PsiPhi-Learning: Reinforcement Learning with Demonstrations using Successor Features and Inverse Temporal Difference Learning

Angelos Filos 1  Clare Lyle 1  Yarin Gal 1  Sergey Levine 2  Natasha Jaques * 2 3  Gregory Farquhar * 4

Abstract

We study reinforcement learning (RL) with no-reward demonstrations, a setting in which an RL agent has access to additional data from the interaction of other agents with the same environment. However, it has no access to the rewards or goals of these agents, and their objectives and levels of expertise may vary widely. These assumptions are common in multi-agent settings, such as autonomous driving. To effectively use this data, we turn to the framework of successor features. This allows us to disentangle shared features and dynamics of the environment from agent-specific rewards and policies. We propose a multi-task inverse reinforcement learning (IRL) algorithm, called inverse temporal difference learning (ITD), that learns shared state features, alongside per-agent successor features and preference vectors, purely from demonstrations without reward labels. We further show how to seamlessly integrate ITD with learning from online environment interactions, arriving at a novel algorithm for reinforcement learning with demonstrations, called ΨΦ-learning (pronounced ‘Sci-Fi’). We provide empirical evidence for the effectiveness of ΨΦ-learning as a method for improving RL, IRL, imitation, and few-shot transfer, and derive worst-case bounds for its performance in zero-shot transfer to new tasks.

1. Introduction

If artificial agents are to be effective in the real world, they will need to thrive in environments populated by other agents. Agents are typically goal-directed, sometimes by definition (Franklin & Graesser, 1996). While their goals can be different, they often depend on shared salient features of the environment, and may be able to interact with and affect the environment in similar ways. Humans and other animals make ready use of these similarities to other agents while learning (Henrich, 2017; Laland, 2018). We can observe the goal-directed behaviours of other humans, and combine these observations with our own experiences, to quickly learn how to achieve our own goals. If reinforcement learning (RL, Sutton & Barto, 2018) agents could similarly interpret the behaviour of others, they could learn more efficiently, relying less on solitary trial and error.

To this end, we formalise and address a problem setting in which an agent (the ‘ego-agent’) is given access to observations and actions drawn from the experiences of other goal-directed agents interacting with the same environment, but pursuing distinct goals. These observed trajectories are unlabelled in the sense that they lack the goals or rewards of the other agents. This type of data is readily available in many real-world settings, either from (i) observing other agents acting simultaneously with the ego-agent in the same (multi-agent) environment, or (ii) multi-task demonstrations collected independently from the ego-agent’s experiences. Consider autonomous driving as a motivating example: the car can observe the decisions of many nearby human drivers with various preferences and destinations, or may have access to a large offline dataset of such demonstrations. Be-
cause the other agents are pursuing their own varied goals, it can be difficult to directly use this information with conventional imitation learning methods (Widrow & Smith, 1964) or inverse RL (IRL) (Ng et al., 2000; Ziebart et al., 2008).

While the ego-agent should not copy other agents directly, it is likely that the behaviour of all agents depends on shared features of the environment. To disentangle such shared features from agent-specific goals, we turn to the framework of successor features (Dayan, 1993; Kulkarni et al., 2016; Barreto et al., 2017). Successor features are a representation of the sum of state features an agent’s policy will reach in the future. An agent’s goal is represented separately as a preference vector.

In this paper, we demonstrate how a reinforcement learner can benefit from multi-task demonstrations using the framework of successor features. The key contributions are:

1. **Offline multi-task IRL:** We propose an inverse RL algorithm, called inverse temporal difference (ITD) learning. Using only demonstrations, we learn shared state features, alongside per-agent successor features and inferred preferences. The reward functions can be trivially computed from these learned quantities. We show empirically that ITD achieves superior or comparable performance to prior methods.

2. **RL with no-reward demonstrations:** By combining ITD with learning from environment interactions, we arrive at a novel algorithm for RL with unlabelled demonstrations, called ψΦ-learning (pronounced ‘Sci-Fi’). ψΦ-learning is compatible with sub-optimal demonstrations. It treats the demonstrated trajectories as being soft-optimal under some task and employs ITD to recover successor features for the demonstrators’ policies. ψΦ-learning inherits the unbiased, asymptotic performance of RL methods while leveraging the provided demonstrations with ITD. When the goals of any of the demonstrators are even partially aligned with the ψΦ-learner, this enables much faster learning than solitary RL. Otherwise, when the demonstrations are not useful or even misleading, it gracefully falls back to standard RL, unlike naive behaviour cloning or IRL.

3. **Few-shot adaptation with task inference:** Taking full-advantage of the successor features framework, our ψΦ-learner can even adapt zero-shot to new goals it has never seen or experienced during training, but which are partially aligned with the demonstrated goals. This is possible due to the disentanglement of representations into task-specific features (i.e., preferences) and shared state features. We can efficiently update the task-specific preferences and rely on generalised policy improvement for safe policy updates. We derive worst-case bounds for the performance of ψΦ-learning in zero-shot transfer to new tasks.

We evaluate ψΦ-learning in a set of grid-world environments, a traffic-flow simulator (Leurent, 2018), and a task from the ProcGen suite (Cobbe et al., 2020), observing advantages over vanilla RL, imitation learning (Reddy et al., 2019; Ho & Ermon, 2016), and auxiliary-task baselines (Hernandez-Leal et al., 2017). Thanks to the shared state features between the ITD and RL components, we find empirically that the ψΦ-learner not only improves its ego-learning with demonstrations, but also enhances its ability to model others agents using its own experience.

### 2. Background and Problem Setting

We consider a world that can be represented as an infinite horizon controlled Markov process (CMP) given by the tuple: \( C = \langle \mathcal{S}, \mathcal{A}, P, \gamma \rangle \). \( \mathcal{S} \) and \( \mathcal{A} \) represent the continuous state and discrete action spaces, respectively. \( s' \sim P(\cdot|s, a) \) describes the transition dynamics and \( \gamma \) is the discount factor. A task is formulated as a Markov decision process (MDP, Puterman, 2014), characterised by a reward function, \( R: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R} \), i.e., \( M = \langle C, R \rangle \).

The goal of an agent is to find a policy which maps from states to a probability distribution over actions, \( \pi: \mathcal{S} \rightarrow \Delta(\mathcal{A}) \). Maximising the expected discounted sum of rewards \( G^\pi = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \) is known as the value function of the policy \( \pi \). The action-value function of the policy \( \pi \) is given by \( Q^\pi(s, a) = \mathbb{E}^{C, \pi}[G^\pi | s_0 = s, a_0 = a] \), where \( \mathbb{E}^{C, \pi}[\cdot] \) denotes expected value when following policy \( \pi \) in environment \( C \).

#### 2.1. RL with No-Reward Demonstrations

We are interested in settings in which, in addition to an environment \( C \), the agent has also access to demonstrations without rewards, i.e., behavioural data of mixed and unknown quality. The demonstrations are generated by other agents, whose goals and levels of expertise are unknown, and who have no incentive to educate the controlled agent. We will refer to the controlled agent, i.e., reinforcement learner, as the ‘ego-agent’ and to the agents that generated the demonstrations as ‘other-agents’. We denote the demonstrations with \( D = \{ \tau_1, \tau_2, \ldots, \tau_N \} \), where the trajectory \( \tau \triangleq (s_0, a_0, \ldots, s_T, a_T; k) \) is generated by the \( k \)-th agent. Note that each trajectory does include an identifier of the agent that generated it. The ego-agent also gathers its own experience by interacting with the environment, collecting data \( B = \{ (s, a, s', r^{epo}) \} \). Due to the lack of reward annotations in \( D \), and the fact that the data may be irrelevant to the ego-agent’s task, it is not trivial to combine demonstrations from \( D \) with the ego-agent’s experience \( B \).

#### 2.2. Successor Features and Cumulants

To make use of the demonstrations \( D \), we wish to capture the notion that while the agents’ rewards may differ, they share the same environment. To do so we turn to the framework
of successor features (SFs) (Barreto et al., 2017), in which rewards are decomposed into cumulants and preferences:

**Definition 1** (Cumulants and Preferences). The (one-step) rewards are decomposed into task-agnostic cumulants \( \Phi(s, a) \in \mathbb{R}^d \), and task-specific preferences \( w \in \mathbb{R}^d \):

\[
R^w(s, a) \triangleq \Phi(s, a) \top w. \tag{1}
\]

The preferences \( w \) are a representation of a possible goal in the world \( \mathcal{C} \), in the sense that each \( w \) gives rise to a task \( M^w = (\mathcal{C}, R^w) \). We use ‘task’, ‘goal’, and ‘preferences’ interchangeably when context makes it clear whether we are referring to \( w \) itself, or the corresponding \( M^w \) or \( R^w \).

The action-value function for a policy \( \pi \) in \( M^w \) is then a function of the preferences \( w \) and the \( \pi \)'s successor features.

**Definition 2** (Successor Features). For a given discount factor \( \gamma \in [0, 1) \), policy \( \pi \) and cumulants \( \Phi(s, a) \in \mathbb{R}^d \), the successor features (SFs) for a state \( s \) and action \( a \) are:

\[
\Psi^\pi(s, a) \triangleq \mathbb{E}^{\mathcal{C}, \pi} \left[ \sum_{t=0}^{\infty} \gamma^t \Phi(s_t, a_t) \bigg| s_0 = s, a_0 = a \right]. \tag{2}
\]

The \( i \)-th component of \( \Psi^\pi(s, a) \) gives the expected discounted sum of \( \Phi(s, a) \)'s \( i \)-th component, when starting from state \( s \), taking action \( a \) and then following policy \( \pi \). Intuitively, cumulants \( \Phi \) can be seen as a vector-valued reward function and SFs \( \Psi^\pi \) the corresponding vector-valued state-action value function for policy \( \pi \).

An action-value function is then given by the dot product of the preferences \( w \) and \( \pi \)'s SFs:

\[
Q^{\pi, w}(s, a) = \Psi^\pi(s, a) \top w. \tag{3}
\]

**Proof.** See Appendix C. \( \square \)

Note that if we have \( \Psi^\pi \), the value of \( \pi \) for a new preference \( w' \) can be easily computed. This property allows the successor features of a set of policies to be repurposed for accelerating policy updates, as follows.

**Definition 3** (Generalised Policy Improvement). Given a set of policies \( \Pi = \{ \pi_1, \ldots, \pi_K \} \) and a task with reward function \( R \), generalised policy improvement (GPI) is the definition of a policy \( \pi' \) such that

\[
Q^{\pi', R}(s, a) \geq \sup_{\pi \in \Pi} Q^{\pi, R}(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{4}
\]

Provided the SFs of a set of policies, i.e., \( \{ \Psi^{\pi^k} \}_{k=1}^K \), we can apply GPI to derive a new policy \( \pi' \) whose performance on a task \( w \) is no worse that the performance of any of \( \pi \in \Pi \) on the same task, given by

\[
\pi'(s) = \arg\max_{a} \max_{\pi \in \Pi} \Psi^\pi(s, a) \top w. \tag{5}
\]

Eqn. (5) suggests that if we could estimate the SFs of other agents, we could utilise them for improving the ego-agent’s policy with GPI. However, to do so with conventional methods we would require access to their rewards, cumulants and/or preferences. In our setting, we can only observe their sequence of states and actions (Section 2.1). Next, we introduce our method that only requires no-reward demonstrations to estimate SFs and can be integrated seamlessly with GPI for accelerating reinforcement learning.

### 3. Accelerating RL with Demonstrations

We now present a novel method, \( \Psi \Phi \)-learning, that leverages reward-free demonstrations to accelerate RL, shown in Figure 1. Our approach consists of two components: (i) a novel inverse reinforcement learning algorithm, called inverse temporal difference (ITD) learning, for learning cumulants, per-agent successor features and corresponding agent preferences from demonstration without reward labels, and (ii) a novel RL algorithm that combines ITD with generalised policy improvement (GPI, Section 2.2).

#### 3.1. Inverse Temporal Difference Learning

Given demonstrations without rewards, \( D \), we model the agents that generated the data (i.e., blue nodes in Figure 1) as soft-optimal for an unknown task. In particular, the \( k \)-th agent’s policy is soft-optimal under task \( w^k \) and is given by

\[
\pi^k(a|s) = \frac{\exp(\Psi^\pi^k(s, a) \top w^k)}{\sum_a \exp(\Psi^\pi^k(s, a) \top w^k)}, \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{6}
\]

We choose to represent the action-value functions of the other agents with their SFs and preferences to enable GPI, and to expose task- and policy-agnostic structure in the form of shared cumulants \( \Phi \). The \( k \)-th agent’s successor features are temporally consistent with these cumulants \( \Phi \)

\[
\Psi^\pi^k(s, a) = \Phi(s, a) + \gamma \mathbb{E}^{\mathcal{C}, \pi^k} \left[ \Psi^\pi^k(s', \pi^k(s)) \right]. \tag{7}
\]

To learn these quantities from \( K \) demonstrators, we parameterise the SFs with \( \theta^k \), preferences with \( w^k \), and shared cumulants with \( \theta_\Phi \). A schematic of the model architecture and further details are provided in Appendix B. The parameters are learned by minimising a behavioural cloning and SFs TD loss based on equations (6) and (7).

**Behavioural cloning loss.** Given demonstrations generated only by the \( k \)-th agent, i.e., \( D^k \subset D \), we train its successor features \( \theta^k \), and the preferences \( w^k \) by minimising the negative log-likelihood of the demonstrations

\[
\mathcal{L}_{BC-Q}(\theta^k, w^k) \triangleq -\mathbb{E}_{\tau \sim D^k} \log \frac{\exp(\Psi(s_t, a_t; \theta^k) \top w^k)}{\sum_a \exp(\Psi(s_t, a_t; \theta^k) \top w^k)}. \tag{8}
\]

Importantly, Eqn. (8) reflects the fact that along a trajectory \( \tau \), the successor features are a function of the state and action at each time-step, \( s_t \) and \( a_t \), while the preferences \( w^k \) are learnable but consistent across time and trajectories.

A sparsity prior, i.e., \( \mathcal{L}_1 \) loss, on preferences \( w^k \) is also used to promote disentangled cumulant dimensions, see Figure 4.

**Inverse temporal difference loss.** The cumulant parameters \( \theta_\Phi \) are trained to be TD-consistent with all agents’
successor features. This procedure inverts the standard TD-learning framework for SFs (Dayan, 1993; Barreto et al., 2017) where they are trained to be consistent with a fixed cumulant \( \Phi \). Instead, we first train the SFs and preference vectors to ‘explain’ the other agents’ behaviour with the behavioural cloning loss Eqn. (8), and then train \( \theta_\Psi \) to be consistent with these successor features by minimising

\[
L_{\text{TD-}\Psi}(\theta_\Psi) \triangleq \mathbb{E}_{(s_t, a_t, s_{t+1}, a_{t+1}, k) \sim D} \left[ \Phi(s_t, a_t; \theta_\Psi) \leftarrow \gamma \Psi(s_{t+1}, a_{t+1}; \bar{\theta}_{\Psi}) - \Phi(s_t, a_t; \bar{\theta}_{\Psi}) \right] . \tag{9}
\]

In practice, our ITD-learning algorithm alternates between minimising \( L_{\text{BC-Q}} \) for training the successor features and preferences, and \( L_{\text{TD-}\Psi} \) for training the shared cumulants, provided only with no-reward demonstrations, as illustrated in blue in Figure 1. From the definition of cumulants and preferences, we can recover the \( k \)-th demonstrator’s reward, by applying Eqn. (1), i.e., \( R^k(s, a) \approx \Phi(s, a; \theta_\Psi) \top w^k \). Our ITD algorithm returns both Q-functions that can be used for imitating a demonstrator and an explicit reward function for each agent, requiring only access to demonstrations without any online interaction with the simulator. Hence ITD-learning is an offline multi-task IRL algorithm. ITD is summarised in Algorithm 1.

Single-task setting. To gain more intuition about the ITD algorithm, consider the simpler case of performing IRL with demonstrations from a single policy. This obviates the need for a representation of preferences, so we can use \( w = 1 \). In this case \( \Psi \) is the action-value function \( Q \) and \( \Phi \) is simply the reward \( R \). Minimising (8) reduces to finding a \( Q \)-function whose softmax gives the observed policy, and minimising (9) finds a scalar reward that explains the \( Q \)-function. Our more general formulation, with cumulants \( \Phi \) in place of a scalar reward, allows us to perform ITD-learning on demonstrations from many policies, and to efficiently transfer to new tasks, as we show next.

3.2. \( \Psi\Phi \)-Learning with No-Reward Demonstrations

Now we present our main contribution, \( \Psi\Phi \)-learning, which combines our ITD inverse RL algorithm with RL and GPI, using no-reward demonstrations from other agents to accelerating the ego-agent’s learning. \( \Psi\Phi \)-learning is depicted in Figure 1 and summarised in Algorithm 1.

\( \Psi\Phi \)-learning is an off-policy algorithm based on Q-learning (Watkins & Dayan, 1992; Mnih et al., 2013). The action-value function is represented with successor features, \( \Psi^{\text{ego}} \), and preferences, \( w^{\text{ego}} \), as in Eqn. (3). The ego-agent interacts with the environment, storing its experience in a replay buffer, \( \mathcal{B} \leftarrow \mathcal{B} \cup \{(s, a, s', r^{\text{ego}})\} \). The \( \Psi\Phi \)-learner also has estimates for the cumulants, per-agent SFs and preferences obtained with ITD from the demonstrations \( \mathcal{D} \).

Reward loss. The ego-rewards, \( r^{\text{ego}} \), are used to ground the cumulants \( \Phi \) and the preferences \( w^{\text{ego}} \), via the loss

\[
L_R(\theta_\Psi, w^{\text{ego}}) \triangleq \mathbb{E}_{(s, a, r^{\text{ego}}) \sim \mathcal{B}} \left[ \Phi(s, a; \theta_\Psi) \top w^{\text{ego}} - r^{\text{ego}} \right] . \tag{10}
\]

Importantly, we share the same cumulants between the ITD-learning from other agents and the ego-learning, so that they span the joint space of reward functions. This can be also seen as a representation learning method, where by enforcing all agents, including the ego-agent, to share the same \( \Phi \), we transfer information about salient features of the environment from learning about one agent to benefit learning about all agents.

Temporal difference learning. The ego-agent’s successor features are learned using two losses. First, we train the SFs to fit the Q-values using the Bellman error

\[
L_Q(\theta_{\Psi^{\text{ego}}}) \triangleq \mathbb{E}_{(s, a, s', r^{\text{ego}}) \sim \mathcal{B}} \left[ \Psi(s, a; \theta_{\Psi^{\text{ego}}}) \top w^{\text{ego}} - r^{\text{ego}} \right] \leftarrow -\gamma \max_{a'} \Psi(s', a'; \bar{\theta}_{\Psi^{\text{ego}}}) \top w^{\text{ego}} . \tag{11}
\]

We additionally train the successor features to be self-consistent (i.e. to satisfy Equation 2) using a TD loss \( L_{\text{TD-}\Psi} \).

\[
L_{\text{TD-}\Psi}(\theta_{\Psi^{\text{ego}}}) \triangleq \mathbb{E}_{(s, a, s', r^{\text{ego}}) \sim \mathcal{B}} \left[ \Psi(s, a; \theta_{\Psi^{\text{ego}}}) - \Phi(s, a; \bar{\theta}_{\Psi}) \leftarrow -\gamma \Psi(s', a'; \theta_{\Psi^{\text{ego}}}) \right] . \tag{12}
\]

GPI behavioural policy. Provided SFs estimates for the other agents, \( \{\theta_{\Psi^k}\}_{k=1}^K \), the ego-agent \( \theta_{\Psi^{\text{ego}}} \) and inferred ego preferences, \( w^{\text{ego}} \), we adopt an action selection mechanism according to the GPI rule in Eqn. (5)

\[
\pi^{\text{ego}}(s) = \arg \max_a \max_{\theta_{\Psi}} \Psi(s, a; \theta_{\Psi}) \top w^{\text{ego}} . \tag{13}
\]

The GPI step lets the agent estimate the value of the demonstration policies on its current task, and then copy the policy that it predicts will be most useful. We combat model overestimation by acting pessimistically with regard to an ensemble of two successor features approximators. If the agent’s estimated values are accurate and the demonstration policies are useful for the ego task, the GPI policy can obtain good performance faster than policy iteration with only the ego value function. The next section quantifies this claim.

Performance Bound. Given a set of demonstration task vectors \( \{w_k\}_{k=1}^K \) and successor features for the corresponding optimal policies \( \{\Psi^{\text{ego}}_k\}_{k=1}^K \), Barreto et al. (2017) show that it is possible to bound the performance of the GPI policy on the ego-agent task \( w' \) as a function of the distance of \( w' \) from the closest demonstration task \( w_j \), and the
value approximation error of the predicted value functions $Q^\pi_s = \Psi_j w^j$. We extend this result to explicitly account for the reward approximation error $\delta_r$ obtained by the learned cumulants and the SF approximation error $\delta_\Psi$.

**Theorem 1.** (Informal statement.)

Let $\pi^*$ be the optimal policy for the ego task $w'$ and let $\pi$ be the GPI policy obtained from $\{Q^\pi\}$, with $\delta_r$, $\delta_\Psi$ the reward and successor feature approximation errors. Then for all $s, a$

$$Q^* (s, a) - Q^\pi (s, a) \leq \frac{2}{1 - \gamma} \left( \phi_{max} \min_j \| w^j - w_j \| \leftrightarrow \right.$$

$$\left. + 2 \delta_r + \| w' \| \delta_\Psi + \frac{\delta_r}{1 - \gamma} \right). \quad (14)$$

**Proof.** See Appendix C for a formal statement. \qed

In settings where the agent has a good reward and SFs approximation, and the ego task vector $w'$ is close to the demonstration tasks, Theorem 1 says that the ego-agent will attain near-optimal performance from the start of training.

### 4. Experiments

We conduct a series of experiments to determine how well $\Psi_\Phi$-learning functions as an RL, IRL, imitation learning, and transfer learning algorithm.

**Baselines.** We benchmark against the following methods:

- **DQN:** Deep Q-learning (Mnih et al., 2013).
- **BC:** Behaviour Cloning, a simple imitation learning method in which we learn $p(a|s)$ via supervised learning on the demonstration data.
- **DQN+BC-AUX:** DQN with an additional behavior-cloning auxiliary loss (Hernandez-Leal et al., 2019).
- **GAIL:** Generative Adversarial Imitation Learning (Ho & Ermon, 2016), which uses a GAN-like approach to approximate the expert policy, and
- **SQILv2:** Soft Q Imitation Learning (Reddy et al., 2019), a recently proposed imitation technique that combines imitation and RL, and works in the absence of rewards. For high-dimensional environments, we replace DQN with PPO. Proximal Policy Optimization (Schulman et al., 2017). Both DQN and PPO are trained to optimize environment reward through experience, and do not have access to other agents’ experiences.

#### 4.1. Environments

Experiments are conducted using four environments, shown in Figure 2. We cover a broad range of problem settings, including both multi-agent and single-agent environments, as well as learning online during RL training, or offline from previously collected demonstrations.

**Highway** (Leurent, 2018) is a multi-agent autonomous driving environment in which the ego-agent must safely navigate around other cars and reach its goal. The other agents follow near-optimal scripted policies for various goals, depending on the scenario. In the single-task scenario, other agents have the same objective as the ego-agent, so their experience is directly relevant. In the adversarial task, the other agents do not move, and the ego-agent has to accelerate and go to a particular lane while avoiding other vehicles. Finally, in the multi-task scenario, the other agents and ego-agent have different preferences over target speed, preferred lane, and following distance. We consider the multi-task scenario to be the most realistic and representative of real highway driving with human drivers. In addition to highway driving, we also study the more complex Roundabout task. Roundabout is inherently multi-task, in that other agents randomly exit either the first or second exit, while the ego-agent must learn to take the third exit.

**CoinGrid** is a single-agent grid-world, environment containing goals of different colours. We collect offline trajectories of pre-trained agents with preferences for different goals. During training, the ego-agent is only rewarded for collecting a subset of the possible goals. We can then test how well the ego-agent is able to transfer to a goal that was never experienced during training (Section 4.5). This environment also enables learning easily interpretable preference vectors, allowing us to visualize how well our method works as an...
IRL method for inferring rewards (Section 4.3).

**FruitBot** is a high-dimensional, procedurally generated, single-agent environment from the OpenAI ProcGen (Cobbe et al., 2020) suite. We use FruitBot to test whether \( \Psi \Phi \)-learning can scale up to more complex RL environments, requiring larger deep neural network architectures that learn directly from pixels. The agent must navigate around randomly generated obstacles while collecting fruit, and avoiding other objects and walls. To create a multi-task version of FruitBot, we define additional tasks which vary agents’ preferences over collecting objects in the environment, and train PPO baselines on these task variants. The ego-agent observes the states and actions of these trained agents playing the game in parallel with its own interactions.

### 4.2. Accelerating RL with No-Reward Demonstrations

This section addresses two hypotheses: H1: When the unlabelled demonstrations are relevant, \( \Psi \Phi \)-learning can accelerate or improve performance of the ego-agent when learning with online RL; and H2: If the demonstrations are irrelevant, biased, or are generated by sub-optimal demonstrators, \( \Psi \Phi \)-learning can perform at least as well as standard RL.

Figure 3 shows the results of \( \Psi \Phi \)-learning and the baselines in the Highway and FruitBot environments. In the single-task scenario (3a), when other agents’ experience is entirely relevant to the ego-agent’s task, imitation learning methods like BC and SQILv2 learn fastest. DQN learns slowly because it does not use the other agents’ experience. However, \( \Psi \Phi \)-learning achieves competitive results, outperforming DQN+BC-Aux (Hernandez-Leal et al., 2019; Ndousse et al., 2020). In the adversarial task (3b), the other agents’ behaviors are irrelevant for the ego-agent’s task, so imitation learning (BC and SQILv2) performs poorly, while traditional RL techniques (DQN and DQN+BC-Aux) perform best. The performance of \( \Psi \Phi \)-learning does not suffer like other imitation learning methods; instead, it retains the performance of standard RL (H2). \( \Psi \Phi \)-learning can flexibly reap the benefits of either imitation learning or RL, depending on what is most beneficial for the task.

The multi-task scenario (3c) is the most realistic autonomous driving task, in which other agents navigate the highway with varying driving styles. Here, \( \Psi \Phi \)-learning clearly outperforms all other methods, suggesting it can leverage information about other agents’ preferences in order to learn the underlying task structure of the environment, accelerating performance on the ego-agent’s RL task (H1). FruitBot (3d) gives consistent results, showing that \( \Psi \Phi \)-learning scales well to high-dimensional, single-agent tasks while still outperforming BC, SQIL, PPO, and PPO+BC-Aux.

**Table 1.** We evaluate how well \( \Psi \Phi \)-learning is able to infer the correct reward function by training an RL agent on the inferred rewards, and comparing this to alternative imitation learning methods in three environments. All methods are trained on expert demonstrations. A “\( \hat{\phi} \)\)” indicates methods that infer an explicit reward function and then use one of DQN or PPO to train an RL agent, depending on the environment. A “\( \hat{\phi} \)” indicates methods that directly learn a policy from demonstrations. A “\( \tilde{\phi} \)” indicates methods that use privileged task id information for handling multi-task demonstrations. We report mean and standard error of normalised returns over 3 runs, where higher-is-better and the performance is upper bounded by 1.0, reached by the same RL agent, trained with the ground truth reward function.

| Methods   | Roundabout\( ^{SC} \) | CoinGrid\( ^{SC} \) | FruitBot\( ^{FD} \) |
|-----------|----------------|----------------|----------------|---|
| BC\( ^{\dagger} \) (Puterman, 1989) | 0.81±0.03 | 0.69±0.05 | 0.37±0.02 |
| SQIL\( ^{\dagger} \) (Reddy et al., 2019) | 0.85±0.02 | 0.64±0.05 | 0.35±0.03 |
| GAIL\( ^{\dagger} \) (Ho & Ermon, 2016) | 0.77±0.07 | 0.73±0.02 | 0.31±0.02 |
| ITD\( ^{\dagger} \) (ours, cf. Section 3.1) | 0.92±0.01 | 0.77±0.03 | 0.35±0.04 |

### 4.3. Inverse Reinforcement Learning

We now test hypothesis H3: ITD is an effective IRL method, and can accurately infer other agents’ rewards. We present a quantitative and qualitative study of the rewards for other agents that are inferred by ITD, as well as the learned cumulants and preferences. Here, we focus solely on offline IRL and use only ITD to learn from offline reward-free demonstrations, without any ego-agent experience.

To quantitatively evaluate how well ITD can infer rewards, we train an RL agent on the inferred reward function, and compare the performance to other imitation learning and RL methods. Table 1 gives the performance in terms of normalised returns on all three environments. Using ITD to infer rewards results in significantly higher performance than BC and SQIL, in two environments, and competitive performance in FruitBot. We note that unlike \( \Psi \Phi \)-learning, BC and SQIL directly learn a policy from demonstrations, and do not actually infer an explicit reward function. In contrast, GAIL does infer an explicit reward function, and ITD gives consistently higher performance than GAIL in all three environments. These results demonstrate that ITD is an effective IRL technique (H3). Qualitatively, we can evaluate how well the cumulants inferred by ITD in the CoinGrid environment span the space of possible goals. We compute the learned cumulants \( \phi(s) \) for each square \( s \) in the grid. Figure 4a shows the original CoinGrid game, and Figure 4b-4d shows the first three dimensions of the learned cumulant vector, \( \hat{\phi}_1, \hat{\phi}_2, \hat{\phi}_3 \) (the rest are given in the Appendix). We find that \( \hat{\phi}_1 \) is most active for red coins, \( \hat{\phi}_2 \) for green, and \( \hat{\phi}_3 \) for yellow. Clearly, ITD has learned cumulant features that span the space of goals for this game. See Appendix E for more details and visualisations of the learned rewards and preferences.
4.4. Imitation learning

Here, we investigate hypothesis **H4**: \( \Psi\Phi \)-learning works as an effective imitation learning method, allowing for accurate prediction of other agents’ actions. To test this hypothesis, we train other agents in the Roundabout environment, then split no-reward demonstrations from these agents into a train dataset (80%) and a held-out test dataset. We use the train dataset to run ITD, which means that we use the data to learn both \( \Phi \) and the \( \Psi \) and \( w \) for other agents. Because it is specifically designed to accurately predict other agents’ actions, we use BC as the baseline. We compare this to using only ITD, and using the full \( \Psi\Phi \)-learning algorithm including ITD and learning from RL and experience to update the shared cumulants \( \Phi \).

Accuracy in predicting other agents’ actions on the held-out test set is used to measure imitation learning performance. Figure 5 shows accuracy over the course of training. At each phase change marked in the figure, the ego-agent is given a new task, to test how the representation learning benefits from diverse ego-experience. We see that although BC obtains accurate train performance, it generalises poorly to the test set, reaching little over 80% accuracy. Without RL, \( \Psi\Phi \)-learning achieves similar performance. However, when using RL to improve imitation, \( \Psi\Phi \)-learning performs well on both the train and test set, achieving markedly higher accuracy (\( \geq 95\% \)) in predicting other agents’ behaviour. This suggests that when \( \Psi\Phi \)-learning uses RL and interaction with the world to improve the estimation of the shared cumulants \( \Phi \), this in turn improves its ability to model the \( Q \) function of other agents and predict their behaviour. Further, \( \Psi\Phi \)-learning adapts well when the agent’s goal changes, since it uses SFs to disentangle the representation of an agent’s goal from environment dynamics. Taken together, these results demonstrate that \( \Psi\Phi \)-learning also works as a competitive imitation learning method (**H4**).

4.5. Transfer and Few-Shot Generalisation

Since SFs have been shown to improve generalization and transfer in RL, here we test hypothesis **H5**: \( \Psi\Phi \)-learning will be able to generalize effectively to new tasks in a few-shot transfer setting. Using the CoinGrid environment, it
is possible to precisely test whether the ego-agent can generalise to a task it has never experienced during training. Specifically, we would like to determine whether: (i) the ego-agent can generalise to tasks it was never rewarded for during training (but which it may have seen other agents demonstrate), and (ii) the ego-agent can generalise to tasks not experienced by any agents during training.

Table 2 shows the results of transfer experiments in which agents are given 0, 1, or 100 additional training episodes to adapt to a new task. Unlike SQIL, $\Psi\Phi$-learning is able to adapt 0-shot to obtain some reward on the new tasks, and fully adapt after a single episode to achieve the maximum reward on all transfer tasks. This is because $\Psi\Phi$-learning uses SFs to disentangle preferences (goals) in the task representation, and learn about the space of possible preferences from observing other agents. To adapt to a new task, it needs only infer the correct preference vector. Task inference is trivially implemented as a least squares regression problem, see Eqn. (10): Having experienced $B^{\text{new}}$ in the new task, the $\Psi\Phi$-learner identifies the preference vector for the new task by solving $\min_{\theta_{\Psi\Phi}} \sum_{g_{\Psi\Phi}} L_{r}(\theta_{g_{\Psi\Phi}}, w)$. In contrast, SQIL requires 100 episodes to reach the same performance.

5. Related Work

Learning from demonstrations. Learning from demonstrations, also referred to as imitation learning (IL, Widrow & Smith, 1964; Pomerleau, 1989; Atkeson & Schaal, 1997), is an attractive framework for sequential decision making when reliable, expert demonstrations are available. Early work on IL assumed access to high-quality demonstrations and aimed to match the expert policy (Pomerleau, 1991; Heskes, 1998; Ng et al., 2000; Abbeel & Ng, 2004; Billard et al., 2008; Argall et al., 2009; Ziebart et al., 2008). Building on this assumption, many recent works have studied various aspects of both single-task (Ratliff et al., 2006; Wulfmeier et al., 2015; Choi & Kim, 2011; Finn et al., 2016b; Ho & Ermon, 2016; Finn et al., 2016a; Fu et al., 2017; Zhang et al., 2018; Rahmatizadeh et al., 2018) and multi-task (Dimitrakakis & Rothkopf, 2011; Mülling et al., 2013; Stulp et al., 2013; Deisenroth et al., 2014; Sharma et al., 2018; Codevilla et al., 2018; Fu et al., 2019; Rhinehart et al., 2020; Filos et al., 2020) IL. However, these methods are all limited by the performance of the demonstrator that they try to imitate. Learning from suboptimal demonstrations has been studied by Coates et al. (2008); Grollman & Billard (2011); Zheng et al. (2014); Choi et al. (2019); Shiarlis et al. (2016); Brown et al. (2019), enabling, under certain assumptions, imitation learners to surpass their demonstrators’ performance. In contrast, our method integrates demonstrations into an online reinforcement learning pipeline and can use the demonstrations to improve learning on a new task.

Reinforcement learning with demonstrations. Demonstration trajectories have been used to accelerate the learning of RL agents (Taylor et al., 2011; Vecerik et al., 2017; Rajeswaran et al., 2017; Hester et al., 2018; Gao et al., 2018; Nair et al., 2018; Paine et al., 2018, 2019), as well as demonstrations where actions and/or rewards are unknown (Borsa et al., 2017; Torabi et al., 2018; Sermanet et al., 2018; Liu et al., 2018; Aytar et al., 2018; Brown et al., 2019). In contrast to the standard imitation learning setup, these methods allow improving over the expert performance as the policy can be further fine-tuned via reinforcement learning. Offline reinforcement learning with online fine-tuning (Kalashnikov et al., 2018; Levine et al., 2020) can be framed under this settings too. Our method builds on the same principles, however, unlike these works, we do not assume that the demonstration data either come with reward annotations, or that they relate to the same task the RL agent is learning (i.e., we learn from multi-task demonstrations which may include irrelevant tasks).

Successor features. Successor features (SFs) are a generalisation of the successor representation (Dayan, 1993) for continuous state and action spaces (Barreto et al., 2017). Prior work has used SFs for (i) zero-shot transfer (Barreto et al., 2017; Borsa et al., 2018; Barreto et al., 2020); (ii) exploration (Janz et al., 2019; Machado et al., 2020); (iii) skills discovery (Machado et al., 2017; Hansen et al., 2019); (iv) hierarchical RL (Barreto et al., 2019) and theory of mind (Rabinowitz et al., 2018). Nonetheless, in all the aforementioned settings, direct access to the rewards or
cumulants was provided. Our method, instead, uses demonstrations without reward labels for inferring the cumulants and learning the corresponding SFs. More closely to this work, Lee et al. (2019) propose learning cumulants and successor features for a single-task IRL setting. Their approach differs from ours in two key respects: first, they use a learned dynamics model to learn the cumulants. Second, the learned SFs are used for representing the action-value function, not to inform the behaviour policy with GPI.

Model of others in multi-agent learning. Our method draws inspiration and builds on the multi-agent learning setting, where multiple agents participate in the same environment and the states, actions of others are observed (Davidson, 1999; Lockett et al., 2007; He et al., 2016; Jaques et al., 2019). However, we do not explore strategic settings, where recursive reasoning (Stahl, 1993; Yoshida et al., 2008) is necessary for optimal behaviour.

6. Discussion

We have presented two major algorithmic contributions. The first, ITD, is a novel and flexible offline IRL algorithm that discovers salient task-agnostic environment features in the form of cumulants, as well as learning successor features and preference vectors for each agent which provides demonstrations. The second, ΨΦ-learning, combines ITD with RL from online experience. This makes efficient use of unlabelled demonstrations to accelerate RL, and comes with theoretical worst-case performance guarantees. We showed empirically the advantages of these algorithms over various baselines: how imitation with ITD can improve RL and enable zero-shot transfer to new tasks, and how experience from online RL can help to improve imitation in turn.

Future Work. We want to explore ways to: (i) adapt ΨΦ-learning to multi-agent strategic settings, where coordination and opponent modelling (Albrecht & Stone, 2018) are essential and (ii) use a universal successor features approximator (Borsa et al., 2018) for ITD, overcoming its current, linear scaling with the number of distinct demonstrators.

Acknowledgements. We thank Pablo Samuel Castro, Anna Harutyunyan, RAIL, OATML and IRIS lab members for their helpful feedback.

References


ΨΦ-Learning: RL with Demonstrations using Successor Features and Inverse TD Learning
ΨΦ-Learning: RL with Demonstrations using Successor Features and Inverse TD Learning


A. Experimental Details

In this section we describe the environments used in our experiments (see Section 4) and the experiment design.

A.1. Highway

We build on the highway-v0 task from the highway-env traffic simulator (Leurent, 2018). The task is specified by:

1. **State space, \( S \):** The kinematic information of the ego vehicle and the five closest vehicles (ordered from closest to the furthest) is used as the Markov state, i.e., \( s_t = \{ [x_t, y_t, \dot{x}_t, \dot{y}_t]\}_{\text{ego}}, \text{other}_1, \ldots, \text{other}_5 \in \mathbb{R}^{6 \times 4} \). The ego-car is illustrated in green and the other cars in blue.

2. **Action space, \( A \):** We use a discrete action space, constructed by \( K \)-means clustering of the continuous actions of the intelligent driving model (Kesting et al., 2010). We found out that keeping 9 actions was sufficient, i.e., \( a_t \in \{0, \ldots, 8\} \).

3. **Demonstrations, \( D \):** At each time-step, the ego-car observes online the state-action pairs for the 5 closest cars.

A.2. Roundabout

We build on the roundabot-v0 task from the highway-env traffic simulator (Leurent, 2018). The task is specified by:

1. **State space, \( S \):** The kinematic information of the ego vehicle and the five closest vehicles (ordered from closest to the furthest) is used as the Markov state, i.e., \( s_t = \{ [x_t, y_t, \dot{x}_t, \dot{y}_t]\}_{\text{ego}}, \text{other}_1, \ldots, \text{other}_3 \in \mathbb{R}^{4 \times 4} \). The ego-car is illustrated in green and the other cars in blue.

2. **Action space, \( A \):** We use a discrete action space, constructed by \( K \)-means clustering of the continuous actions of the intelligent driving model (Kesting et al., 2010). We found out that keeping 6 actions was sufficient, i.e., \( a_t \in \{0, \ldots, 5\} \).

3. **Demonstrations, \( D \):** At each time-step, the ego-car observes online the state-action pairs for the 3 closest cars.

A.3. CoinGrid

We build a simple multi-task grid-world. The task is specified by:

1. **State space, \( S \):** We use a symbolic, multi-channel representation of the \( 7 \times 7 \) grid-world (Chevalier-Boisvert et al., 2019): the first three channels specify the presence or absence of the three different coloured boxes, the forth channel was the walls mask and the fifth and last channel was the position and orientation of the agent. We represent the orientation of the agent by ‘painting’ the cell in front of the agent. Therefore \( s_t \in \{0, 1\}^{7 \times 7 \times 5} \).

2. **Action space, \( A \):** We use the \{LEFT, RIGHT, FORWARD\} actions from Minigrid (Chevalier-Boisvert et al., 2018) to navigate the maze, i.e., \( a_t \in \{0, 1, 2\} \).

3. **Demonstrations, \( D \):** At the beginning of training, the agent is given state-action pairs of other agents collecting either red or green coins.

A.4. Fruitbot

We build on the Fruitbot environment from OpenAI’s ProcGen benchmark (Cobbe et al., 2020). The task is specified by:

1. **State space, \( S \):** We use the original high-dimensional \( 64 \times 64 \) RGB observations, i.e., \( s_t \in [0, 1]^{64 \times 64 \times 3} \).

2. **Action space, \( A \):** We use the original 15 discrete actions, i.e., \( a_t \in \{0, \ldots, 14\} \).

3. **Demonstrations, \( D \):** At each time-step, the agent observes online the states and actions of 3 trained agents playing the game in parallel: One agent collects both fruits and other objects, one collects other objects and avoids fruits and the last one randomly selects actions.
B. Implementation Details

For our experiments we used Python (Van Rossum & Drake Jr, 1995). We used JAX (Bradbury et al., 2018; Babuschkin et al., 2020) as the core computational library, Haiku (Hennigan et al., 2020) and Acme (Hoffman et al., 2020) for implementing $\Psi\Phi$-learning and the baselines, see Section 4. We also used Matplotlib (Hunter, 2007) for the visualisations and Weight & Biases (Biewald, 2020) for managing the experiments.

B.1. Computation Graph

Figure 10. **Computational graph of the $\Psi\Phi$-learning algorithm.** Demonstrations $D$ contain data from other agents for unknown tasks. We employ inverse temporal difference learning (ITD, see Section 3.1) to recover other agents’ successor features (SFs) and preferences. The ego-agent combines the estimated SFs of others along with its own preferences and successor features with generalised policy improvement (GPI, see Section 2.2), generating experience. Both the demonstrations and the ego-experience are used to learn the shared cumulants. Losses $L^*$ are represented with double arrows and gradients flow according to the pointed direction(s).

B.2. Neural Network Architecture

Figure 11. **Neural network architecture of the $\Psi\Phi$-learner.** The rectangular nodes are tensors parametrised by MLPs and the circles are learnable vectors. We share an observation network/torso, $E$, across all the network heads. The network heads that related to the other agents are in blue and trained from demonstrations $D$. The ego-agent’s experience $B$ is used for training the green heads. The shared cumulants and torso are trained with both $D$ and $B$. An ensemble of two successor features approximators is used for the ego- and other-agents for combatting model overestimation, see Section 3.
B.3. Hyperparameters

Table 3. ΨΦ-learner’s hyperparameters per environment. The tuning was performed on a DQN (Mnih et al., 2013) baseline with population based training (Jaderberg et al., 2017) using Weights & Biases (Biewald, 2020) integration with Ray Tune (Liaw et al., 2018). We selected the best hyperparameters configuration out of 32 trials per environment and used this for our ΨΦ-learner.

<table>
<thead>
<tr>
<th></th>
<th>Highway</th>
<th>CoinGrid</th>
<th>FruitBot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Torsion network, Ψ</td>
<td>MLP(512, 256)</td>
<td>IMPALA (Espeholt et al., 2018), shallow (no LSTM)</td>
<td>IMPALA (Espeholt et al., 2018), deep (no LSTM)</td>
</tr>
<tr>
<td>Cumulants approximator, Φ</td>
<td>MLP(128, 128)</td>
<td>MLP(256, 128)</td>
<td>MLP(256, 128)</td>
</tr>
<tr>
<td>Successor features approximator, Ψ</td>
<td>MLP(256, 128)</td>
<td>MLP(512, 256)</td>
<td>MLP(512, 256)</td>
</tr>
<tr>
<td>Ensemble size, Φ</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Λ₁ coefficient</td>
<td>0.05</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>Number of dimensions in Φ</td>
<td>8</td>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td>Minibatch size</td>
<td>512</td>
<td>64</td>
<td>32</td>
</tr>
<tr>
<td>n-step</td>
<td>4</td>
<td>8</td>
<td>128</td>
</tr>
<tr>
<td>Discount factor, γ</td>
<td>1.0</td>
<td>0.9</td>
<td>0.999</td>
</tr>
<tr>
<td>Target network update period</td>
<td>100</td>
<td>1000</td>
<td>2500</td>
</tr>
<tr>
<td>Optimiser</td>
<td>ADAM (Kingma &amp; Ba, 2014), lr=1e-3</td>
<td>ADAM (Kingma &amp; Ba, 2014), lr=1e-4</td>
<td>ADAM (Kingma &amp; Ba, 2014), lr=5e-5</td>
</tr>
</tbody>
</table>

B.4. Compute Resources

All the experiments were run on Microsoft Azure Standard_NC6s_v3 machines, i.e., with a 6-core vCPU, 112GB RAM and a single NVIDIA Tesla V100 GPU. The iteration cycle for (i) Highway experiments was 3 hours; (ii) CoinGrid experiments was 5.5 hours and (iii) Fruitbot experiments was 19 hours.
C. Proofs

We begin by formalizing the statement of Theorem 1. When not specified the norm \( \| \cdot \| \) refers to the 2-norm. Given a function \( F : \mathcal{X} \to \mathbb{R}^d \) for some finite set \( \mathcal{X} \), we will write \( F(x) \) to denote the value of the function on input \( x \) and \( F \) to denote the matrix representation of this function in \( \mathbb{R}^{|\mathcal{X}| \times d} \).

**Theorem 1** (Formal statement). Let \( C = (S, A, P, \gamma) \) be a CMP with a finite state space. Let \( \phi : \mathcal{S} \to \mathbb{R}^d \), and let \( \Phi = \phi(S) \in \mathbb{R}^{S \times d} \). Let \( (r_i)_{i=1}^k \) denote a set of reward functions on \( C \), \( \Psi_i \) be a collection of successor features approximations for policies \( (\pi_i^*)_{i=1}^k \) (\( \pi_i^* \) optimal for \( r_i \)) with true successor feature values \( \Psi_i \), and \( w_i \) the best least-squares linear approximator of \( r_i \) given \( \Phi \), with errors

\[
\| \Phi w_i - r_i \|_\infty < \delta_r \quad \text{and} \quad \| \tilde{\Psi}_i - \Psi_i \| < \delta_\Psi \quad \forall i.
\]

Let \( w' \) be a new preference vector for a reward function \( r' \), with maximal error \( \delta_r \) as well. Let \( \tilde{Q}^i = \Psi_i w' \). Let \( \pi^* \) be the optimal policy for the ego task \( w' \) and let \( \pi \) be the GPI policy obtained from \( \{ \tilde{Q}^i \} \), with \( \delta_r, \delta_\Psi \) the reward and successor feature approximation errors. Then for all \( s, a \)

\[
Q^*(s, a) - Q^\pi(s, a) \leq \frac{2}{1 - \gamma} \left[ (\phi_{\text{max}} \| w_j - w' \| + 2\delta_r) + \| \delta_r \| \| \Psi_i \| + \frac{1}{1 - \gamma} \delta_r \right]
\]

Barreto et al. (2017) construct their bound on the sub-optimality of the GPI policy as a function of the error of the value approximations \( \tilde{Q}^i \). Because we bound the reward approximation error, rather than the value approximation error, we require an additional step to obtain a bound on the errors of the value function approximations. To prove Theorem 1, we must therefore first use the following lemma to bound the effect of the reward approximation error on the value approximation error. While this result is straightforward, we include a short proof for completeness.

**Lemma 1.** Fix some policy \( \pi \). Let \( r \) be reward vector and let \( w \) be the least-squares solution to \( \min \| \Phi w - r \| \). Let \( \Psi \) be the true successor features for \( \Phi \) under policy \( \pi \), and let \( Q^\pi \) be the value. Let \( \delta_r = R(S) - \Phi w, \delta_{\text{max}} = \| \delta_r \|_\infty \). Then letting \( \tilde{Q} = \Psi w \), we have

\[
\| Q^\pi - \tilde{Q} \|_\infty \leq \frac{1}{1 - \gamma} \delta_r
\]

**Proof.**

\[
\| Q^\pi - \tilde{Q} \|_\infty \leq \sum_t \gamma^t \| P^\pi \Phi w - r \|_\infty \leq \sum_t \gamma^t \| P^\pi \delta_r \|_\infty = \sum_t \gamma^t \text{max} \sum_{s'} (P^\pi)^t (s', s) \delta_r(s) \]

Since \( P^\pi \) is a stochastic matrix, so are all of its powers, and so the rows of \( (P^\pi)^t \) sum to 1.

\[
\leq \sum_t \gamma^t \text{max} \sum_{s'} (P^\pi)^t (s', s) \delta_{\text{max}} = \sum_t \gamma^t \delta_{\text{max}} = \frac{1}{1 - \gamma} \delta_{\text{max}}
\]

We now prove the main result.
Proof. We follow the proof of Barreto et al. (2017, Theorem 2), with additional error terms to account for the reward and successor feature approximation errors.

\[
Q^*(s, a) - \bar{Q}^*(s, a) \leq Q^*(s, a) - \bar{Q}^*(s, a) + \frac{2}{1 - \gamma} \epsilon
\]  
(Barreto et al., 2017, Theorem 1)

\[
\leq \frac{2}{1 - \gamma} \| r_j - r' \|_{\infty} + \frac{2}{1 - \gamma} \epsilon
\]  
(Barreto et al., 2017, Lemma 1)

\[
\leq \frac{2}{1 - \gamma} (\phi_{\text{max}} \| w_j - w' \| + \delta_{\psi} + \delta_{r}) + \frac{2}{1 - \gamma} \epsilon
\]

\[
\leq \frac{2}{1 - \gamma} (\phi_{\text{max}} \| w_j - w' \| + 2 \delta_{\psi} + 2 \delta_{r}) + \frac{2}{1 - \gamma} \| \bar{\Psi}_j w' - \Psi_j w' + \Psi_j w' - Q_j \|
\]

\[
\leq \frac{2}{1 - \gamma} (\phi_{\text{max}} \| w_j - w' \| + 2 \delta_{\psi}) + \frac{2}{1 - \gamma} \| \bar{\Psi}_j w' - \Psi_j w' \| + \| \Psi_j w' - Q_j \|
\]

\[
\leq \frac{2}{1 - \gamma} (\phi_{\text{max}} \| w_j - w' \| + 2 \delta_{\psi}) + \frac{2}{1 - \gamma} \| w' \| \| \delta_{\psi} + \frac{2}{1 - \gamma} \| \Psi_j w' - Q_j \|
\]

\[
\leq \frac{2}{1 - \gamma} (\phi_{\text{max}} \| w_j - w' \| + 2 \delta_{\psi}) + \frac{2}{1 - \gamma} \| w' \| \| \delta_{\psi} + \frac{2}{1 - \gamma} \| \Psi_j w' - Q_j \|
\]

\[
= \frac{2}{1 - \gamma} \left[ (\phi_{\text{max}} \| w_j - w' \| + 2 \delta_{\psi}) + \| w' \| \| \delta_{\psi} + \frac{1}{(1 - \gamma)} \delta_{r} \right]
\]  
(Lemma 1)
D. Algorithms

Algorithm 1: Inverse Temporal Difference Learning

Input:
\[ D = \{ (s_1, a_1, \ldots, a_T; k)_{k=1}^N \} \]
No-reward demonstrations
\[ \lambda_w \]
\[ L_1 \] loss coefficient

Output:
\[ \theta_\Phi \]
Parameters of cumulants network
\[ \{ \theta_{\Psi k} \}_{k=1}^K \]
Parameters of successor features approximators
\[ \{ w^k \}_{k=1}^K \]
Preferences vectors for the \( K \) agents

// initialisations
1. Initialise parameters \( \theta_\Phi, \{ \theta_{\Psi k}, w^k \}_{k=1}^K \)

2. while budget do

3. Sample trajectories \( \{ \tau_i = (s_1^{(i)}, a_1^{(i)}, \ldots, s_T^{(i)}, a_T^{(i)}; k(i)) \}_{i=1}^N \sim D \)

4. Calculate behavioural cloning loss \( \mathcal{L}_{BC-Q}(\theta_{\Psi k}, w^k) \) on samples \( \{ \tau_i \}_{i=1}^N \) \( \triangleright \) see Eqn. (8)

5. \[ \theta_{\Psi k} \leftarrow \nabla_{\theta_{\Psi k}} \mathcal{L}_{BC-Q}(\theta_{\Psi k}, w^k) \] \( \triangleright \) update \( \Psi_s \)

6. \[ w^k \leftarrow \nabla_{w^k} \left( \mathcal{L}_{BC-Q}(\theta_{\Psi k}, w^k) + \lambda_w \| w^k \|_1 \right) \] \( \triangleright \) update \( w_s \)

7. Calculate inverse temporal difference loss \( \mathcal{L}_{ITD}(\theta_\Phi) \) on samples \( \{ \tau_i \}_{i=1}^N \) \( \triangleright \) see Eqn. (9)

8. \[ \theta_\Phi \leftarrow \nabla_{\theta_\Phi} \mathcal{L}_{ITD}(\theta_\Phi) \] \( \triangleright \) update \( \Phi \)
Algorithm 2: ΨΦ-Learning

Input:
\( \mathcal{D} = \{(s_1, a_1, \ldots, a_T; k)_{k=1}^{K}\} \) No-reward demonstrations
\( \lambda_w \) \( L_1 \) loss coefficient

Output:
\( \theta_{\Psi_{\text{ego}}} \) Ego successor features approximator
\( \theta_{\Phi} \) Parameters of cumulants network
\( \{\theta_{\Psi_k}\}_{k=1}^{K} \) Parameters of successor features approximators
\( \{w^k\}_{k=1}^{K} \) Preferences vectors for the \( K \) agents

// initialisations
1 Empty replay buffer for ego-experience \( B = \{\} \)
2 Initialise parameters \( \theta_{\Psi_{\text{ego}}}, w_{\text{ego}}, \theta_{\Phi}, \{\theta_{\Psi_k}, w^k\}_{k=1}^{K} \)

while budget do
  // agent-environment interaction
  3 Reset episode, \( s \leftarrow \text{env.reset}(), t \leftarrow 0 \)
  while not done do
    4 \( w^\text{ego} \leftarrow \arg \min_w \mathcal{L}_R(\theta_{\Phi}, w; B) \) \( \triangleright \) ego-task inference, see Eqn. (10)
    5 \( a \leftarrow \pi_{\text{GPI}}(s; \Psi_{\text{ego}}, w_{\text{ego}}, \{\theta_{\Psi_k}\}_{k=1}^{K}) \) \( \triangleright \) GPI, see Eqn. (13)
    6 Step in the environment, \( s', r_{\text{ego}}, \text{done} \leftarrow \text{env.step}(a) \)
    7 Append transition in the replay buffer, \( B \leftarrow B \cup (s, a, r_{\text{ego}}, s') \)
    8 \( s' \leftarrow s, t \leftarrow t + 1 \)
  // parameter updates/learning
  9 (online demonstrations) Append demonstrations in \( \mathcal{D} \) \( \triangleright \) optional
  10 \( \theta_{\Phi}, \{\theta_{\Psi_k}, w^k\}_{k=1}^{K} \leftarrow \text{ITD}(\mathcal{D}, \lambda_w, \theta_{\Phi}, \{\theta_{\Psi_k}, w^k\}_{k=1}^{K}) \) \( \triangleright \) see Algorithm. (1)
  11 Sample transitions \( \{(s^{(i)}, a^{(i)}, r_{\text{ego}}^{(i)}, s'^{(i)})\} \sim B \)
  12 Calculate the reward loss \( \mathcal{L}_R(\theta_{\Phi}, w^\text{ego}) \) \( \triangleright \) see Eqn. (10)
  13 \( \theta_{\Phi} \leftarrow \nabla_{\theta_{\Phi}} \mathcal{L}_R(\theta_{\Phi}, w^\text{ego}) \) \( \triangleright \) update \( \Phi \)
  14 Calculate TD losses \( \mathcal{L}_Q(\theta_{\Psi_{\text{ego}}}) \) and \( \mathcal{L}_{\text{TD-Ψ}}(\theta_{\Psi_{\text{ego}}}) \) \( \triangleright \) see Eqn. (11,12)
  15 \( \theta_{\Psi_{\text{ego}}} \leftarrow \nabla_{\theta_{\Psi_{\text{ego}}}} \left( \mathcal{L}_Q(\theta_{\Psi_{\text{ego}}}) + \frac{1}{\Psi} \mathcal{L}_{\text{TD-Ψ}}(\theta_{\Psi_{\text{ego}}}) \right) \) \( \triangleright \) update \( \Psi_{\text{ego}} \)
E. Visualisations

Figure 12. Qualitative evaluation of the learned cumulants in the CoinGrid task. Cumulants $\phi_1$, $\phi_2$, and $\phi_3$ seem to capture the red, green, and yellow blocks, respectively. The yellow blocks are captured by both and $\phi_4$. Therefore, linear combinations of the learned cumulants can represent arbitrary rewards in the environment, which involve stepping on the coloured blocks.

Figure 13. Sensitivity of our ITD (see Section 3.1) and $\Psi\Phi$-learning (see Section 3.2) algorithms to the dimensionality of the learned cumulants. We consistently observe across all three experiments (a)-(c) that for a small number of $\Phi$ dimensions the cumulants are not expressive enough to capture the axis of variation of the different agents’ reward functions (including the ego-agent in (c)). We also note that the performance of both ITD and $\Psi\Phi$-learning is relatively robust for a medium and large number of $\Phi$ dimensions. We attribute this to the used sparsity prior, i.e., $L_1$ loss, to the preferences $w$. In our experiments we selected the smallest number of $\Phi$ dimensions that demonstrated good performance to keep the number of model parameters as small as possible (in bold in the figures and reported in Table 3).