Robust Policy Gradient with Successor Features for Transfer in Reinforcement Learning

by

Ehsan Mehralian

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Robust Policy Gradient with Successor Features for Transfer in Reinforcement Learning

Ehsan Mehralian
Master of Science
Graduate Department of Computer Science
University of Toronto
2020

Humans have a significant ability to adapt their skills and use their already gained knowledge in a new situation with different goals or rewards. Such adaptation capability is an important sign of intelligence, but current reinforcement learning agents often perform poorly in similar situations. In this work, we propose a framework to learn a generalizable policy that can efficiently adapt to an unseen task where different tasks only differ in their reward function. Our approach is based on two key components: (a) successor features, a representation scheme that makes it possible to immediately compute the value of a policy on any task, and (b) Robust policy gradient, a generalization of standard policy gradient theorem to find a generalizable policy that can work well on a set of tasks. Putting these two together leads to an approach that integrates naturally into the RL framework and can be applied to all Actor-Critic methods without the need for much change in the original algorithm implementation. We provide our approach in a firm theoretical ground and present experiments that show it successfully promotes transfer in A2C and PPO methods in a sequence of tasks in the Linear Quadratic Regulator environment.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Despite recent successes in reinforcement learning (RL) (Mnih et al., 2015; Lillicrap et al., 2016), there is still a huge gap between RL algorithms and human intelligence. RL algorithms have been able to achieve human-level performance in many tasks, especially when combined with deep neural networks (Mnih et al., 2013; Silver et al., 2016; Berner et al., 2019). However, the agent requires a fairly large amount of time to interact with the environment to learn an effective policy. One way to reduce the required number of samples is to utilize knowledge learned from one task to other related tasks. However RL agents, unlike humans, cant adapt to even a slight change in the environment which makes them unusable in real-world applications. This problem highlights the importance of the ability to efficiently adapt to the environment changes.

Humans and animals are able to adjust their learnt skills to be used in different situations. This flexibility is considered as one of the hallmarks of intelligence. For instance, consider the task of driver's navigation in a city. Drivers that know how to get from one point to another are still able to drive the car if you change their destination. They can adapt their driving skills to new scenarios because they have a representation of the world that generalizes beyond the specific source and destination. This efficient transfer of knowledge between tasks and flexibility to reuse previous skills to solve a new task is largely absent in RL agents.

Several recent papers studied the problem of RL agents getting overly specialized to the environment setup they are trained on (Whiteson et al., 2011; Zhang et al., 2018a; Machado et al., 2018). It has been shown that the RL agents suffer from the lack of generalization to even minor changes in the environment and are prone to memorizing the solution and overfitting to the specific environment setting. For example, Cobbe et al. (2019) showed that the agent trained to have nearly optimal behaviour in a video game fails to adapt to the new but highly similar levels while a human can seamlessly generalize across similar tasks.

To overcome these limitations, several frameworks have been proposed to incorporate generalization over tasks and transfer capabilities into RL agents. Each framework assumes a specific kind of transfer because transferring knowledge between two tasks is not possible if they are completely unrelated (Taylor and Stone, 2009). We focus on a specific kind of transfer where the agent is working in a fixed environment but only the reward function can change. This transfer scenario is useful to understand how RL agents transfer the knowledge about the environment shared between tasks and is flexible enough to cover some scenarios of interest. For example, one can see the reward function as the agent's preference and the

transfer as the change of this preference in different situations. As another example, the reward function can be considered as a way to define intermediate goals that are set to break down a complex task and the transfer as the change of subgoals in the environment. We will elaborate more on this in Chapter 3.

Dayan (1993)'s Successor Representation (SR) and its extension, Successor Features (SF), to include function approximation are apt for transfer in shared dynamics. SR decouples the reward specification from the expected future state occupation in the value function. This dissociation has made these representation schemes a natural choice for the knowledge to be transferred between tasks that only differ in reward function. This approach has received a lot of attention recently. Many methods have been proposed using successor features for transfer learning (Barreto et al., 2017; Borsa et al., 2019; Madarasz and Behrens, 2019; Ma et al., 2020). Our work is mainly inspired by the work of Barreto et al. (2017) which is one of the pioneer papers in this direction. They introduced SF and used it as a pivotal element to improve efficiency of their transfer learning method (we will discuss their relation to our method in the related work section).

Most of the previous methods use SFs in the value-based RL algorithms and none of them theoretically showed its application in policy-based methods. In this project, we theoretically show how we can extend the policy gradient theorem to use successor features and propose a method to use SFs in the whole family of actor-critic methods. The main idea is that using SFs to represent the critic enables us to quickly compute the critic of any new reward function which is a better estimate of initial critic after the change of task and speeds up the learning process. This method can make all actor-critic methods suitable for transfer with minimal changes in the original algorithm, without encountering any slowdown in the single task scenario. We also present a theorem that formalizes the notion of transferability of policy in similar tasks and provide a performance bound for the transferred policy before any learning has taken place that directly depends on the task's similarity.

Despite the advantages of using SFs for transfer, it also has limitations when applied to the transfer learning scenarios. The expectation of future outcomes when following a policy (SFs of the policy) directly depends on the policy that the agent is following. When we approximate SFs through the process of learning the optimal policy of a specific task, the approximated optimal policy depends on the reward function of that task. This natural dependence to the reward function makes the SFs to also implicitly depend on the reward function of the task they are learnt for. As a result, when the difference between source and target tasks increases, the SFs learnt on the source task is less suitable for the target task. Using this SFs to compute the initial critic of the target task can even result in a slower learning compared to when starting from a random critic i.e. negative transfer.

To overcome this limitation, we propose the robust policy gradient method. The idea is that instead of approximating an optimal policy that maximizes the performance metric of a specific task, we try to learn a policy that maximizes the worst performance metric for the set of tasks in an uncertainty set associated with the reward functions. This robust policy and its learnt SFs by definition can generalize better for the tasks in that uncertainty set, starting from them we can find the optimal policy of an unseen task in that set faster. Similar to the policy gradient theorem, we derive the robust policy gradient theorem that presents how we can find the gradient of robust performance metric with respect to policy parameters. This method also integrates naturally in the policy based methods in the RL framework and can be adopted in all actor critic methods with minimal change to the original algorithm.

We finally apply the proposed methods on the Advantage Actor Critic (A2C; (Mnih et al., 2016)) and Proximal Policy Optimization (PPO; (Schulman et al., 2017)) methods and empirically evaluate the

performance of proposed methods. We show how they help to speed up the solution of target tasks in transfer scenario in the Linear Quadratic Regulator (LQR) environment as a continuous control problem.

1.1 Related Works

Our work is inspired by and builds on top of a broad range of topics, including multitask RL, transfer learning, and generalization in RL. In this section, we overview the most relevant ones. For a comprehensive presentation of the subject please see (Taylor and Stone, 2009) and (Lazaric, 2012) and references therein.

General methods: There are lots of recent papers that have proposed methods for the problem of transfer learning in RL/ deep RL. Similar to our approach, some works proposed a method to train an agent that can generalize across some tasks. Among them, some methods introduced a new architecture to leverage prior knowledge and promote transfer. For example, Rusu et al. (2016) and Kirkpatrick et al. (2016) introduced new neural network architectures well-suited for continual learning based on sequential learning that tries to overcome catastrophic forgetting to retain previously learned skills. Teh et al. (2017) proposed a method to capture common behaviour among tasks by proposing a new objective that penalizes tasks policies to be different from the shared policy as a regularizer. So the knowledge gained in one task is distilled in the shared policy and then transferred to the other policies. Finn et al. (2017) used meta-learning to find a policy that can adapt easily instead of having high performance. This policy then is used as the initial policy that can be fine-tuned effectively to speed up learning in transfer scenario. None of these methods uses a robust policy search framework similar to ours. We provided a principled way to learn a policy robust to the change of reward function that naturally integrates into the RL framework.

SF based methods: Another line of research is the attempts to use SF for transfer as it allows to immediately compute the value of a policy π on any task. Note that general value functions used in the Horde architecture (Sutton et al., 2011) can be seen as the more generalized version of SF. They also calculate several value functions concurrently each of them associated with different pseudo-reward functions. Schaul et al. (2015) extended Hord architecture to the universal value function approximators (UVFAs) that takes the embedding of goal as input and approximates its value function over state space. The idea is to augment the standard value function with an extra argument at the description of the task so it generalizes not just over state space but also different goals. They first learn goal-specific value functions and then through matrix factorization try to learn flexible goal and state embeddings. Barreto et al. (2017) proposed a framework based on two ideas of successor features (SFs) and generalized policy improvement (GPI). GPI is a generalisation of the policy improvement method that improves the policy based on a set of value functions rather than on a single one. They use SFs to compute the value of the previous policy on the task at hand – the value functions of the previous policies under the new tasks reward function. In this case, applying GPI to the set of action – value functions will result in a policy that performs at least as well as any of the previous policies.

Universal SF based methods: More recent work from Ma et al. (2020) and Borsa et al. (2019) try to combine the idea of universal value functions and SF to learn an approximate universal SF, similar

to UVFAs. Ma et al. (2020) proposed the Universal Successor Features (USFs) model, which learns SFs that can generalize over goals in an end to end model. The idea is that universal SFs can capture the generalizable knowledge about the underlying dynamics of different tasks with shared dynamics. Then using this USFs provides a good initial point to start learning a specific task in the transfer scenario. This method assumes a smooth change of SFs when there is a smooth change in the goals representations and relies on the neural network to implicitly learn this structure. Borsa et al. (2019) proposed the Universal Successor Feature Approximators (USFAs) model that imposes the universality of UVFS into SFs to be used in service of GPI. USFAs estimate SFs over multiple policies. Then applying GPI using those estimations provides a superior zero-shot policy in an unseen task ((Borsa et al., 2019)).

1.2 Contributions

In this thesis, we study the problem of transfer in reinforcement learning and in particular focus on a specific instance of transfer learning problem where the tasks only differ in the reward function. The main contributions of this work are as follows:

- A theoretical result showing how one can compute the gradient of performance metric with respect to policy parameters when the value function is represented with SFs.
- A method to use SFs for transfer in actor-critic methods that can help to speed up the solution in transfer learning.
- An upper bound on the performance of the transferred policy before any learning that directly depends on the similarity of tasks reward functions.
- A method to find a policy robust to the change of reward function that can be applied to the whole family of actor-critic methods.
- Empirical results demonstrating the effectiveness of proposed methods applied on A2C method in a continuous control task.

1.2.1 Outline

This thesis consists of five chapters. In Chapter 2 we briefly discuss basic notions from reinforcement learning that we use throughout the thesis. Chapter 3 provides theoretical results showing how we can combine SFs in actor critic methods and intorduces the robust policy gradient method. In chapter 4, we provide extensive empirical analyses of the proposed method. Finally, Chapter 5 concludes the thesis.

Chapter 2

Background

This chapter introduces and describes basic concepts of reinforcement learning and some building blocks which will be useful for understanding the rest of this document. We begin with the reinforcement learning problem formulation and Markov decision processes, then move on to the definition of policy and value function. We then review basic reinforcement learning methods and briefly describe policy-based algorithms. Finally, in Section 2.3, we describe the main idea of successor representation and its generalization to successor features.

2.1 Reinforcement Learning

Reinforcement learning (RL) is learning by interacting with an environment (Sutton and Barto, 2018). This area of machine learning deals with sequential decision-making problems. An RL problem can be described as an agent which learns from the consequences of its decisions. This agent is a discrete time stochastic control process where it interacts with its environment. As shown in Figure 2.1, at every step of interaction, the agent sees an observation of the state of the world and then decides on an action to take. It follows three consequences: (i) the agent obtains a reward signal from the environment, (ii) the state transitions to a next state, and (iii) the agent obtains an observation of the new state (François-Lavet et al., 2018).

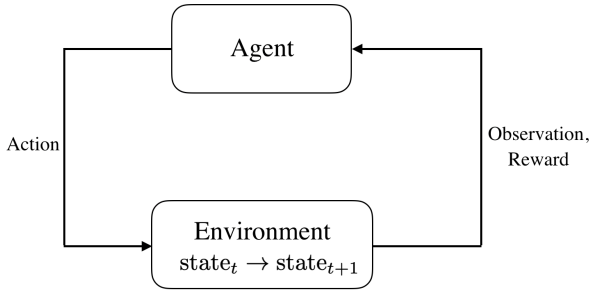

Figure 2.1: The reinforcement learning loop.

The goal of RL framework is to find a mapping between situations and decisions for this agent so that the suggested decisions maximize some measure of the long-term future reward. Here, we review

the main elements of RL before delving into RL methods.

2.1.1 Markov Decision Process

The common approach to formalize a decision-making problem in the RL framework is to model it with a Markov Decision Process (MDP). The following presentation adopts most of the notation from Szepesvári (2010).

Definition 2.1.1. A finite-action discounted Markov Decision Process (MDP) is a tuple $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ where $\mathcal{X}$ is a non-empty countable set of states, $\mathcal{A} = \{a_1, a_2, ...a_{|A|}\}$ is the finite set of $|A|$ actions, The function $\mathcal{P}$ defines the dynamics of the MDP: specifically, $\mathcal{P}(.|x, a)$ gives the next-state distribution upon taking action a in state x. $\mathcal{R}(\cdot|x, a, x')$ gives the correspanding distribution of immediate rewards, where the random variable $R(x, a, x') \sim \mathcal{R}(\cdot|x, a, x')$ determines the reward received in the transition from state x to state x' while taking action a and $\gamma \in [0, 1)$ is the discount factor that gives smaller weights to future rewards.

The key property of MDPs is that they are Markovian. The current state and action capture all relevant information from the history. The state and action are sufficient statistics of the future. In simple terms, it means the probability of each possible value for X_t and R_t depends only on the immediately preceding state and action, X_{t-1} and A_{t-1}, and given them, not at all on earlier states and actions.

2.1.2 Policy and Value function

As mentioned earlier, the goal of RL framework is to maximize some measure of the long term future reward. The rewards the agent can expect to receive in the future depend on what actions it will take. A policy defines how an agent selects actions. To find the optimal policy of some MDP, it is often useful to estimate how good it is to be in a given state, called value function (or how good it is to perform a given action in a given state, called action value function). The policy can either be stochastic, denoted by $\pi(a|x)$, or deterministic $a = \pi(x)$. Here we only consider the general case of stochastic policy. It is formally defined as:

Definition 2.1.2. A policy $\pi : \mathcal{X} \times \mathcal{A} \to [0, 1]$, is a distribution over actions given states.

$$\pi(a|x) = \mathbb{P}[A = a|X = x].$$

That is if the agent is following the policy π, then $\pi(a|x)$ is the probability of taking action a in state x.

The action a alters the state of the agent and its environment according to the transition probability function $\mathcal{P}(x_{t+1}|x_t, a_t)$. Jointly, the states and actions of the agent form a trajectory $\tau = (x_0, a_0, x_1, a_1, ...)$. In this project, we only consider the infinite horizon, episodic MDPs. A common measure in the infinite-horizon case that is used as the performance metric of the agent is the discounted return. The discounted return of trajectory τ is defined as:

$$G(\tau) = \sum_{t=0}^{\infty} \gamma^t R(x_t, a_t, x_{t+1}).$$

We can formally define the expected return (also called value function) V^π and action-value function Q^π of an underlying policy π as follows: (Szepesvári, 2010)

Definition 2.1.3. Let $(R_t; t \geq 1)$ be the sequence of rewards when starting from a state X_0 (or (X_0, A_0) for the action-value function) drawn from a positive probability distribution over $\mathcal{X}$ (or $\mathcal{X} \times \mathcal{A}$) and following the policy π for $t \geq 0$ (or $t \geq 1$ for the action-value function). Then,

$$V^\pi(x) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | X_0 = x\right], x \in \mathcal{X},$$

$$Q^\pi(x, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | X_0 = x, A_0 = a\right], x \in \mathcal{X}, a \in \mathcal{A}.$$

From the definition of the value function, the optimal value function that gives the highest achievable expected return when the starting from state x can be defined as:

$$V^*(x) = \max_\pi V^\pi(x).$$

Similar to $V^*(x)$, if the action-value function yields the maximum expected return for every state-action pair, it is said to be optimal and is denoted by Q^*:

$$Q^*(x, a) = \max_\pi Q^\pi(x, a).$$

The particularity of the action value function, Q, as compared to the state value function, V, is that the optimal policy can be obtained directly (without the need to know transition probability function, $\mathcal{P}$) from $Q^*(s, a)$:

$$\pi^*(x) = \arg\max_{a \in \mathcal{A}} Q^*(x, a).$$

2.1.3 Bellman Equation

With the help of the Bellman equation, we can compute the value function of a state recursively in terms of the value function of future states. Bellman equation decomposes the value function in two parts, the expected immediate reward and the discounted value of successor states. We can define the Bellman equation as follows:

$$V^\pi(x) = \mathbb{E}[R_{t+1} + \gamma V^\pi(X_{t+1}) | X_t = x].$$

2.2 RL Methods

The goal of reinforcement learning is to compute a policy directly from interactions between the agent and its environment such that an agent following this policy will have the maximum possible performance.

There are many ways to categorize RL methods. One fundamental split is that of model-based vs. model-free methods (Sutton and Barto, 2018). Model-based methods seek to optimize returns by learning the transition and reward models, then performing some form of dynamic programming to optimize behaviour. Model-free methods, on the other hand, attempt to learn a policy directly, without explicitly modeling the environment dynamics. This work focuses on model-free methods.

Two main approaches to learn agents with model-free reinforcement learning are value-based and policy optimization methods.

The key idea of value-based methods is that optimal actions are those which result in the highest value. All we need to do to find the optimal policy is to estimate optimal action values, $Q^*(x, a)$, i.e how good it is to take an action at a particular state. Then the solution is the greedy policy that selects actions with the highest estimated value.

On the other hand, policy optimization methods find a parameterized policy by optimizing a performance objective (typically the expected cumulative reward) without the need to compute a value function. A value function may still be used to learn the policy parameter, but is not required for action selection. One class of policy optimization methods, among others like evolution strategies (Heidrich-Meisner and Igel, 2008), is policy gradient methods. This class of algorithms directly optimize performance objective by performing stochastic gradient ascent on the parameters θ of a family of policies π_θ. The focus of this project is on this class of algorithms. In the next section we briefly describe the basic aspects of this methods.

2.2.1 Policy Gradient Methods

The idea of policy gradient reinforcement learning (Sutton et al., 1999) is to directly approximate a stochastic policy by operating in the parameter space of a parametrized policy. Policy gradient methods use stochastic gradient ascent for maximizing the performance (objective) function of the corresponding policy.

The performance function is defined as:

$$J(\theta) = V^{\pi_\theta}(x_0) = \sum_{a \in \mathcal{A}} \pi_\theta(a|x_0) Q^{\pi_\theta}(x_0, a), \tag{2.1}$$

the value function of policy π_θ starting from state x_0. We also define $d^\pi(x) = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}\{x_t = x | x_0, \pi_\theta\}$ as the unnormalized stationary discounted future-state distribution under π (if we normalize it with $1-\gamma$ it turns to a probability distribution). It shows the weighted occupancy frequency of being in state x, after following π starting in x_0. We assume d^π exists and is independent of x_0 for all policies.

In gradient ascent, the parameter update direction is given by the gradient $\nabla_\theta J(\theta)$. It points to the direction of steepest ascent of the expected return to find the best θ for π_θ that produces the highest return. The policy gradient update is therefore given by

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta),$$

where α is a learning rate.

Now using the policy gradient theorem (Sutton et al., 1999), we can find the gradient of performance metric with respect to the policy parameter as follows:

$$\nabla_\theta J(\theta) = \sum_{x \in \mathcal{X}} d^{\pi_\theta}(x) \sum_{a \in \mathcal{A}} \nabla_\theta \pi_\theta(a|x) Q^{\pi_\theta}(x, a).$$

We can reformulate the above gradient in an expectation form as follows:

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_{x\in\mathcal{X}} d^{\pi_\theta}(x) \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|x) Q^{\pi_\theta}(x,a) \\ &= \mathbb{E}_{x\sim d^{\pi_\theta}} \left[\sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|x) Q^{\pi_\theta}(x,a) \right] \\ &= \mathbb{E}_{x\sim d^{\pi_\theta}, a\sim\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|x) Q^{\pi_\theta}(x,a) \right]. \end{aligned} \tag{2.2}$$

The policy gradient theorem can be generalized to include a comparison of the action value to an arbitrary baseline $b(x)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{x\sim d^{\pi_\theta}, a\sim\pi_\theta} \left[\left(Q^{\pi_\theta}(x,a) - b(x) \right) \nabla_\theta \log \pi_\theta(a|x) \right].$$

Because:

$$\mathbb{E}_{x\sim d^{\pi_\theta}} \left[\mathbb{E}_{a\sim\pi_\theta} [b(x) \nabla_\theta \log \pi_\theta(a|x)] \right] = \mathbb{E}_{x\sim d^{\pi_\theta}} \left[b(x) \nabla_\theta \mathbb{E}_{a\sim\pi_\theta} [\log \pi_\theta(a|x)] \right] = \mathbb{E}_{x\sim d^{\pi_\theta}} \left[b(x) \nabla_\theta 1 \right] = 0.$$

So the added baseline term dose not change the expectation of gradient, but it can significantly reduce its variance (Williams, 1992).

Now we need a way to obtain samples of this gradient to use in stochastic gradient ascent. One of the simplest ways is to use the Monte Carlo estimator also known as the REINFORCE (Williams, 1992). It takes the form:

$$\theta_{t+1} = \theta_t + \alpha (G_t - b(x_t)) \nabla_\theta \log \pi_{\theta_t}(a_t|x_t)$$

Intuitively, we sample a trajectory using our policy and move the parameters of policy in the direction that most increases the probability of repeating the action a_t on future visits to state x_t times the return from time t minus the baseline $(G_t - b(x_t))$.

Although REINFORCE with baseline method will converge asymptotically to a local maximum, like any other Monte Carlo method it tends to learn slowly. If we are able to use an estimate of state-action value function instead of the return, we can solve some of these limitations. There is a class of algorithms called actor critic which bootstrapping in temporal difference learning method solve this problem.

Actor-Critic Methods

As we discussed earlier each of the value based and policy based methods have their own advantages and disadvantages. Actor-Critics aim to take advantage of all the good stuff from both value based and policy based while eliminating all their drawbacks. The principal idea is to split the process in two parts, one for computing an action based on a state and another one to produce the Q values of the action. This leads us to Actor-Critic methods, where the critic that measures how good the action taken is and the Actor that controls how our agent behaves.

$$\nabla_\theta J(\theta) = \mathbb{E}_{x\sim d^{\pi_\theta}, a\sim\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|x) Q_u^{\pi_\theta}(x,a) \right]. \tag{2.3}$$

2.3 Successor Representations in Reinforcement learning

Almost all reinforcement learning algorithms involve the prediction of the value of being in state x or the value of taking the action a in state x. These value functions depend on of two pieces of information: (1) the expected discounted future states which will be encountered, and (2) the expected reward in those states. The first component makes up the Successor Representation (SR; (Dayan, 1993)) which comes from the dynamics of the environment and agent's policy.

The main idea behind the SR is calculating how often you expect to be in other states when starting from the state x and following the policy π. It can be seen as encoding the dynamics of the Markov chain induced by the policy π and by the environments transition probability function $\mathcal{P}$. If the agent has access to the SR, it can accurately predict the discounted accumulated value of any reward function, for any state, by simply multiplying the SR of that state in the expected immediate reward of states as follows:

$$V(x) = \sum_{x'} \psi^{\pi}(x, x')R(x'),$$

where $\psi^{\pi}(x, x')$ is the expected discounted future occupancy, i.e. SR, of state x' starting from state x and $R(x')$ is the expected reward received in state x'.

The Successor Representation is defined analogously to the value function; instead of accumulating rewards (as in the value function), the SR accumulates state occupancies. Since the SR captures the visitation of successor states, it is directly dependent on the policy π and the transition dynamics $\mathcal{P}(x_{t+1}|x_t, a_t)$. More concretely, the SR with respect to a policy π, ψ^{π}, is defined as follows:[1]

$$\psi^{\pi}(x, x') = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}\{X_t = x'\}|X_0 = x\right], \tag{2.4}$$

where $\mathbb{I}$ denoting the indicator function. So for ψ^{π} we have:

$$\psi^{\pi}(x, x') = \mathbb{E}\left[\mathbb{I}\{X_0 = x'\} + \gamma\psi^{\pi}(X_{t+1}, x')|X_0 = x\right]. \tag{2.5}$$

That is, SR satisfies a Bellman equation in which $\mathbb{I}$ play the role of rewards. Therefor in practice any RL method can be used to compute ψ^{π}. For example we can use TD learning as follows:

$$\hat{\psi}^{\pi}(X_t, j) \leftarrow \hat{\psi}^{\pi}(X_t, j) + \alpha(\mathbb{I}\{X_t = j\} + \gamma\hat{\psi}^{\pi}(X_{t+1}, j)) - \hat{\psi}^{\pi}(X_t, j)), \tag{2.6}$$

for all $j \in \mathcal{X}$, where $\hat{\psi}$ is the estimate of SR being learnt following policy π and α denoting the step size.

2.3.1 Successor Features

Successor features (SF; Barreto et al. (2017)) generalize the successor representation to the function approximation setting as follows:

[1] Unlike the standard notation in RL that we typically describe the return as predicting the signal from t + 1 onward, Dayan describes the SR as predicting future state visitation from time t onward.

Definition 4. (Successor Features). For a given $0 \leq \gamma < 1$, the policy π, and for a feature representation $\phi(x) \in \mathbb{R}^d$, the successor features for a state x are:

$$\psi^{\pi}(x) = \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t \phi(X_t) | X_0 = x\big], x \in \mathcal{X}. \tag{2.7}$$

We can also extend the definition of successor features from being based on state representation $\phi(x)$, to state-action representation $\phi(x, a)$. In this case we can define successor features of state-action (x, a), as well as state x as follows:

$$\psi^{\pi}(x) = \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t \phi(X_t, A_t) | X_0 = x\big], x \in \mathcal{X},$$

$$\psi^{\pi}(x, a) = \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t \phi(X_t, A_t) | X_0 = x, A_0 = a\big], x \in \mathcal{X}, a \in \mathcal{A}.$$

Note that we overload the function $\psi^{\pi}(x)$ with $\psi^{\pi}(x, a)$. The relation between $\psi^{\pi}(x)$ and $\psi^{\pi}(x, a)$ is as follows:

$$\psi^{\pi}(x) = \mathbb{E}_{A \sim \pi(\cdot|x)}[\psi(x, A)]$$

.

As mentioned in Chapter 1, in this project we are interested in the multitask scenario, where different tasks only differ in their reward function. Now we present a simple reward model and show how it leads to the successor features.

Suppose that the reward function associated with taking action a in state x can be computed as:

$$R(x, a) = R(x, a; w) = \phi(x, a)^{\top} w, \tag{2.8}$$

where $\phi \in \mathbb{R}^d$ are the features of (x, a) and $w \in \mathbb{R}^d$ are weights. SFs allow one to immediately compute the value of a policy π on any task with weight vector w. Having (2.7), let $\phi_t = \phi(X_t, A_t)$, by simply rewriting the definition of the value function in Definition (2.1.3) we have:

$$\begin{aligned} V^{\pi}(x) &= \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | X_0 = x] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | X_0 = x] \\ &= \mathbb{E}[\phi_{t+1}^{\top} w + \gamma \phi_{t+2}^{\top} w + \gamma^2 \phi_{t+3}^{\top} w + ... | X_0 = x] \\ &= \mathbb{E}[\sum_{i=t}^{\infty} \gamma^{i-t} \phi_{i+1} | X_0 = x]^{\top} w \\ &= \psi^{\pi}(x)^{\top} w. \end{aligned} \tag{2.9}$$

As one can see, SFs decouple the dynamics of the MDP from its rewards. One benefit of doing so is that if we replace w with w', the weight vector associate with the reward function of a new task, we immediately obtain the value function of π on the new task using (2.9).

Chapter 3

Robust Policy Gradient Methods Using Successor Features

Policy gradient methods assume a parametric policy space and try to find the parameter that maximizes the performance by performing gradient ascent on the policy parameters. These methods have shown considerable success in solving high-dimensional sequential decision making tasks (Mordatch et al., 2015; Schulman et al., 2016; Rajeswaran et al., 2017). However, they are expensive in terms of their sample complexity. Learning a policy for even relatively simple tasks can require millions of steps of data collection (Schulman et al., 2016). The need for sample efficiency is even more important for agents in real-world environments. Transferring the knowledge gained while solving one task to another related, but different task is one approach to mitigate this problem (Taylor and Stone, 2009).

Despite the importance of transfer in reinforcement learning, one of the factors that makes transfer difficult or impossible in RL and prevents the widespread use of this approach is the over specialization of a policy learnt for one task. RL policies trained within a specific environment tend to overfit to the task, and cannot generalize even to small changes in the environment or task domain (Zhang et al., 2018b). How to learn generalizable policies is still a subject of ongoing research. In this chapter we will explain our contribution to this research.

In this chapter, at first we show that we can reformulate the policy gradient theorem to use successor features representation scheme to make Actor-Critic methods more suitable for transfer and provide a performance guarantees for the transferred policy. We then explain the limitation of this approach for transfer based on over specialization of source task's policy. After that in Section 3.3 we propose a new objective to enforce generalization in the learnt policy which results in a regularization term added to the original policy gradient update. Finally, we discuss the wide applicability of proposed method on Actor-Critic methods and apply it on two popular algorithms of this family, Advantage Actor Critic (A2C) and Proximal Policy Optimization (PPO).

3.1 Problem Statement

In this work we are interested in the problem of Transfer in RL. The specific type of transfer that we consider is when different tasks only differ in their reward function and the remaining specification of tasks are the same. We consider the space of all tasks that have the same state space $\mathcal{X}$, action space $\mathcal{A}$,

transition probability function $\mathcal{P}$, and discount factor γ, but only the reward function R can be different. We want to answer how one can leverage the knowledge gained on the set of tasks with different reward functions, to speed up the solution of a new, unseen MDP in that set.

We now formally define this setup. Suppose $w \in \mathbb{R}^d$ and $\phi : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to \mathbb{R}^d$, where ϕ is a fixed function and any w defines a new MDP. We define

$$\mathcal{M}^{\phi}(\mathcal{X}, \mathcal{A}, \mathcal{P}, \gamma) = \{\mathcal{M}(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma) | R(x, a, x') \sim \mathcal{R}(\cdot|x, a, x'), R(x, a, x') = \phi(x, a, x')^{\top} w\}, \tag{3.1}$$

that is, $\mathcal{M}^{\phi}$ is the set of MDPs whose reward functions can be represented by a goal specific weight vector w and a feature vector ϕ that is a function of x', a, x in which x' is the state we end up being when taking action a in state x. Note that we don't have any restriction on the ϕ. When the function ϕ is expressive enough, it can represent any reward function. In this work we assume this linear decomposition of reward function holds exactly, but our solution also applies to the case that this linear decomposition in only approximately satisfies as discussed by Barreto et al. (2018).

3.2 Policy Gradient Methods Using Successor Features

Suppose that you have approximated the optimal policy for task $M_i \in \mathcal{M}^{\phi}$. Further suppose that you are presented with a new related, but different task M_j. If we can use the knowledge gained while learning task M_i in solving task M_j, we expect to be able to speed up the solution of this new task. The way that we propose to use this knowledge in policy gradient methods is by using the action-value function of policy π_i on task M_j as the initial critic when solving task M_j.

As discussed in Section 2.3, when equation (2.8) holds, SFs converts the difficult problem of computing the value function of a policy π_i on task M_j to the much simpler supervised problem of approximating w_j, then we have: $Q_j^{\pi_i}(x, a) = \psi^{\pi_i}(x, a)^{\top} w_j$. Now we need to show how we can incorporate successor features representation in the policy gradient theorem in a way that it meets two conditions. First, it naturally integrate into the algorithm of solving a task and do not create more computational burden for it. Second, it has the ability to use useful information from the previously solved tasks.

Our first theorem shows how we can compute the gradient of the performance metric with respect to the policy parameter, θ, while using successor features representation for action-value function and meets the desired conditions:

Theorem 1. Let π_θ be a policy parameterized by a vector θ. For any MDP M, the gradient of performance metric, $J(\theta)$ (equation (2.1)), with respect to the policy parameter, θ, is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{x \sim d^{\pi_\theta}, a \sim \pi_\theta}[\langle \nabla_\theta \log \pi_\theta(a|x) \psi^{\pi_\theta}(x, a), w \rangle]. \tag{3.2}$$

Proof.

$$
\begin{aligned}
\nabla_\theta \psi^{\pi_\theta}(x) &= \nabla_\theta \sum_a \pi_\theta(a|x)\psi^{\pi_\theta}(x,a), \forall x \in \mathcal{X} \\
&= \sum_a [\nabla_\theta \pi_\theta(a|x)\psi^{\pi_\theta}(x,a) + \pi_\theta(x,a)\nabla_\theta \psi^{\pi_\theta}(x,a)] \\
&= \sum_a [\pi_\theta(a|x)(\nabla_\theta \log \pi_\theta(a|x))\psi^{\pi_\theta}(x,a)] + \sum_a \left[\pi_\theta(x,a)\nabla_\theta[\phi(x,a) + \sum_{x'} \gamma \mathcal{P}(x'|x,a)\psi^{\pi_\theta}(x')]\right] \\
&= \sum_a [\pi_\theta(a|x)(\nabla_\theta \log \pi_\theta(a|x))\psi^{\pi_\theta}(x,a)] + \gamma \sum_a \pi_\theta(a|x) \sum_{x'} [\mathcal{P}(x'|x,a)\nabla_\theta \psi^{\pi_\theta}(x')] \\
&= \mathbb{E}_{a\sim\pi_\theta(\cdot|x)} [\psi^{\pi_\theta}(x,a)\nabla_\theta \log \pi_\theta(a|x)] + \gamma \mathbb{E}_{x'\sim\mathcal{P}(\cdot|x,a)\pi_\theta(a|x)}[\nabla_\theta \psi^{\pi_\theta}(x')] \\
&= \sum_x \sum_{k=0}^{\infty} \gamma^k \mathbb{P}(x \to x', k, \pi_\theta) \mathbb{E}_{a\sim\pi_\theta(\cdot|x')}[\nabla_\theta \log \pi_\theta(a|x')\psi^{\pi_\theta}(x',a)],
\end{aligned}
$$

after repeated unrolling, where $\mathbb{P}(x \to x', k, \pi_\theta)$ is the probability of transitioning from state x to state x' in k steps under policy π_θ. We can write it in form of an expectation as follows:

$$
\nabla_\theta \psi^{\pi_\theta}(x) = \mathbb{E}_{x\sim d^{\pi_\theta}, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|x)\psi^{\pi}(x,a)]. \tag{3.3}
$$

It is then immediate that:

$$
\begin{aligned}
\nabla_\theta J(\theta) = \nabla_\theta V^{\pi_\theta}(x_0) &= \nabla_\theta \langle \psi^{\pi_\theta}(x_0), w\rangle \\
&= \mathbb{E}_{x\sim d^{\pi_\theta}, a\sim\pi_\theta}[\langle \nabla_\theta \log \pi_\theta(a|x)\psi^{\pi}(x,a), w\rangle].
\end{aligned}
$$

□

This way of expressing the gradient is the same way that is used on the original policy gradient theorem (Sutton et al., 1999) and here we just extended their results to the successor features formulation. We can also use the "linearity in the first argument" property of inner product to write the gradient of performance metric in the following form:

$$
\nabla_\theta J(\theta) = \mathbb{E}_{x\sim d^{\pi_\theta}, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|x)\langle \psi^{\pi_\theta}(x,a), w\rangle]. \tag{3.4}
$$

As one can see, our theorem shows that using successor features representation in the critic doesn't make any difference on the policy update. Comparing equation (3.4) with the original policy gradient equation (equation (2.2)), the first term, $\nabla_\theta \log \pi_\theta(a|x)$, is exactly the same in both formulations which shows the direction of policy update. The last two terms, $\langle \psi^{\pi_\theta}(x,a), w\rangle$ constructed the value of taking action a in state x, while following policy π on the task associated with the weight vector w, $Q^{\pi}_{w}(x,a)$ which is again what we have in the original policy gradient formulation. We can also explain this theorem in a more intuitive way. If we consider SF as a representation scheme, using it to represent action-value function is only considered as a change of representation which shouldn't have any affect on the policy's gradient.

In this theorem, we integrated successor features in the policy gradient methods in the RL framework. This natural integration of successor features in the original formulation makes it applicable to the whole family of policy gradient methods as long as the algorithem involves learning action-values without increasing its computational complexity.

Until now, we only showed how we can get the same policy update using SFs. But why would one bother to use this new formulation if this is exactly the same as original policy gradient theorem? This is where SFs come into play. Suppose that we have learnt the policy π_i and SFs ϕ^{π_i} for task $M_i \in \mathcal{M}^\phi$ with associated weight vector w_i. Now if the reward function changes to $R_j(x,a) = \phi(x,a)^\top w_j$, associated with a new, unseen but related task $M_j \in \mathcal{M}^\phi$, we can compute the action value function of the policy π_i, on the task M_j immediately using: $Q_j^{\pi_i}(x,a) = \psi^{\pi_i^\top} w_j$. Now we can use this value as the initial critic for solving task M_j starting from policy π_i. It has two advantages: First, we can get a better estimation of critic with the help of SFs. Intuitively, the role of critic in Actor-Critic methods is to criticizes the actions made by the actor and determine whether the performance have gone better or worse than expected after selecting an action (Sutton and Barto, 2018). As you can see in equation (2.3), we need to have the critic associated with the current policy (the true critic of policy at each step). With the help of SFs we can compute the true critic of the previous task's policy that we use as the initial policy on task M_j (instead of a random critic or previous tasks critic $(Q_i^{\pi_i}(x,a))$). Second, the policy π_i is a good policy to start from as we have the following guarantee on its performance. Note that this advantage is not related to use of SFs, but using SFs makes us able to derive a bound on the policies performance.

Theorem 2. Let $M_i, M_j \in \mathcal{M}^\phi$ and let $Q_j^{\pi_i^*}$ be the action-value function of an optimal policy of M_i when executed in M_j. Suppose that the approximate action-value $\widehat{Q}_j^{\pi_i^*}$ is such that

$$|Q_j^{\pi_i^*}(x,a) - \widehat{Q}_j^{\pi_i^*}(x,a)| \leq \epsilon, \forall x \in \mathcal{X}, \forall a \in \mathcal{A}. \tag{3.5}$$

Finally let $\phi_{max} = \max_{x,a} \|\phi(x,a)\|$, where $\|\cdot\|$ is the ℓ_2-norm. Then,

$$Q_j^{\pi_j^*}(x,a) - \widehat{Q}_j^{\pi_i^*}(x,a) \leq \frac{\phi_{max}}{1-\gamma}\|w_j - w_i\| + \epsilon. \tag{3.6}$$

Proof. We start by noting that for any $x \in \mathcal{X}$ and any $a \in \mathcal{A}$ the following holds:

$$\begin{aligned} Q_j^{\pi_j^*}(x,a) - \widehat{Q}_j^{\pi_i^*}(x,a) &= Q_j^{\pi_j^*}(x,a) - Q_j^{\pi_i^*}(x,a) + Q_j^{\pi_i^*}(x,a) - \widehat{Q}_j^{\pi_i^*}(x,a) \\ &\leq |Q_j^{\pi_j^*}(x,a) - Q_j^{\pi_i^*}(x,a)| + |Q_j^{\pi_i^*}(x,a) - \widehat{Q}_j^{\pi_i^*}(x,a)|. \end{aligned} \tag{3.7}$$

By equation (3.5) the second term $(|Q_j^{\pi_i^*}(x,a) - \widehat{Q}_j^{\pi_i^*}(x,a)|)$ is bounded by ϵ. Now we need to bound $|Q_j^{\pi_j^*}(x,a) - Q_j^{\pi_i^*}(x,a)|$. Based on Lemma 1 by Barreto et al. (2017), we have:

$$|Q_j^{\pi_j^*}(x,a) - Q_j^{\pi_i^*}(x,a)| \leq \frac{1}{1-\gamma}\max_{x,a}|R_j(x,a) - R_i(x,a)|.$$

Following from there we have:

$$
\begin{aligned}
|Q_j^{\pi_j^*}(x,a) - Q_j^{\pi_i^*}(x,a)| &\leq \frac{1}{1-\gamma} \max_{x,a} |R_j(x,a) - R_i(x,a)| \\
&= \frac{1}{1-\gamma} \max_{x,a} |\phi(x,a)^\top w_j - \phi(x,a)^\top w_i| \\
&= \frac{1}{1-\gamma} \max_{x,a} |\phi(x,a)^\top (w_j - w_i)| \\
&\leq \frac{1}{1-\gamma} \max_{x,a} \|\phi(x,a)\| \|w_j - w_i\| \qquad \text{(Cauchy-Schwarzs inequality)} \\
&= \frac{\phi_{\max}}{1-\gamma} \|w_j - w_i\|. \qquad (3.8)
\end{aligned}
$$

Plugging (3.5) and (3.8) back in (3.7) we get the desired result. □

This theorem covers the case where the policy and policy's value function are not computed exactly, either because of suboptimality of the final policy in policy gradient method, the approximation in calculating the value function, or even because of the use of function approximation. These errors are compounded in ϵ in (3.5) which appears again in the final upper bound as the second term.

In this theorem we are comparing the performance of optimal policy of task M_j and the policy learnt on task M_i by comparing their action value function. As shown in (3.6), this difference is upper-bounded by two terms. The first term is of the more interest as it reflects the difference of the tasks. This term shows that the performance of optimal policy of a task on a different task is related to how much these tasks are different. It confirms the intuition that we expect the agent to work well on task M_j, if it is learnt on a similar task M_i.

3.2.1 Limitations of Using Successor Features for Transfer

With the use of Successor Features one can immediately compute the value of a policy π on any task with weight vector w. This ability along with other properties of SFs has led to the increasing usage of this representation scheme to tackle the problem of transferring knowledge between tasks when they only differ in their reward functions. While learning a SFs representation has significant benefits for transfer, it also has some fundamental limitations.

One of the limitations that has also been studied by Lehnert et al. (2017) is the dependence of SFs to the policy they are learned for. By definition, SFs are meant to tell about the features the agent sees in the future which is dependent on what policy it is following. One of the direct conclusions we can get from Theorem 2 is that the difference between optimal policies of two tasks is directly related to the difference of their reward functions. When transferring between two tasks, each task has its own optimal policy and SFs which needs to be learned. When these task have slight differences in their reward function, their optimal policy and SFs are likely to be very similar, so starting from one of them to initialize the search for the optimal policy of the other one speeds up the process. But when their reward functions are significantly different, this initialization can even slow down the learning process and impose negative transfer.

One way to tackle this problem is to learn a generalizable policy in a way to reduce its dependence to the first task's optimal policy, but still can achive a good perforamnce on that task. We propose a method to find this policy by introducing a new performance metric which encourages generalizability

of the policy. Maximizing this robust performance metric results in a policy that can be used to as the inital point to spped up the policy search for other tasks. We describe this method in the next section.

3.3 Robust Policy Gradient Using Successor Features

In this section we describe the robust policy gradient method to tackle the limitation of transfer in policy gradient method with SFs. The idea is to learn a policy that is robust to the uncertainty associated with the change of task, which in this project refers to the robustness to the change of reward function. Starting from this robust policy and its SFs, we can speed up the search for the optimal policy of a new unseen task.

This uncertainty set is associated to the set of tasks with weight vectors around the first (source) tasks weight vector, the area that we expect to see a new task, in the space of weight vectors ($\mathbb{R}^d$). We define this set by a ball centered at the first task's weight vector w_0, with an arbitrary fixed radius,

$$\|w - w_0\|_2 \leq \epsilon_0. \tag{3.9}$$

When equation (2.8) holds, that is $R_0(x', a, x) = \phi(x', a, x)^\top w_0$, we can write this bound in terms of ℓ_∞-norm in the space of reward functions as follows:

$$\begin{aligned}
\max_{x,a} |R(x,a) - R_0(x,a)| &= \max_{x,a} |\phi(x,a)^\top w - \phi(x,a)^\top w_0| \\
&= \max_{x,a} |\phi(x,a)^\top (w - w_0)| \\
&\leq \max_{x,a} \|\phi(x,a)\|_2 \|(w - w_0)\|_2,
\end{aligned} \tag{3.10}$$

where the last line is from the Cauchy-Schwarzs inequality. Plugging (3.9) in (3.10), we have:

$$\max_{x,a} |R(x,a) - R_0(x,a)| \leq \max_{x,a} \|\phi(x,a)\|_2 \epsilon_0 = \epsilon_0'. \tag{3.11}$$

That is the set contains every task with reward function R such that $\max_{x,a} |R(x,a) - R_0(x,a)| \leq \epsilon_0'$. Furthermore, as every value function is associated with a reward function, given an uncertainty set of reward functions, we get a set of value functions containing the value functions $V_w^{\pi_\theta}$, corresponding to weight vector w of each of the reward function R in the uncertainty set.

Having the uncertainty set $\|w - w_0\|_2 \leq \epsilon_0$, we want to learn a policy that tries to maximize the worst value function (robust value function), $\check{V}_{w_0,\epsilon_0}^{\pi_\theta}(x)$, in that uncertainty set. To do that, instead of trying to maximize the performance metric of a specific task, we propose a robust objective $\check{J}_{w_0,\epsilon_0}(\theta)$ which illustrates the performance of a policy for the lowest value function in that uncertainty set. We formally define the robust value function, $\check{V}_{w_0,\epsilon_0}^{\pi_\theta}(x)$, and robust performance metric, $\check{J}_{w_0,\epsilon_0}(\theta)$, as follows:

Definition 3.1 Let R_0 be the reward function of MDP M_0 and let $w_0 \in \mathbb{R}^d$ where $R_0(x,a) = \phi(x,a)^\top w_0$ and finally let $\epsilon_0 \geq 0$. Then,

$$\begin{aligned}
\check{V}_{w_0,\epsilon_0}^{\pi_\theta}(x) &= \min_{\|w - w_0\|_2 \leq \epsilon_0} V_w^{\pi_\theta}(x), \\
\check{J}_{w_0,\epsilon_0}(\theta) &= \bar{V}_{R_0,\epsilon_0}^{\pi_\theta}(x_0),
\end{aligned}$$

where x_0 is a designated start state.

Now the goal is to find the parameter, θ^*, associated with the optimal policy, π^*, that maximizes this robust performance metric:

$$\check{\theta}^*_{w_0,\epsilon_0} = \arg\max_{\theta} \check{J}_{w_0,\epsilon_0} = \arg\max_{\theta} \min_{\|w-w_0\|_2 \leq \epsilon_0} V_w^{\pi_\theta}(x_0). \tag{3.12}$$

To solve this max-min optimization problem, we at first solve the inner minimization problem in close form using Lemma 1 and then approximate the solution of the outer maximization problem with the help of stochastic gradient ascent and Theorem 3.

With the help of linear decomposition of value function in terms of successor features ψ^π and weight vector w, we can solve the inner minimization problem using Lagrange multipliers method as follows:

Lemma 1. Let $x, w_0 \in \mathbb{R}^d$ be two fixed arbitrary vectors and $\epsilon_0 \geq 0$. Then:

$$\min_{\|w-w_0\|_2 \leq \epsilon_0} \langle x, w \rangle = x^\top w_0 - \epsilon_0 \|x\|_2. \tag{3.13}$$

Proof. We want to minimize $f(w) = \langle x, w \rangle$, subject to the constraint $\|w - w_0\|_2 \leq \epsilon_0$ where $w \in \mathbb{R}^d$ is the optimization variable. We can write the constraint in the following form:

$$g(w) = \|w - w_0\|_2 - \epsilon_0 \leq 0.$$

We form the Lagrangian function $L(w, \mu)$ with μ as the Lagrange multiplier as follows:

$$\begin{aligned} L(w, \mu) &= f(w) + \mu g(w) \\ &= \langle x, w \rangle + \mu(\|w - w_0\|_2 - \epsilon_0). \end{aligned}$$

The variable w^* is a solution to corresponding nonlinear minimization problem if the following KKT conditions hold:

$$\begin{aligned} \nabla_w L(w^*, \mu^*) &= 0 \\ \mu^* g(w^*) &= 0 \\ g(w^*) &\leq 0 \\ \mu^* &\geq 0. \end{aligned}$$

The first two conditions give us two equations to find (w^*, μ^*), and then we verify them to satisfy that last two conditions.

Based on the first condition, we find (w^*, μ^*), the saddle points of $L(w, \mu)$ by setting gradient of the Lagrangian function to zero and solve for w and μ:

$$\begin{aligned} \nabla_w L(w, \mu) &= \nabla_w f(w) + \mu . \nabla_w g(w) \\ &= x + \mu \nabla_w (\|w - w_0\|_2 - \epsilon_0) \\ &= x + \mu \nabla_w ((w - w_0)^T (w - w_0) - \epsilon_0) \\ &= x + \mu (w - w_0). \end{aligned}$$

Setting the gradient equal to zero we have:

$$\begin{aligned}\nabla_w L(w,\mu)=0 &\Rightarrow w-w_0=-\frac{x}{\mu}\\ &\Rightarrow w=w_0-\frac{x}{\mu}.\end{aligned} \tag{3.14}$$

The second condition results in having $\mu^*=0$ or $g(w^*)=0$. As μ is in the denominator of equation (3.14), we can't have $\mu^*=0$, so we set $g(w^*)=0$:

$$\begin{aligned}g(w^*)=0\Rightarrow\|w-w_0\|_2=\epsilon_0 &\Rightarrow\|(w_0-\frac{x}{\mu})-w_0\|_2=\epsilon_0\\ &\Rightarrow\|\frac{x}{\mu}\|_2=\epsilon_0\\ &\Rightarrow\frac{x^\top x}{\mu^2}=\epsilon_0^2\\ &\Rightarrow\mu=\pm\frac{\|x\|_2}{\epsilon_0}.\end{aligned} \tag{3.15}$$

As $\epsilon_0\geq 0$, the forth condition eliminates the negative answer. So we have $\mu^*=\frac{\|x\|_2}{\epsilon_0}$ and $w^*=w_0-\frac{x\epsilon_0}{\|x\|_2}$.

To complete the prove, we evaluate the objective function f at $w^*=w_0-\frac{x\epsilon_0}{\|x\|_2}$:

$$\begin{aligned}f(w^*)=\langle x,w^*\rangle &= x^\top(w_0-\frac{x\epsilon_0}{\|x\|_2})\\ &= x^\top w_0-\epsilon_0\|x\|_2.\end{aligned} \tag{3.16}$$

□

Now back to the main problem of approximating the optimal policy that maximizes the robust performance metric, $\check{J}_{w_0,\epsilon_0}(\theta)$, we can use Lemma 1 to convert the max min optimization problem to a simple maximization problem. Then similar to the original policy gradient theorem described in Section 2.2.1, we can use stochastic gradient ascent to directly approximate a parametrized policy π_θ that maximizes $\check{J}_{w_0,\epsilon_0}(\theta)$. To use gradient ascent algorithm, we need to find the gradient $\nabla_\theta\check{J}_{w_0,\epsilon_0}(\theta)$ which shows the direction of parameter update. Using Theorem 3, we can find this gradient as follows:

Theorem 3. Let π_θ be a policy parameterised by a vector θ. For MDP M_0 with associated reward weight vector $w_0\in\mathbb{R}^d$, the gradient of performance metric $\check{J}_{w_0,\epsilon_0}(\theta))$ with respect to the policy parameter θ, is:

$$\nabla_\theta\check{J}_{w_0,\epsilon_0}(\theta)=\mathbb{E}_{x\sim d^{\pi_\theta},a\sim\pi_\theta}[\nabla_\theta\log\pi_\theta(a|x)\langle\psi^{\pi_\theta}(x,a),w_0-\epsilon_0\frac{\psi^{\pi_\theta}(x)}{\|\psi^{\pi_\theta}(x)\|_2}\rangle]. \tag{3.17}$$

Proof. Using the successor features representation of value function for robust value function $\check{V}^{\pi_\theta}_{w_0,\epsilon_0}(x)$

$$\begin{aligned}\check{V}^{\pi_\theta}_{w_0,\epsilon_0}(x) &= \min_{\|w-w_0\|_2\leq\epsilon_0} V^{\pi_\theta}_w(x)\\ &= \min_{\|w-w_0\|_2\leq\epsilon_0}\langle\psi^\pi_w(x),w\rangle\\ &= \langle\psi^\pi_{w_0}(x),w_0\rangle-\epsilon_0\|\psi^\pi_{w_0}(x)\|_2. \qquad \text{(Lemma 1)}\end{aligned}$$

Now we can take the gradient of it as follows:

$$\begin{aligned}\nabla_\theta \check{V}^{\pi_\theta}_{w_0,\epsilon_0}(x) &= \nabla_\theta \left(\langle \psi^{\pi_\theta}_{w_0}(x), w_0\rangle - \epsilon_0 \|\psi^{\pi_\theta}_{w_0}(x)\|_2\right), \forall x \in \mathcal{X} \\ &= \langle \nabla_\theta \psi^{\pi_\theta}_{w_0}(x), w_0\rangle - \epsilon_0 \nabla_\theta \|\psi^{\pi_\theta}_{w_0}(x)\|_2. \end{aligned} \tag{3.18}$$

Our strategy will be to calculate $\nabla_\theta \psi^{\pi_\theta}_{w_0}(x)$ and $\nabla_\theta \|\psi^{\pi_\theta}_{w_0}(x)\|_2$. We start with $\nabla_\theta \|\psi^{\pi_\theta}_{w_0}(x)\|_2$.

Based on simple calculus for an arbitrary function $f_\theta(x) \in \mathbb{R}^d$ we have:

$$\nabla_\theta \langle f_\theta, f_\theta\rangle = \nabla_\theta \|f_\theta\|_2^2 = 2\|f_\theta\|_2 \nabla_\theta \|f_\theta\|_2 \Rightarrow \nabla_\theta \|f_\theta\|_2 = \frac{\nabla_\theta \|f_\theta\|_2^2}{2\|f_\theta\|_2} = \frac{\nabla_\theta \langle f_\theta, f_\theta\rangle}{2\|f_\theta\|_2} = \langle \nabla_\theta f_\theta, \frac{f_\theta}{\|f_\theta\|_2}\rangle.$$

So setting $f_\theta(x) = \psi^{\pi_\theta}_{w_0}(x)$, we can calculate the second term in equation 3.19 as follows:

$$\nabla_\theta \|\psi^{\pi_\theta}_{w_0}(x)\|_2 = \langle \nabla_\theta \psi^{\pi_\theta}_{w_0}(x), \frac{\psi^{\pi_\theta}_{w_0}(x)}{\|\psi^{\pi_\theta}_{w_0}(x)\|_2}\rangle. \tag{3.19}$$

Plugging (3.19) back in (3.18) we get:

$$\begin{aligned}\nabla_\theta \check{V}^{\pi_\theta}_{w_0,\epsilon_0}(x) &= \langle \nabla_\theta \psi^{\pi_\theta}_{w_0}(x), w_0\rangle - \epsilon_0 \langle \nabla_\theta \psi^{\pi_\theta}_{w_0}(x), \frac{\psi^{\pi_\theta}_{w_0}(x)}{\|\psi^{\pi_\theta}_{w_0}(x)\|_2}\rangle \\ &= \langle \nabla_\theta \psi^{\pi_\theta}_{w_0}(x), w_0 - \epsilon_0 \frac{\psi^{\pi_\theta}_{w_0}(x)}{\|\psi^{\pi_\theta}_{w_0}(x)\|_2}\rangle. \end{aligned} \tag{3.20}$$

Now we need to calculate $\nabla_\theta \psi^{\pi_\theta}_{w_0}(x)$. We have already calculated it in Theorem 1, equation (3.3) and it is:

$$\nabla_\theta \psi^{\pi_\theta}(x) = \mathbb{E}_{x\sim d^{\pi_\theta}, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|x) \psi^{\pi_\theta}(x,a)].$$

Plugging $\nabla_\theta \psi^{\pi_\theta}_{w_0}(x)$ in equation (3.20), we get our desired statement:

$$\begin{aligned}\nabla_\theta \bar{J}_{w_0}(\theta) &= \nabla_\theta \bar{V}^{\pi_\theta}_{w_0}(x_0) \\ &= \mathbb{E}_{x\sim d^\pi, a\sim\pi}[\langle \nabla_\theta \log \pi_\theta(a|x) \psi^{\pi_\theta}(x,a), w_0 - \epsilon_0 \frac{\psi^{\pi_\theta}_{w_0}(x)}{\|\psi^{\pi_\theta}_{w_0}(x)\|_2}\rangle]. \end{aligned}$$

□

Using this theorem, we can see how to calculate the gradient of robust performance metric with respect to policy parameters. Having this gradient, we can use any first-order optimizer like stochastic gradient ascent to directly approximate the policy that maximizes the robust performance metric.

Comparing this theorem with Theorem 1, the only difference is the $\frac{\psi^{\pi_\theta}_{w_0}(x)}{\|\psi^{\pi_\theta}_{w_0}(x)\|_2}$ term subtracted from the weight vector w_0. One way to think about this term is to consider it as a regularizer and ϵ_0 as the regularization parameter. Intuitively the role of this term is to change the task that the policy is trying to solve through changing the weight vector w_0. so at every step of policy update, it tries to find a task with the minimum value function under the current policy and then updates the policy to improve its performance on that specific task.

Based on Theorem 2, the similarity of optimal policies of two tasks decreases with the increase in the difference of their reward function. With the increase of ϵ_0, the size of the uncertainty set that

we consider in the robust performance metric increases. That is we are including the tasks with more different reward functions and as a result more different optimal policies in the uncertainty set around the main task. So we expect that with the increase of ϵ_0, the learned robust policy gets more conservative and less proper for transfer.

This theorem naturally integrates in the policy-gradient based RL approaches and can be applied on any method in this family that its gradient is linear in critic. In the next section we show how we can apply it to two common policy gradient methods, Advantage Actor Critic (A2C; (Mnih et al., 2016)) and Proximal Policy Optimization (PPO; (Schulman et al., 2017)).

3.4 Designing Transferable Policy Gradient Algorithms

In the last two sections, we proposed two methods to speed up learning the optimal policy for a new unseen task when we already had experience in a related task. As we described earlier, these methods naturally integrate in the whole family of Actor-Critic methods and can be used without a need for major change in the original algorithm. In this section we describe how one can apply the proposed methods on Advantage Actor Critic (A2C) and Proximal Policy Optimization (PPO) as two of the most popular Actor-Critic methods.

3.4.1 A2C

One of the problems of Vanilla Actor-Critic method (using action-value function, $Q^{\pi_\theta}(x,a)$, as critic), is having noisy gradients which causes unstable learning. One of the sources of this high variance in the gradient is the high variability of action-value function (Sutton and Barto, 2018). We can reduce this variability by subtracting a function, independent of action, from the action-value function. This function is called baseline. The baseline dosn't change the expectation of the update, but it can have a large effect on its variance. An intuitive choice for baseline is the state-value function $V^{\pi_\theta}(x)$. This new critic is called Advantage function and is defined as: $A^{\pi_\theta}(x,a) = Q^{\pi_\theta}(x,a) - V^{\pi_\theta}(x)$. When using Advantage function as the critic in the Actor-Critic method it is called Advantage Actor Critic (A2C). Substituting advantage function as the critic in equation (2.3), we get the A2C gradient update as follows:

$$\nabla_\theta J(\theta) = E_{x\sim d^{\pi_\theta}, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|x) A^{\pi_\theta}(x,a)\right].$$

A2C with SFs (A2C-SF): Theorem 1 can be generalized to include a comparison of the $\psi(x,a)$ to an arbitrary baseline $b(x)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{x\sim d^{\pi_\theta}, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|x)\langle \psi^{\pi_\theta}(x,a) - b(x), w\rangle]. \tag{3.21}$$

The baseline can be any function as long as it does not vary with action a. The equation remains valid because the subtracted quantity is zero:

$$\begin{aligned}\mathbb{E}_{x\sim d^{\pi_\theta}, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|x)\langle b(x), w\rangle] &= \mathbb{E}_{x\sim d^{\pi_\theta}}\left[\langle b(x), w\rangle \mathbb{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|x)]\right] \\ &= \mathbb{E}_{x\sim d^{\pi_\theta}}\left[\langle b(x), w\rangle \nabla_\theta 1\right] = 0.\end{aligned}$$

Using $\psi^{\pi_\theta}(x)$ as the baseline, we can define successor features advantage function as A^{ψ^π}:

$$A^{\psi^{\pi_\theta}}(x,a) = \psi^{\pi_\theta}(x,a) - \psi^{\pi_\theta}(x). \tag{3.22}$$

Substituting the $\psi^{\pi_\theta}(x)$ in (3.21), we have:

$$\nabla_\theta J(\theta) = \mathbb{E}_{x\sim d^{\pi_\theta}, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|x)\langle A^{\psi^{\pi_\theta}}(x,a), w\rangle]. \tag{3.23}$$

One way to approximate successor features advantage function (A^{ψ^π}) is by taking sample from the environment dynamics:

$$\begin{aligned} A^{\psi^{\pi_\theta}}(x,a) = \psi^{\pi_\theta}(x,a) - \psi^{\pi_\theta}(x) &= \mathbb{E}_{x'\sim\mathcal{P}(\cdot|x,a)}[\phi(x,a) + \gamma\psi^{\pi_\theta}(x')] - \psi^{\pi_\theta}(x) \\ &\approx \phi(x,a) + \gamma\psi^{\pi_\theta}(x') - \psi^{\pi_\theta}(x). \end{aligned} \tag{3.24}$$

Robust A2C with SFs (Robust A2C-SF): Similar to the A2C-SF algorithm, when using $\psi^\pi(s)$ as the baseline in Theorem 3, we get the Robust A2C-SF update as follows:

$$\nabla_\theta \check{J}_{w_0}(\theta) = \mathbb{E}_{x\sim d^{\pi_\theta}, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|x)\langle A^{\psi^\pi}(x,a), w_0 - \epsilon_0 \frac{\psi^{\pi_\theta}(x)}{\|\psi^{\pi_\theta}(x)\|_2}\rangle]. \tag{3.25}$$

3.4.2 PPO

To improve the data efficiency in single task reinforcement learning many algorithms have been proposed, one of them is the Proximal Policy Optimization (PPO; (Schulman et al., 2017)) algorithm . The authors proposed a new objective function that enables multiple epochs of minibatch updates instead of just one update. The key contribution of PPO is ensuring that a new update of the policy does not change it too much from the previous policy. This leads to less variance in training at the cost of some bias. The objective function of this algorithem is:

$$J^{\text{CLIP}}(\theta) = \mathbb{E}_t[\min(r_t(\theta)A_{\theta_{\text{old}}}, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_{\theta_{\text{old}}})],$$

where r_t is the ratio of the probability under the new and old policies ($r_t(\theta) = \frac{\pi_\theta(a_t|x_t)}{\pi_{\theta_{\text{old}}}(a_t|x_t)}$) and ϵ is a hyperparameter, usually 0.1 or 0.2.

PPO with SFs (PPO-SF): Using $A^{\psi^{\pi_\theta}}$ defined as:

$$A^{\psi^{\pi_{\theta_{\text{old}}}}}(x,a) = \psi^{\pi_{\theta_{\text{old}}}}(x,a) - \psi^{\pi_{\theta_{\text{old}}}}(x).$$

we can write the objective as:

$$J^{\text{CLIP}}(\theta) = \mathbb{E}_t[\min(r_t(\theta)\langle A^{\psi^{\pi_{\theta_{\text{old}}}}}, w\rangle, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\langle A^{\psi^{\pi_{\theta_{\text{old}}}}}, w\rangle)].$$

Robust PPO with SFs (Robust PPO-SF): Based on the robust performance metric definition, we can modify the original objective of PPO in the following way to maintin robustness.

$$\check{J}^{\text{CLIP}}(\theta) = \mathbb{E}_t[\min(r_t(\theta)\langle A^{\psi^{\pi_{\theta_{\text{old}}}}}, w_0 - \epsilon_0 \frac{\psi^{\pi_\theta}(x)}{\|\psi^{\pi_\theta}(x)\|_2}\rangle, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\langle A^{\psi^{\pi_{\theta_{\text{old}}}}}, w_0 - \epsilon_0 \frac{\psi^{\pi_\theta}(x)}{\|\psi^{\pi_\theta}(x)\|_2}\rangle)]. \tag{3.26}$$

Chapter 4

Empirical Evaluation

In this chapter we describe the experiments conducted to test the proposed method in a multitask setting and assess its ability to generalize to unseen tasks. We seek to answer the following questions:

- Can policy gradient with SFs successfully transfer between multiple goals and speed up learning?
- When does policy gradient with SFs fail to speed up learning?
- Can the robust policy gradient with SFs method improve the original policy gradient with SFs method and standard methods on a fixed task?

To answer these questions, we begin by training our agent on a single task (source task) to learn the policy and SFs for that specifc task. Then we switch the environment to a new task (target task) and observe how quickly and how well the agent can adapt to this new, unseen task. In practice, this means we initialize the target task's neural network parameters with the parameters of source task.

We start by describing the environments that we used for our experimetns in Section 4.1 and then in Section 4.2 we inverstigate how SFs in policy gradient methods help to trasfer knowledge between tasks. We then emperically illustrate limitations of this method and finally in Section 4.3 we design some experiments to show how robust policy gradient with SFs method tackles these limitations.

4.1 Linear Quadratic Regulator

In order to evaluate our algorithms we examine their performance in the Linear Quadratic Regulator (LQR) environment as a continuous control problem. In the LQR environment we assume a linear dynamical system such that:

$$x_{t+1} = Ax_t + Ba_t, \tag{4.1}$$

where $x_t \in \mathcal{X} \subset \mathbb{R}^d$ is the continuous state at time t and $a_t \in A \subset \mathbb{R}^k$ is the continuous control input (i.e., action) at time t. Matrix $A \in \mathbb{R}^{d\times d}$ is the system matrix which governs the evolution of the states from one time step to another, and the matrix $B \in \mathbb{R}^{d\times k}$ is the control matrix which determines how the system input affects the states and both are independent of time t.

In this environment there is an immediate cost (negative of reward) associated with being at state x_t and taking the action a_t in the quadratic form:

$$c(x_t, u_t) = x_t^\top Q x_t + a_t^\top R a_t, \tag{4.2}$$

where $Q \in \mathbb{R}^{d\times d}$ and $R \in \mathbb{R}^{k\times k}$ are symmetric, positive definite matrices ($\forall x, x^\top Q x > 0$ and $\forall a, a^\top R a > 0$) that place weights on the competing objectives based on which of the objectives we care more about.The first term indicates quadratic state cost that penalizes deviation of system's state from the all zeros state and the second term indicates quadratic control cost which penalizes high control signals.

Assume that A, B, Q, R are known. The goal in this environment is to stabilize the system around the all-zeros equilibrium state with all-zeros control input, so the system remains at zero forever.

In the discounted finite horizon LQR problem, we want to find a^* so that:

$$\begin{aligned} a^* = &\arg\min_a \sum_{t=0}^{T} \gamma^t c(x_t, a_t) \\ &\text{s.t.} \quad \forall i \in \mathbb{N}; i \leq T-1 : x_{i+1} = Ax_i + Ba_i. \end{aligned} \tag{4.3}$$

In this environment, if we want to stabilize the system around a desired point, it is enough to reframe the state of the system as the derivation from the desired state, that is $x'_t := x_t - g$ where $g \in \mathbb{R}^d$ is the desired state. In other words:

$$c(x_t, a_t) = (x_t - g)^\top Q (x_t - g) + a_t^\top R a_t. \tag{4.4}$$

In our experiemtns, we set $x, a \in \mathbb{R}^2$, used the identity matrix for B, Q, R, and chose A in a way that that the system would be stable over time. Specifically,

$$\begin{aligned} B = Q = R = I \\ A = \begin{bmatrix} 0.9 & 0 \\ 0 & 0.9 \end{bmatrix}. \end{aligned}$$

4.1.1 Linear Decomposition of LQR Cost Function

We are interested in the scenario where all components of an MDP are the same, except for the reward function. In the LQR environment it means that all tasks share same dynamical system and just differ in the equilibrium point, g.

The expected one-step cost of the LQR environment can be computed as

$$\begin{aligned} c(x_t, a_t) &= (x_t - g)^\top Q (x_t - g) + a_t^\top R a_t \\ &= x_t^\top Q x_t - 2 x_t^\top Q g + g^\top Q g + a_t^\top R a_t \\ &= \underbrace{[x_t^\top Q x_t + a_t^\top R a_t, -2 x_t^\top Q, 1]^\top}_{\phi(x_t, a_t)^\top} \underbrace{[1, g, g^\top Q g]}_{w(g)}, \end{aligned} \tag{4.5}$$

where w only depends on the equilibrium point and any w specifies a different task. As we chose $x, a \in \mathbb{R}^2$ in this project, so we have $\phi, w \in \mathbb{R}^4$. This linear decomposition of cost function allows us to use the successor features formulation to represent the cost-to-go function in exact form.

4.2 Policy Gradient with Successor Features

We start with the first question. When learning with multiple goals, can policy gradient with SFs method successfully transfer knowledge between tasks and speed up training? To answer this question, we begin by training the agent on a source task. After a while, we switch the environment to a new unseen task (target task) and observe how quickly the agent can adapt to this new task. That is we use the model (function approximators of actor and critic) parameters that the agent learned on the source task as initialization for the model on the target task.

We consider the LQR environment. The initial state is drawn uniformly from $[-1, 1]^2$. The objective is to stabilize the agent around the goal location. We choose the source task's goal, g_s, uniformly from $[-1, 1]^2$. Each episode terminates after 1000 time steps. After 300 episodes, the goal location (target task's goal, g_t) moves to a point sampled from the intersection of a Gaussian distribution with mean g_s and σ^2 variance and $[-1, 1]^2$, i.e. $N(g_s, \sigma^2) \cap [-1, 1]^2$. Roughly speaking, as g_t gets farther away from g_s, their optimal policies will be more different. Therefor as σ increases, the probability of having different source and target tasks increases, their successor features might be more different, and transferring knowledge between them gets more difficult. We choose σ to be $\{0.1, 0.5, 1.0, 2.0\}$ to consider transfer scenarios in different difficulties.

We use the ℓ_2-norm objective (equation 4.6) for the source task's training. For the target task's training, we compare two different objectives for learning the critic. In the first one, called SF2, after the change of task we will still use ℓ_2-norm objective to learn ψ^π.

$$L_\theta(X_t) = \|G_{t:t+n} - \hat{\psi}^\pi_{t+n-1}(X_t)\|_2^2, \tag{4.6}$$

where $G_{t:t+n} = \phi_t + \gamma\phi_{t+1} + ... + \gamma^{n-1}\phi_{t+n} + \gamma^n\hat{\psi}^\pi_{t+n-1}(X_{t+n})$ and $\phi_t = \phi(x_t, a_t)$.

The second version of our agent uses the inner product objective to learn ψ^π after the change of task:

$$L_\theta(X_t) = \langle G_{t:t+n} - \psi^\pi_{t+n-1}(X_t), w\rangle^2. \tag{4.7}$$

We refer to this instance of our algorithm as SF2w. Note that both of the agents use the ℓ_2-norm objective for the source task's training. We also compared our methods with the standard version of the agent which directly predicts value function instead of SFs as a baseline in our comparisons which we call them PPO-V and A2C-V.

To evaluate the performance of the above algorithms, we use the discounted return metric where the discount factor is 0.99 . We compare these methods using the same set of the goals. The results are averages of 30 runs. For all algorithms we model the policy in each state as a Gaussian distribution. We use a neural network (NN) with two hidden layers of 64 units and D units in final layer where D is the dimension of action space $(64 \times 64 \times D)$. We use hyperbolic tangent function (tanh) activation function to predict the mean and D parameters with softplus activation function $(f(x) = \ln(1 + e^x))$ to predict the standard deviation of policy distribution. We use a $(64 \times 64 \times 1)$ NN for state-value function estimation in PPO-V and A2C-V methods and a $(64 \times 64 \times d)$ NN to predict SFs in their SF2 and SF2w variation, where d is the dimension of w. For this set of experiments our action space has 2 dimension so D is equal 2 and we chose d to be 4.

For the A2C and PPO based methods we use RMSProp (Tieleman and Hinton, 2012) and Adam (Kingma and Ba, 2015) optimizers respectlively to update the parameters of networks.

Parameter	-V	-SF2	-SF2w	-RobustSF2w
A2C actor lr	0.001	0.001	0.001	0.001
A2C critic lr	0.03	0.03	0.03	0.02
PPO actor lr	0.0002	0.0002	0.0002	0.0002
PPO critic lr	0.005	0.001	0.001	0.005

Table 4.1: Value of hyper-parameters for A2C and PPO based methods. The columns refer to different variations of algorithms, V: the standard version, SF2: using (4.6) for critic loss in the target task, SF2w: using (4.7) for critic loss in the target task, RobustSF2w: using robust policy gradient in the source task and inner product as the critic update.

We find the best value for the hyper-parameters based on the time to threshold metric of source task over 10 runs, that is we choose the value that achieved a specific average performance over 10 runs faster than others. The search spaces that we choose for actor learning rate and critic learning rate are $\{0.0001, 0.0002, 0.0005, 0.001, 0.005\}$ and $\{0.0005, 0.001, 0.005, 0.01, 0.02, 0.03, 0.04, 0.05\}$, respectively. We find the best learning rates by conducting a grid search in this space. Table 4.1 shows the selected values for each parameter for different methods. Note that we also multiply the learning rates by 0.99 in each episode.

Also in the methods based on PPO algorithm we take 50 samples from the environment and update the parameters in 16 epochs with mini batches with the size of 32 while in A2C based methods we update the parameters once after taking 5 samples.

The results of our experiments can be seen in Figure 4.1. Before the change of task, in the first 300 episodes, the discounted return of all methods quickly decreases and saturates around zero. Note that SF2 and SF2w are exactly the same during the source task. SFs based A2C and PPO have comparable performance with the original A2C and PPO. Existing methods that use SFs representation are considerably slower when learning from a single task and much of the benefit comes from using learned SFs to transfer knowledge in next tasks. But we see that SF-A2C and SF-PPO methods still have a competitive result compared to A2C and PPO which shows their usefulness even in the single task scenario.

When the goal suddenly changes at episode 300, the performance in all methods decreases. With the increase of σ, the probability of having more different source and target tasks increases and as expected the drop in the performance of all methods increases. Comparing the performance of methods a few episodes after the change of task, we observe that both SF2 and SF2w methods have higher performance than the original method. The reason is that with the help of learned SFs during the first task, now we can compute the exact value function of the current policy for the target task, but in the A2C-V and PPO-V we have the value function of source task. This more accurate estimation of critic helps to achieve a higher performance in the early episodes after the change of task.

For the PPO based methods, we observe that this superiority of SFs based methods still holds over the PPO-V, while they all get slower with the increase of σ. But in A2C based methods as the learning proceeds, we observe that A2C-V beats A2C-SF2 while A2C-SF2w still converges faster than A2C-V. One possible explanation is that, the critic objective in SF2 method does not have any signal about the change of task so it cannot get out of the source's task local minima easily. This problem is more important when the policy of source task makes exploration for the target task difficult while in the case of SF2w, using the vector w which specifies the current task, helps to solve this problem, so the critic converges faster to the optimal critic and achieves faster learning rate. We don't see this problem in

the PPO-SF2, because the critic in this method gets updated many more times over multiple epochs of mini-batch updates while we update the critic in the A2C method only once after each data collection.

As described earlier, with the increase of σ^2, we only increase the chance of having farther goal states between source and target task. To study the limitations arise in this setting, we need a more definite scenario. In the next part, we will discuss this limitation using another experiment.

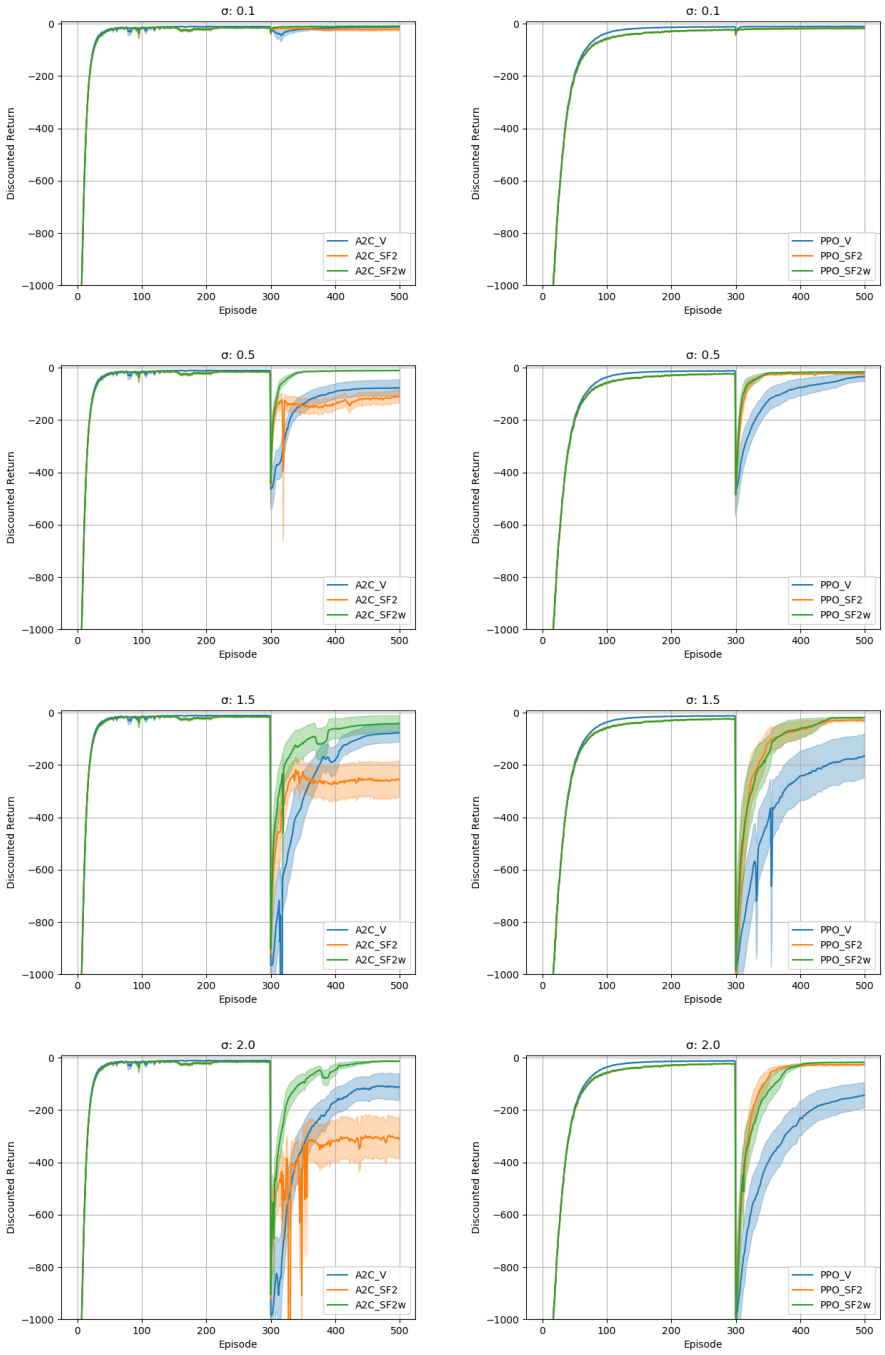

Figure 4.1: Discounted Return in the LQR domain. After a 300 episodes, the goal location is moved to $g_t \sim \mathcal{N}(g_s, \sigma^2)$. Shaded areas represent standard error over 30 runs.

4.2.1 Limitation of Successor Features in Transfer

To answer the second question about limitations of policy gradient with SFs method and in general SFs in transfer learning, we study a scenario that emphasizes this effect. We consider two cases: a) The source task's goal state is at $[0.9, 0.9]$ and the target task's goal state is at $[0.0, 0.0]$. b) The source task's goal state is at $[0.0, 0.0]$ and the target task's goal state is at $[0.9, 0.9]$. In both cases the initial state is drawn uniformly from $[-1, 1]^2$. The distance of goal states in both scenarios are the same. The only difference is the problem with exploration especially on the target task.

In the first case, after the change of task, the source task's policy still leads the agent to the upper right corner of the state space from different start states. In the second case, after the change of task, the policy forces the agent to go around the centre of state space. Given that we choose the initial state uniformly, the chance of agent to be in the upper right corner is low. So it is much more difficult for the agent to explore the new goal in the second scenario compared to this first one.

We expect that the high difference in the optimal policies of source and target tasks makes it difficult for A2C-V and PPO-V methods to adapt quickly to the target task. But In the first task as we don't have an exploration problem they should converge as fast as the source task's convergence rate but the difficult exploration problem of case b makes the adaptation much slower.

On the other hand, the SFs based methods in the first case are expected to have a better estimate of critic after the change of the task which should result in a faster transfer. But in case b they are not expected to be useful in an efficient transfer.

Figure 4.2 shows the results of this experiment. In both cases we have a significant reward change, but in case b because of the exploration problem, which is the result of using source task's policy as the initial policy of target task, the convergence rate is much slower compared to first case and even to the source task's convergence. The dependency of SFs on the source task's policy, the high difference of optimal policies in source and target tasks, and the exploration problem imposed by the target task's policy results in even slower convergence compared to the source task's convergence rate even in these methods. In general, even using these SFs can't help much to have an efficient transfer. These results show the importance of finding a generalizable policy to be used as the initial policy of target task which we will investigate in the next section.

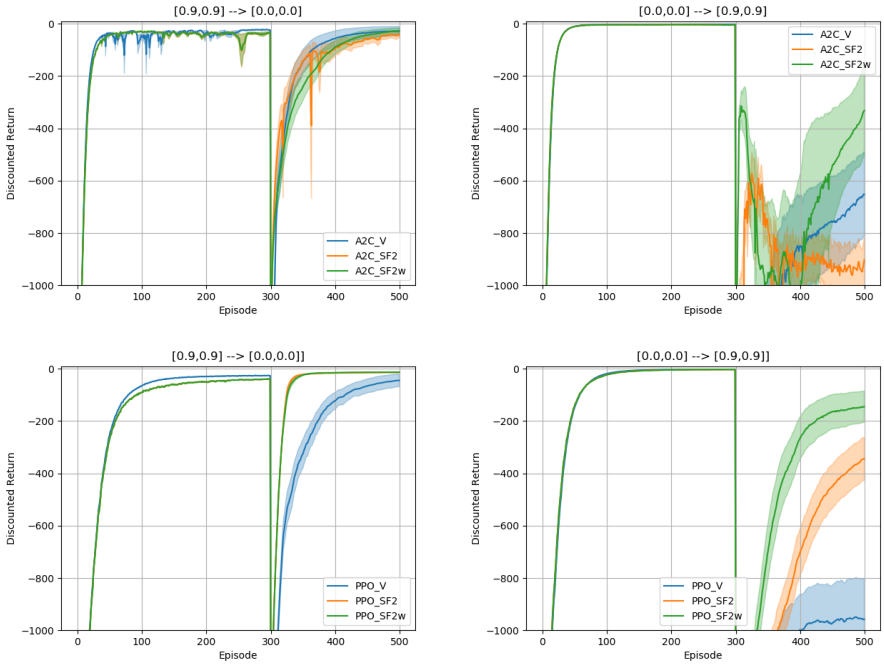

Figure 4.2: Discounted return in the LQR environment. After 300 episodes, the goal location is moved [Left]: from [0.9,0.9] to [0.0,0.0] (case a), [Right]: from [0.0,0.0] to [0.9,0.9] (case b). Shaded areas represent standard error over 30 runs.

4.3 Robust Actor Critic Using Successor Features

As we saw in the previous section, successor features depend on the policy they are trained with and even in the proposed methods, the policy only approximates the optimal behaviour for a specific task. Hence, when transferring knowledge between tasks with different optimal policies, with the increase of this difference, the learned SFs can get unusable. In this scenario even using the learned policy as the initialization of a new tasks policy can make exploration problem and even slow down the learning compared to when we start learning from a random initial point.

If we can learn a policy that performs well on a set of tasks (instead of the optimal policy of a specific task), the resulting policy can help to learn SFs that are more generalizable. Also, this generalizable policy can serve as a good initialization for fine-tuning to a more specific behaviour. Now we try to empirically evaluate the proposed Robust SF based methods designed to find this policy.

We train the RSFA2C method based on Theorem 3 in the source task, that is for the A2C-RobustSF method we use equation (3.25) and for the PPO-RobustSF method, equation (3.26). After the change of task, just like SF based methods, we use both objectives to learn critic based on equation (4.6) and (4.7), calling them RobustSF2 and RobustSF2w, respectively. We use exactly the same setting as we described in Section 4.2. The only difference is that this method has a hyper-parameter ϵ that we need to tune it.

Figure 4.3 and 4.4 shows the result of this experiment for a value of σ and three different values of ϵ. As we can see, with the increase of ϵ (which means increasing difference of tasks that the policy is trying to solve them all), the performance in the source tasks decreases. That is ϵ controls the diversity of the set of tasks that the policy is trying to maximize its performance over them. An ϵ equal to zero is equal

to not having any Robustness is equal to the simple SF based methods. A large value of ϵ means we want a policy that can work well on a set of very different tasks which results in a conservative policy with limited the learning capacity and fails to transfer efficiently.

After the change of the task, we are using the generalized policy and its successor features as the initial policy and to compute the initial critic for the target task. In both versions of RobustSF2 and RobustSF2w versions, in both of the A2C and PPO methods, results associated with the intermediate value of ϵ outperform their SFs based methods and show faster adaptation to the target task compared to the other methods and when starting from scratch. One possible explanation is that they could find the best trade-off between generalization and gathering knowledge from the source task.

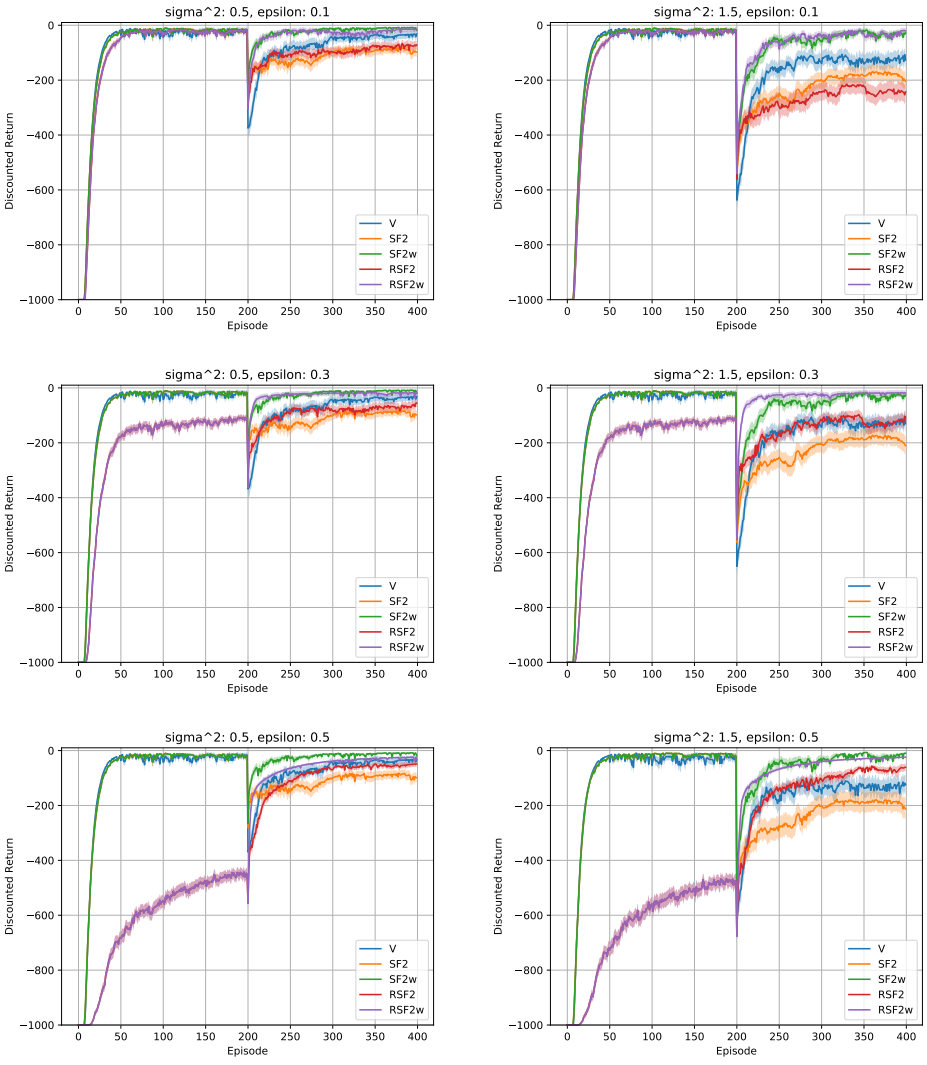

Figure 4.3: Discounted Return in the LQR domain for A2C algorithm. After a 200 episodes, the goal location is moved to [Left]:$g_t \sim \mathcal{N}(g_s, 0.5)$, [Right]:$g_t \sim \mathcal{N}(g_s, 1.5)$. Shaded areas represent standard error over 30 runs.

To study the effect of ϵ in transfer and the sensitivity of our method to value of this hyper-parameter, we consider transfer to four different target tasks with different difficulty of transfer ($g_t \in \{[0.3, 0.3], [0.5, 0.5], [0.7, 0.7], [0.9, 0.9]\}$) when the source tasks's goal is at $[0.0, 0.0]$. To compare the effect

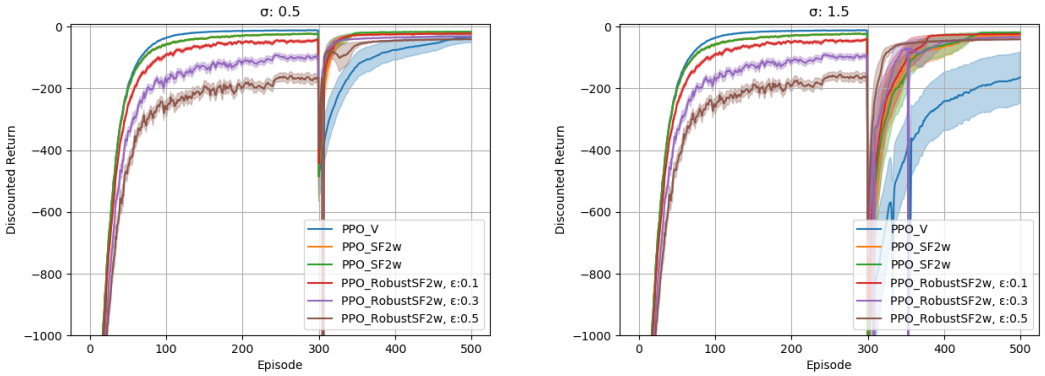

Figure 4.4: Discounted Return in the LQR domain for PPO algorithm. After a 300 episodes, the goal location is moved to [Left]:$g_t \sim \mathcal{N}(g_s, 0.5)$, [Right]:$g_t \sim \mathcal{N}(g_s, 1.5)$. Shaded areas represent standard error over 30 runs.

of different values of epsilon, we compare the area under the discounted return curve of agents trained with different ϵ in the 200 episodes of training on the target task. As a better transfer results in the discounted return curve to get close to zero faster in the LQR environment, a lower area under the training curve means faster convergence and a better transfer. Figure 4.5 shows the area under the training curve of RobustSF2w in A2C method for these four cases for different values of ϵ. As we can see, as the difference between the optimal policy of source and target tasks increases, the optimal value of ϵ increases.

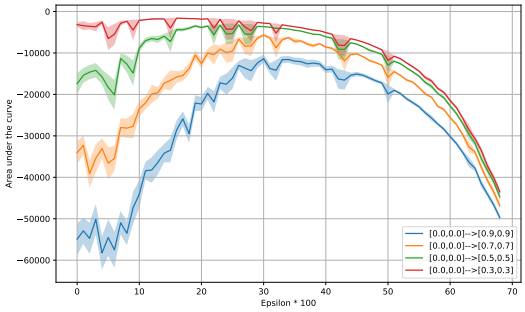

Figure 4.5: The area under the curve of different value of ϵ in different transfer settings. Shaded areas represent standard error over 30 runs. Lower absolute value of area under the curve means a faster transfer.

Chapter 5

Conclusion

In this thesis, we introduced a framework to improve the generalizability of Actor-Critic methods over tasks with different reward functions. This framework builds on two concepts. The first one is successor features, a representation scheme for the value function that decouples the reward function from the dynamics of the environment. The second concept is the robust policy gradient, an algorithm that directly approximates a stochastic policy that maximizes the performance function on a set of tasks instead of a single, specific task.

We started by Theorem 1 that shows how we can reformulate the original policy gradient theorem to use successor features as the action-value function representation. Then we provided an upper bound on the performance of the transferred policy before any learning happens, which shows that the performance of transferred policy directly depends on the similarity of source and target task. Although learnt SFs may boost the learning speed, we empirically studied a scenario where using SFs can't be helpful. In fact, the dependence of SFs on the policy they are learned for, makes SFs unusable if the optimal policies of source and target task are so different. This problem suggests learning transferable SFs that reduce the dependence of SFs to a specific task's policy.

To tackle this problem, we then proposed a new performance metric (robust performance metric) for policy gradient RL methods that encourages the generalizability in the RL method. The learned policy and its SFs provide a better starting point for parameters of a new task so that starting from those parameters one can find the policy of a new unseen task much faster. The empirical evaluation of the proposed method showed the effectiveness of this objective in a transfer scenario with both A2C and PPO algorithms.

We conclude by discussing several ways in which our method may be used.

5.1 Future Work

Successor Feature representation has received a lot of attention recently. In addition to the ability of computing value of a policy immediately, the recent work by Lehnert and Littman (2019) showed how SFs can be a bridge between model-based and model-free approaches. Specifically, SFs can be seen as the expectation model of the world. On the other hand, Actor-Critic methods require to have the true critic of the current policy during the training, but computing the true critic increases the sample complexity of the method so in practice we usually only use an approximation of the true critic. In this

project, we proposed a method to use SFs for the critic representation to compute a better estimate of the critic immediately after the change of task. But this is not the only way to use SFs.

One possible future extension of this work is to use the learned SFs as the expectation model of the world to learn a more accurate estimate of the critic, so without increasing the sample complexity of the method, have a better estimate of the true critic.

In this project, we considered transfer on the set of tasks with only different reward functions. Although this is a type of transfer with many use cases, the more general type of transfer is when not only the reward function but also the probability transition function can be different in the set of tasks of interest. The robust policy gradient theorem that we provided only uses the linearity of value function represented by successor features. Another interesting extension of this work is to remove the assumption about the tasks having a shared dynamics and only use the more general case of linear value function approximation. In this case, the robust policy gradient theorem still holds and can be used to find generalizable policies in a more general transfer scenario.

Bibliography

Barreto, A., Borsa, D., Quan, J., Schaul, T., Silver, D., Hessel, M., Mankowitz, D. J., Zídek, A., and Munos, R. (2018). Transfer in deep reinforcement learning using successor features and generalised policy improvement. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 510–519. PMLR. 13

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., Silver, D., and van Hasselt, H. (2017). Successor features for transfer in reinforcement learning. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 4055–4065. 2, 3, 10, 15

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680. 1

Borsa, D., Barreto, A., Quan, J., Mankowitz, D. J., van Hasselt, H., Munos, R., Silver, D., and Schaul, T. (2019). Universal successor features approximators. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. 2, 3, 4

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. (2019). Quantifying generalization in reinforcement learning. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1282–1289. PMLR. 1

Dayan, P. (1993). Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624. 2, 10

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR. 3

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., and Pineau, J. (2018). An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3-4):219–354. 5

Heidrich-Meisner, V. and Igel, C. (2008). Evolution strategies for direct policy search. In Rudolph, G., Jansen, T., Lucas, S. M., Poloni, C., and Beume, N., editors, *Parallel Problem Solving from Nature - PPSN X, 10th International Conference Dortmund, Germany, September 13-17, 2008, Proceedings*, volume 5199 of *Lecture Notes in Computer Science*, pages 428–437. Springer. 8

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 25

Kirkpatrick, J., Pascanu, R., Rabinowitz, N. C., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2016). Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796. 3

Lazaric, A. (2012). Transfer in reinforcement learning: A framework and a survey. In Wiering, M. and van Otterlo, M., editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 143–173. Springer. 3

Lehnert, L. and Littman, M. L. (2019). Successor features support model-based and model-free reinforcement learning. *CoRR*, abs/1901.11437. 32

Lehnert, L., Tellex, S., and Littman, M. L. (2017). Advantages and limitations of using successor features for transfer in reinforcement learning. *CoRR*, abs/1708.00102. 16

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 1

Ma, C., Ashley, D. R., Wen, J., and Bengio, Y. (2020). Universal successor features for transfer reinforcement learning. *CoRR*, abs/2001.04025. 2, 3, 4

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M. J., and Bowling, M. (2018). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *J. Artif. Intell. Res.*, 61:523–562. 1

Madarasz, T. and Behrens, T. E. J. (2019). Better transfer learning with inferred successor maps. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 9026–9037. 2

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In Balcan, M. and Weinberger, K. Q., editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org. 2, 21

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602. 1

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533. 1

Mordatch, I., Lowrey, K., Andrew, G., Popovic, Z., and Todorov, E. V. (2015). Interactive control of diverse complex characters with neural networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 3132–3140. Curran Associates, Inc. 12

Rajeswaran, A., Lowrey, K., Todorov, E., and Kakade, S. M. (2017). Towards generalization and simplicity in continuous control. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6550–6561. 12

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *CoRR*, abs/1606.04671. 3

Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015). Universal value function approximators. In Bach, F. R. and Blei, D. M., editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1312–1320. JMLR.org. 3

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 12

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347. 2, 21, 22

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489. 1

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition. 5, 7, 15, 21

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A., Leen, T. K., and Müller, K., editors, *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 1057–1063. The MIT Press. 8, 14

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In Sonenberg, L., Stone, P., Tumer, K., and Yolum, P., editors, *10th International Conference on*

Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan, May 2-6, 2011, Volume 1-3, pages 761–768. IFAAMAS. 3

Szepesvári, C. (2010). *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers. 6, 7

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research (JMRL)*, 10:1633–1685. 1, 3, 12

Teh, Y. W., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. (2017). Distral: Robust multitask reinforcement learning. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 4496–4506. 3

Tieleman, T. and Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. 25

Whiteson, S., Tanner, B., Taylor, M. E., and Stone, P. (2011). Protecting against evaluation overfitting in empirical reinforcement learning. In *2011 IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning, ADPRL 2011, Paris, France, April 12-14, 2011*, pages 120–127. IEEE. 1

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8:229–256. 9

Zhang, A., Ballas, N., and Pineau, J. (2018a). A dissection of overfitting and generalization in continuous reinforcement learning. *CoRR*, abs/1806.07937. 1

Zhang, A., Satija, H., and Pineau, J. (2018b). Decoupling dynamics and reward for transfer learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net. 12