] \cdot \omega^i \\ &= \mathcal{B}^i \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi^k_{\mathbf{s}_t} \middle| \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}, \pi^i\right] \omega^i \\ &= \mathcal{B}^i \psi^{\pi^i}\left(\phi^k_{\mathbf{s}_t}, \mathbf{a}\right)^T \omega^i \qquad (7) \\ &= \psi^{\pi^i}\left(\mathcal{B}^i \phi^k_{\mathbf{s}_t}, \mathbf{a}\right)^T \omega^i. \qquad (8) \end{aligned}$$

These equivalences now give us a straight-forward way to transfer knowledge to new tasks while keeping the solution found for old tasks intact (as long as we have access to all feature mappings ϕ^k and policies π^k):

1) When training on task $k > 1$ initialize the parameters θ_{ϕ^k} and θ_{ψ^k} with $\theta_{\phi^{k-1}}$ and $\theta_{\psi^{k-1}}$ respectively (otherwise initialize randomly) and train ψ^{π^k} and ϕ^k via stochastic gradient descent on Eqs. (5)-(6).
2) In addition, train all $\mathcal{B}^i$ with $i < k$ to preserve the relation $\phi^i_{\mathbf{s}} \approx \mathcal{B}^i \phi^k_{\mathbf{s}}$.
3) To obtain successor features for the previous tasks, estimate the expectation of the features for the current task k under the old task policies to obtain $\psi^{\pi^i}\left(\phi^k_{\mathbf{s}}, \mathbf{a}\right)$ so that Eq. (7) can be computed during evaluation. Note that this means we have to estimate the expectation of the current task features under all old task dynamics and policies[2]. Since we expect significant overlap between tasks in our experiments this can be implemented memory efficiently by using one single neural network with multiple output layers to implement all task specific successor features. Alternatively, if the successor feature networks are small, one can just preserve the old task successor feature networks and use Eq. (8) for selecting actions for old tasks.

When – as in Sec. III-B – we are only interested in finding the optimal policy π^{i^*} for each task these steps correspond to alternating stochastic gradient descent steps on two objective functions analogous to Equations (5) and (6), under the model architecture depicted in Fig. 2. More precisely, we write $\psi^i(\phi^k_{\mathbf{s}}, \mathbf{a}; \theta_{\psi^i}) \approx \psi^{\pi^{i^*}}(\phi^k_{\mathbf{s}}, \mathbf{a})$ and obtain the following objectives for task k:

[2]In principle, the expectations for all tasks $i < k$ need to be evaluated with samples from these tasks. In our case, we however found that the shared structure between tasks was large enough to allow for estimating all expectations based on the current tasks samples only.

$$L^k\Big(\{\theta_{\psi^1},\dots,\theta_{\psi^k}\}\Big) = \sum_{i\leq k} \mathop{\mathbb{E}}_{\substack{(s,a,s')\\ \in\mathcal{D}_T^i}} \left[\left(\phi^k_{\mathbf{s}} + \gamma\psi^i(\phi^k_{\mathbf{s}'}, \mathbf{a}^{i*}; \theta^-_{\psi^i}) - \psi^i(\phi^k_{\mathbf{s}}, \mathbf{a}; \theta_{\psi^i})\right)^2\right], \tag{9}$$

$$L^k\Big(\theta_\phi, \theta_d, \omega^k, \{\mathcal{B}^1,\dots,\mathcal{B}^{k-1}\}\Big) = \mathop{\mathbb{E}}_{(s,R(s))\in\mathcal{D}_R^k} \left[\left(R(\mathbf{s}) - {\phi^k_{\mathbf{s}}}^T\omega^k\right)^2 + \left(\mathbf{s} - d(\phi^k_{\mathbf{s}}; \theta_{d^k})\right)^2\right] + \sum_{i<k} \mathop{\mathbb{E}}_{(s,R(s))\in\mathcal{D}_R^i} \left[\left(\phi^i_s - \mathcal{B}^i\phi^k_s\right)^2\right], \tag{10}$$

where $a^{i*} = \text{argmax}_{a'} Q(\mathbf{s}', \mathbf{a}'; \pi^{i^*})$ is the current greedy best action for task i and in cases where we are willing to store the old $\psi^i(\cdot)$ for $i < k$ Eq. (9) only needs to be optimized with respect to θ_{ψ^k} (dropping all other terms)[3]. Several interesting details can be noted about this formulation. First, if we assume that all $\psi^i(\cdot)$ are implemented using one neural network with k output layers – or if the successor feature networks are small – then the overhead for learning k tasks is small (we only have to store $k-1$ additional weight matrices plus one additional reward weight vector per task) this is in contrast to other successful transfer learning approaches for RL that have recently been proposed such as [21]. Second, the regression of the old task features via the transformation matrices $\mathcal{B}^i$ forces the CNN that outputs ϕ^k_s to represent the features for all tasks [4]. As such we expect this approach to work well when tasks have shared structure; if they have no shared structure one would have to increase the number of parameters (and thus possibly the dimensionality of ϕ^k).

To gain some intuition for the reasons why the above model should work we here want to give a – hypothetical – example: Let us assume the set of extracted features $\phi(\cdot)$ to be the relative distance to a set of objects from the current position of the agent. Then, the successor features $\psi(\cdot)$ would estimate the discounted sum of those relative distances under the current policy dynamics. When transferring to a new environment, the spatial relationship of the objects could, for example, change. Then $\phi(\cdot)$ would need to adapt accordingly. But since we assume the two environments to share structure (e.g., they contain the same objects), filters in the early layers of $\phi(\cdot)$ could be largely reused (or transferred). The adapted features (e.g., the relative distances from the current pose to changed objects) now would differ from those of the previous environments, this change in scale could be directly captured by a linear mapping $\mathcal{B}$. $\psi(\cdot)$ would also need to be adapted, but due to the shared structure between environments and their similarity in the successor features we would expect adapting them to be fast. Similarly, the reward mapping can either be re-learned quickly or transferred directly (e.g., if we assume that the reward penalizes proximity to objects).

[3]In practice there is no noticeable performance difference.

[4]May be seen as special case of the distillation technique [30].

V. Simulated Experiments

A. Experimental setup

We first test our algorithm using a simulation of different maze-like 3D environments. The environment contains cubic objects and a target for the agent to reach (rendered as a green sphere) (cf. Fig. 3). We model the legal actions as four discrete choices: {stand still, turn left (90°), turn right (90°), go straight ($1m$) } to simplify the problem (we note that in simulation the agent moves in a continuous manner). The agent is a simulated Pioneer-3dx robot moving under a differential drive model (with Gaussian control noise, thus the robot will have observations of the environment from a continuous viewing position and angle space).

The agent obtains a reward of -0.04 for each step it takes, -0.96 for colliding with obstacles, 1 for reaching the goal (this reward structure forces time-optimal behavior). Each episode starts with the agent in a random location and ends when it reaches the goal (unless noted otherwise).

In each time-step the agent receives as an observation a frame captured from the forward facing camera (as shown in Fig. 3, re-scaled to 64×64 pixels). The state in each time-step is then given by the 4 most recently obtained observations. The top-down views of the four different mazes we consider are shown in Fig. 5.

For training the model (Fig. 2) we employed stochastic gradient descent with the ADAM optimizer [31], a mini-batch size of 64 and a learning rate of 2.5×10^{-4} and 2.5×10^{-5} for visual inputs for the supervised learner and the reinforcement learners respectively, 5.0×10^{-5} for depth inputs. We performed a coarse grid search for each learning algorithm to choose the optimizer hyper parameters (learning rate in range $[1 \times 10^{-6}, 1 \times 10^{-3}]$) and use the same minibatch size across all considered approaches. Training was performed alongside exploration in the environment (one batch is considered every 4 steps).

B. Baseline method - supervised learning & DQN

As a baseline for our experiments, we train a CNN by supervised learning to directly predict the actions computed by an A^* planner from the same visual input that the SF-RL model receives. The network structure is the same as the CNN from the SF-RL model (θ_ϕ) and differs only in that the output 512 units are fed into a final softmax layer. As an additional baseline we also compare to the DQN

Fig. 3: Exemplary views the agent observes in the simulated environment.

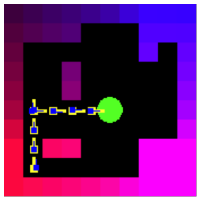
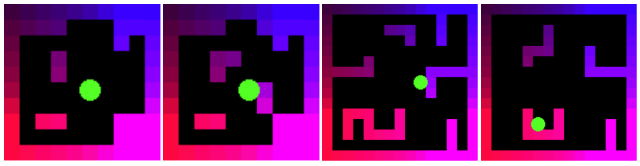


Fig. 4: Comparison between the true (yellow) and the predicted (blue) poses.



(a) Map1 (b) Map2 (c) Map3 (d) Map4

Fig. 5: Top-down schematic view of the four different maze environments we consider for the simulated experiments.

approach from [5]. To ensure a fair comparison we evaluate DQN by learning from scratch and in a transfer learning situation in which we finetune the DQN model trained on the base task; such a fine-tuning approach is known to perform better than simply transferring with fixed features [32], [21] (we also conduct transfer learning experiments with fixed features for DQN for completeness).

The training data for the supervised learner is generated beforehand, consisting of $1.6e5$ labeled samples. To generate these samples, full localization is required, while for evaluating the learned network it is not required. As such, this setup can be thought of as the best case scenario for training a CNN to imitate a planner in this domain.

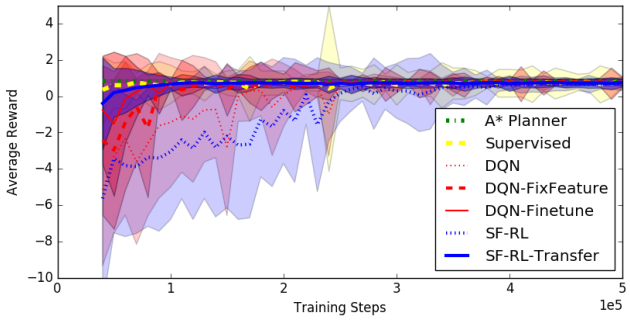
To ensure a fair comparison between different methods in the following plots, we scale the number of steps taken by the supervised learner, so that the number of updates matches that of the SF-RL model and that of DQN (the two reinforcement learners start learning at $3e4$ iterations and makes an update every 4 steps after that).

C. Visual navigation in a 3D maze

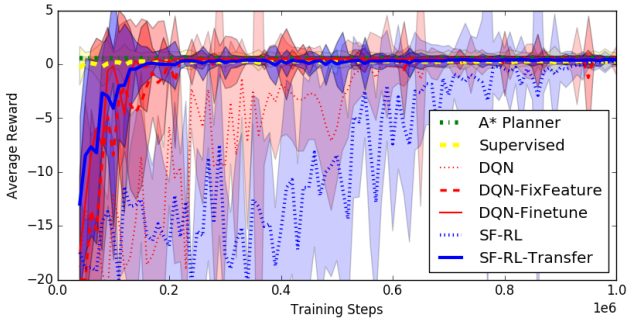
For the first experiment we trained our deep successor feature reinforcement learner (SF-RL), DQN and the supervised learner on the base map: *Map1* (Fig. 5a). To compare the algorithms we perform a testing phase every 10,000 steps consisting of evaluating the performance of the current policy for 5,000 testing steps.

1) Base environment: We first train on *Map1* from scratch. We observe that the supervised learning and reinforcement learning (DQN and SF-RL) models converge to performance comparable to the optimal A^* planner. We also observe that the supervised learner converges significantly faster in this experiment. This is to be expected since it has access to the optimal paths – as computed via A^* – for starting positions covering the whole environment right from the beginning of training. In contrast to this, the reinforcement learners gradually have to build up a dataset of experience and can only make use of the sparsely distributed reward signal to evaluate the actions taken.

2) Transfer to different environment: We then perform a transfer learning experiment (using the trained models from above) to a changed environment *Map2* where more walls are added (Fig. 5b). In Fig. 6a we show a performance comparison between the supervised learner (*Supervised*), DQN learning from scratch (*DQN*) and using task transfer (with fixed CNN layers: *DQN-FixFeature*, and by fine-tuning the whole network: *DQN-Finetune*), SF-RL from scratch (*SF-RL*) and using task transfer (*SF-RL-Transfer*).



(a) Comparison between learning algorithms for *Map2*.



(b) Comparison between learning algorithms for *Map4*.

Fig. 6: Average reward $\pm$ one standard deviation obtained by A^* using the true system model, the supervised learner, as well as DQN and SF-RL when learning from scratch and with task transfer from *Map1* (6a) and *Map3* (6b).

We observe that *SF-RL-Transfer* converges to performance comparable to the optimal policy much faster than training from scratch. Furthermore, in Fig. 6a the learning speed of *SF-RL-Transfer* is even comparable to that of *Supervised*, who is learning directly from perfectly labeled actions. We observe that when training from scratch, *DQN* is slightly faster than *SF-RL* (we attribute this to the fact that *SF-RL* optimizes a more complicated loss function including e.g. an auto-encoder loss). In the transfer learning setting *SF-RL-Transfer* is comparable to *DQN-Finetune*, and converges faster than *DQN-FixFeature*. It is important to realize that our method preserves the ability to solve the old task after this transfer occurred, which *DQN-Finetune* is not capable of. To verify this preservation of the old policies we re-evaluated *DQN-Finetune* and *SF-RL-Transfer* on all tasks and summarize the results in Tab. I (*DQN-FixFeature* keeps the network for the initial tasks completely unchanged thus it is unnecessary to evaluate its performance again). We note that our agent is still able to perform well on the old task, while the DQN agent deviated significantly from the optimal policy (it is still able to solve most of the episodes in this case via a "random-walk"). We also want to emphasize that in contrast to *DQN-FixFeature*, *SF-RL-Transfer* has the ability of continuously adapting its features to new tasks while keeping a mapping to all previous task features. Additionally, *DQN-FixFeature* has to perform the same transfer procedure for all kinds of transfer scenarios due to its black-box property; while with the flexibility of the more structured representation

TABLE I: Final testing statistics for all considered environments, each evaluated from 50 random starting positions. The maximum number of steps per episode was: 200 steps for *Map1&2*, 500 steps for *Map3&4*.

Pre-train on / Transfer to	Success ratio	Reward	Steps
Testing on Map1			
A^* baseline		0.814 ± 0.070	5.640 ± 1.747
DQN-Finetune			
Map1 / -	50/50	0.791 ± 0.114	6.220 ± 2.845
Map1 / Map2	48/50	0.398 ± 1.755	15.000 ± 38.800
SF-RL-Transfer			
Map1 / -	50/50	0.765 ± 0.243	6.410 ± 3.915
Map1 / Map2	50/50	0.733 ± 0.235	6.796 ± 2.999
Testing on Map3			
A^* baseline		0.635 ± 0.138	10.120 ± 3.438
DQN-Finetune			
Map3 / -	50/50	0.566 ± 0.178	11.84 ± 4.442
Map3 / Map4	4/50	-18.335 ± 5.703	460.46 ± 135.450
SF-RL-Transfer			
Map3 / -	50/50	0.489 ± 0.348	13.460 ± 5.936
Map3 / Map4	50/50	0.444 ± 0.416	13.780 ± 8.707

of *SF-RL-Transfer*, we only need to retrain the successor feature network θ_ψ and keep the reward mapping ω fixed when only the dynamics changes, or if the dynamics of the environment stay fixed or close to the already observed dynamics, *SF-RL-Transfer* can adapt quickly by either changing only ω or in combination with θ_ψ.

D. More complicated transfer scenarios

We then experiment in a more complicated transfer scenario: transferring a base controller from *Map3* (Fig. 5c) to *Map4* (Fig. 5d). As can be seen from the visualization, the objects change significantly from *Map3* to *Map4*. Also, the goal location moves from the center of an open area to a "hidden" corner. The results for this experiment are depicted in Fig. 6b, revealing a similar trend as for the simpler mazes. A re-evaluation of the *DQN-Finetune* and *SF-RL-Transfer* agent is shown in Tab. I. We note that the *DQN-Finetune* agent loses the policy for *Map3* after being transferred to *Map4* as the locations of the target and objects are changed dramatically, while our agent still is able to solve the old task after the transfer.

Furthermore, when transferring from *Map1* to *Map2* we move from a simpler to a more complicated environment, while *Map4* is "simpler" than *Map3*.

E. Analysis of learned representation

As an additional test, we analyzed the representation ϕ_s^k learned by the SF-RL approach. Specifically, since the reward is defined on the pose of the agent and optimal path finding clearly depends on the agent being able to localize itself we analyzed as to whether ϕ_s^k encodes the robot pose. To answer this, we extract features ϕ_s^k for all states along collected optimal trajectories and regressed the ground truth poses of the robot (obtained from our simulator) using a neural network with two hidden layers (128 units each). Fig. 4 shows the results from this experiment, overlaying the ground truth poses with the predicted poses from our regressor on a held out example. From these we can conclude that indeed, the transition dynamics is encoded and the agent is able to localize itself, and this information can reliably be retrieved post-hoc (i.e., after training).

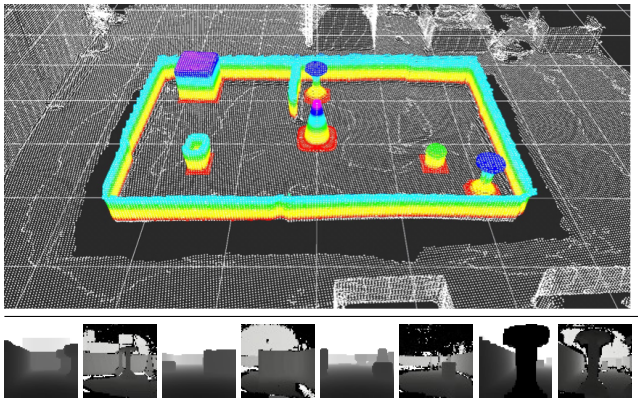


Fig. 7: 3D Model of *Map5*. The robot should avoid the colored regions containing objects and navigate to the traffic cone in the center. The bottom part shows pairs of images comparing between the rendered depth images from our simulator, and real depth images taken by a kinect camera at approximately the same pose in a real environment modeled after the simulator.

VI. Real-World Experiments

In order to show the applicability of our method to more realistic scenarios, we conducted additional experiments using a real robot. We start by swapping the RGB camera input for a simulated depth sensor in simulation and then perform a transfer learning experiment to a different, real, environment from which we collect real depth images.

A. Rendered Depth Experiments

To obtain a scenario more similar to a real world scene we might encounter, we build a maze-like environment *Map5* (Fig. 7) in our robot simulator that includes realistic walls and object models. In this setting the robot has to navigate to the target (traffic cone in the center) and avoid colliding with objects and walls. We then simulate the robot within this environment, providing rendered depth images from a simulated kinect camera as the input modality (as opposed to the artificial RGB images we used before).

B. Real World Transfer Experiments

We then change to a real robot experiment in which the robot can explore the maze depicted in *Map6* (Fig. 1) (note that the position of the objects and the target are changed from the simulated environment *Map5* in Fig. 7). We collect real depth images in the actual maze-world using the on-board kinect sensor of a Robotino. To avoid training for long periods of time in the real environment we pre-recorded images at all locations that the robot can explore (taking 100 images per position and direction with randomly perturbed robot pose to model noise).

The results of training from scratch in this real environment as well as when transfer from the simulated environment is performed are depicted in Fig. 8 (the agent

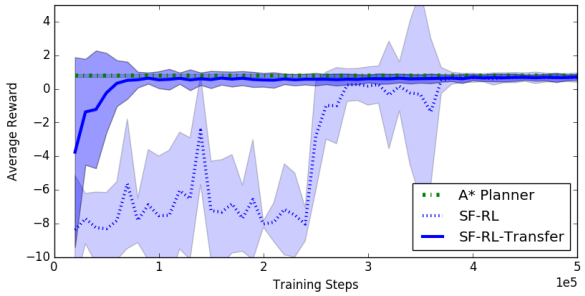


Fig. 8: Comparison between SF-RL trained on the real world *Map6*, and SF-RL-Transfer with the base model trained on the simulated *Map5* transferred to *Map6*.

starts to learn here after $1e4$ steps, whereas in previous experiments this number is set to $3e4$). Similar to the previous experiments we see a large speed-up when transferring knowledge even though the simulated depth images contain none of the characteristic noise patterns present in the real-world kinect data. We note that the agent achieves satisfactory performance at around 60,000 iterations, which corresponds to approximately 8 hours of real experience (assuming data is collected at a rate of 2Hz).

After training with the pre-recorded images, the robot is tested in real world environments. A video of the real experiments in two changed environments: *Map6* & *Map7* (*Map7* is not discussed here due to space constraints) can be found at: `https://youtu.be/RYlrWl632sM`.

VII. Conclusion

We presented a method for solving robot navigation tasks from raw sensory data, based on an extension of the theory behind successor feature reinforcement learning. Our algorithm is able to naturally transfer knowledge between related tasks and yields substantial speedups over deep reinforcement learning from scratch in the experiments we performed. Despite of these encouraging results, there are several opportunities for future work including testing our approach in more complicated scenarios and extending it to more naturally handle partial observability.

References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[2] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[3] J.-C. Latombe, *Robot Motion Planning*. Kluwer, 1991.

[4] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, 2015.

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. of the International Conference on Learning Representations (ICLR)*, 2016.

[7] T. D. Kulkarni, A. Saeedi, S. Gautam, and S. J. Gershman, "Deep successor reinforcement learning," *arXiv preprint arXiv:1606.02396*, 2016.

[8] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research (JMLR)*, 2016.

[9] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, D. Blei and F. Bach, Eds. JMLR Workshop and Conference Proceedings, 2015.

[10] A. Barreto, R. Munos, T. Schaul, and D. Silver, "Successor features for transfer in reinforcement learning," *arXiv preprint arXiv:1606.05312*, 2016.

[11] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proc. of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

[12] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. of the 33nd International Conference on Machine Learning, (ICML)*, 2016.

[13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. of the International Conference on Learning Representations (ICLR)*, 2016.

[14] R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup, "Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction." in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume*, 2011.

[15] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *Proc. of the 32nd International Conference on Machine Learning (ICML)*, 2015.

[16] M. Ring, "Continual learning in reinforcement environments," *PhD thesis, Oldenbourg Verlag*, 1995.

[17] M. E. Taylor and P. Stone, "An introduction to inter-task transfer for reinforcement learning," *AI Magazine*, vol. 32, no. 1, 2011.

[18] A. Wilson, A. Fern, S. Ray, and P. Tadepalli, "Multi-task reinforcement learning: a hierarchical Bayesian approach," in *Proc. of the 24th International Conference on Machine Learning, (ICML)*, 2007.

[19] E. Parisotto, L. J. Ba, and R. Salakhutdinov, "Actor-mimic: Deep multitask and transfer reinforcement learning," in *Proc. of the International Conference on Learning Representations (ICLR)*, 2016.

[20] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," in *Proc. of the International Conference on Learning Representations (ICLR)*, 2016.

[21] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[22] L. Tai and M. Liu, "Towards cognitive exploration through deep reinforcement learning for mobile robots," *arXiv preprint arXiv:1610.01733*, 2016.

[23] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," *arXiv preprint arXiv:1609.05143*, 2016.

[24] L. Tai and M. Liu, "Deep-learning in mobile robotics-from perception to control systems: A survey on why and why not," *arXiv preprint arXiv:1612.07139*, 2016.

[25] P. Dayan, "Improving generalization for temporal difference learning: The successor representation," *Neural Computation*, vol. 5, no. 4, 1993.

[26] M. Riedmiller, S. Lange, and A. Voigtlaender, "Autonomous reinforcement learning on raw visual input data in a real world application," in *IJCNN*, 2012.

[27] R. Jonschkowski and O. Brock, "Learning state representations with robotic priors," *Autonomous Robots*, vol. 39, no. 3, 2015.

[28] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2016.

[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning : An Introduction*. MIT Press, 1998.

[30] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[31] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. of the International Conference on Learning Representations (ICLR)*, 2015.

[32] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems*, 2014, pp. 3320–3328.