Transfer in Deep Reinforcement Learning Using Successor Features and Generalised Policy Improvement

André Barreto [1] **Diana Borsa** [1] **John Quan** [1] **Tom Schaul** [1] **David Silver** [1]
Matteo Hessel [1] **Daniel Mankowitz** [1] **Augustin Žídek** [1] **Rémi Munos** [1]

Abstract

The ability to transfer skills across tasks has the potential to scale up reinforcement learning (RL) agents to environments currently out of reach. Recently, a framework based on two ideas, successor features (SFs) and generalised policy improvement (GPI), has been introduced as a principled way of transferring skills. In this paper we extend the SF&GPI framework in two ways. One of the basic assumptions underlying the original formulation of SF&GPI is that rewards for all tasks of interest can be computed as linear combinations of a fixed set of features. We relax this constraint and show that the theoretical guarantees supporting the framework can be extended to any set of tasks that only differ in the reward function. Our second contribution is to show that one can use the reward functions themselves as features for future tasks, without any loss of expressiveness, thus removing the need to specify a set of features beforehand. This makes it possible to combine SF&GPI with deep learning in a more stable way. We empirically verify this claim on a complex 3D environment where observations are images from a first-person perspective. We show that the transfer promoted by SF&GPI leads to very good policies on unseen tasks almost instantaneously. We also describe how to learn policies specialised to the new tasks in a way that allows them to be added to the agent's set of skills, and thus be reused in the future.

1. Introduction

In recent years reinforcement learning (RL) has undergone a major change in terms of the scale of its applications: from relatively small and well-controlled benchmarks to problems designed to challenge humans—who are now consistently outperformed by artificial agents in domains considered out of reach only a few years ago (Mnih et al., 2015; Bowling et al., 2015; Silver et al., 2016; 2017).

At the core of this shift has been deep learning, a machine learning paradigm that recently attracted a lot of attention due to impressive accomplishments across many areas (Goodfellow et al., 2016). Despite the successes achieved by the combination of deep learning and RL, dubbed "deep RL", the agents's basic mechanics have remained essentially the same, with each problem tackled as an isolated monolithic challenge. An alternative would be for our agents to decompose a problem into smaller sub-problems, or "tasks", whose solutions can be reused multiple times, in different scenarios. This ability of explicitly transferring skills to quickly adapt to new tasks could lead to another leap in the scale of RL applications.

Recently, Barreto et al. (2017) proposed a framework for transfer based on two ideas: generalised policy improvement (GPI), a generalisation of the classic dynamic-programming operation, and successor features (SFs), a representation scheme that makes it possible to quickly evaluate a policy across many tasks. The SF&GPI approach is appealing for two reasons: it allows transfer to take place between any two tasks, regardless of their temporal order, and it integrates almost seamlessly within the RL framework.

In this paper we extend Barreto et al.'s (2017) framework in two ways. First, we argue that its applicability is broader than initially shown. SF&GPI was designed for the scenario where each task corresponds to a different reward function; one of the basic assumptions in the original formulation was that the rewards of all tasks can be computed as a linear combination of a fixed set of features. We show that such an assumption is not strictly necessary, and in fact it is possible to have guarantees on the performance of the transferred policy even on tasks that are not in the span of the features.

The realisation above adds some flexibility to the problem of computing features that are useful for transfer. Our second contribution is to show a simple way of addressing this

[1]DeepMind. Correspondence to: André Barreto <andrebarreto@google.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

problem that can be easily combined with deep learning. Specifically, by looking at the associated approximation from a slightly different angle, we show that one can replace the features with actual rewards. This makes it possible to apply SF&GPI online at scale. In order to verify this claim, we revisit one of Barreto et al.'s (2017) experiments in a much more challenging format, replacing a fully observable 2-dimensional environment with a 3-dimensional domain where observations are images from a first-person perspective. We show that the transfer promoted by SF&GPI leads to good policies on unseen tasks almost instantaneously. Furthermore, we show how to learn policies that are specialised to the new tasks in a way that allows them to be added to the agent's ever-growing set of skills, a crucial ability for continual learning (Thrun, 1996).

2. Background

In this section we present the background material that will serve as a foundation for the rest of the paper.

2.1. Reinforcement learning

In RL an agent interacts with an environment and selects actions in order to maximise the expected amount of reward received (Sutton & Barto, 1998). We model this scenario using the formalism of *Markov decision processes* (MDPs, Puterman, 1994). An MDP is a tuple $M \equiv (\mathcal{S}, \mathcal{A}, p, R, \gamma)$ whose components are defined as follows. The sets $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively. The function p defines the *dynamics* of the MDP: specifically, $p(\cdot|s, a)$ gives the next-state distribution upon taking action a in state s. The random variable $R(s, a, s')$ determines the reward received in the transition $s \xrightarrow{a} s'$; it is often convenient to consider the expected value of this variable, $r(s, a, s')$. Finally, the discount factor $\gamma \in [0, 1)$ gives smaller weights to future rewards. Given an MDP, the goal is to maximise the expected *return* $G_t = \sum_{i=0}^{\infty} \gamma^i R_{t+i}$, where $R_t = R(S_t, A_t, S_{t+1})$. To do so the agent computes a *policy* $\pi : \mathcal{S} \mapsto \mathcal{A}$.

A principled way to address the RL problem is to use methods derived from *dynamic programming* (DP) (Puterman, 1994). RL methods based on DP usually compute the *action-value function* of a policy π, defined as:

$$Q^{\pi}(s, a) \equiv \mathrm{E}^{\pi}\left[G_t \,|\, S_t = s, A_t = a\right], \tag{1}$$

where $\mathrm{E}^{\pi}[\cdot]$ denotes expectation over the sequences of transitions induced by π. The computation of $Q^{\pi}(s, a)$ is called *policy evaluation*. Once a policy π has been evaluated, we can compute a *greedy* policy $\pi'(s) \in \mathrm{argmax}_a Q^{\pi}(s, a)$ that is guaranteed to perform at least as well as π, that is: $Q^{\pi'}(s, a) \geq Q^{\pi}(s, a)$ for any $(s, a) \in \mathcal{S} \times \mathcal{A}$. The computation of π' is referred to as *policy improvement*. The alternation between policy evaluation and policy improvement is at the core of many DP-based RL algorithms, which usually carry out these steps only approximately.

As a convention, we will add a tilde to a symbol to indicate that the associated quantity is an approximation; we will then refer to the respective tunable parameters as $\boldsymbol{\theta}$. For example, the agent computes an approximation $\tilde{Q}^{\pi} \approx Q^{\pi}$ by tuning $\boldsymbol{\theta}_Q$. In deep RL some of the approximations computed by the agent, like $\tilde{Q}^{\pi}$, are represented by complex nonlinear approximators composed of many levels of nested tunable functions; among these, the most popular models are by far deep neural networks (Goodfellow et al., 2016).

In this paper we are interested in the problem of *transfer* in RL (Taylor & Stone, 2009; Lazaric, 2012). Specifically, we ask the question: given a set of MDPs that only differ in their reward function, how can we leverage knowledge gained in some MDPs to speed up the solution of others?

2.2. SF&GPI

Barreto et al. (2017) propose a framework to solve a restricted version of the problem above. Specifically, they restrict the scenario of interest to MDPs whose expected one-step reward can be written as

$$r(s, a, s') = \boldsymbol{\phi}(s, a, s')^{\top}\mathbf{w}, \tag{2}$$

where $\boldsymbol{\phi}(s, a, s') \in \mathbb{R}^d$ are features of (s, a, s') and $\mathbf{w} \in \mathbb{R}^d$ are weights. In order to build some intuition, it helps to think of $\boldsymbol{\phi}(s, a, s')$ as salient events that may be desirable or undesirable to the agent. Based on (2) one can define an environment $\mathcal{M}^{\phi}(\mathcal{S}, \mathcal{A}, p, \gamma)$ as

$$\mathcal{M}^{\phi} \equiv \{M(\mathcal{S}, \mathcal{A}, p, r, \gamma) | r(s, a, s') = \boldsymbol{\phi}(s, a, s')^{\top}\mathbf{w}\}, \tag{3}$$

that is, $\mathcal{M}^{\phi}$ is the set of MDPs induced by $\boldsymbol{\phi}$ through all possible instantiations of $\mathbf{w}$. We call each $M \in \mathcal{M}^{\phi}$ a *task*. Given a task $M_i \in \mathcal{M}^{\phi}$ defined by $\mathbf{w}_i \in \mathbb{R}^d$, we will use Q_i^{π} to refer to the value function of π on M_i.

Barreto et al. (2017) propose SF&GPI as a way to promote transfer between tasks in $\mathcal{M}^{\phi}$. As the name suggests, GPI is a generalisation of the policy improvement step described in Section 2.1. The difference is that in GPI the improved policy is computed based on a *set* of value functions rather than on a single one. Let $Q^{\pi_1}, Q^{\pi_2}, ...Q^{\pi_n}$ be the action-value functions of n policies defined on a given MDP, and let $Q^{\max} = \max_i Q^{\pi_i}$. If we define $\pi(s) \leftarrow \mathrm{argmax}_a Q^{\max}(s, a)$ for all $s \in \mathcal{S}$, then $Q^{\pi}(s, a) \geq Q^{\max}(s, a)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. The result also extends to the scenario where we replace Q^{π_i} with approximations $\tilde{Q}^{\pi_i}$, in which case the lower bound on $Q^{\pi}(s, a)$ gets looser with the approximation error, as in approximate DP (Bertsekas & Tsitsiklis, 1996).

In the context of transfer, GPI makes it possible to leverage knowledge accumulated over time, across multiple tasks, to learn a new task faster. Suppose that the agent has access

to n policies $\pi_1, \pi_2, ..., \pi_n$. These can be arbitrary policies, but for the sake of the argument let us assume they are solutions for tasks M_1, M_2, ..., M_n. Suppose also that when exposed to a new task $M_{n+1} \in \mathcal{M}^{\phi}$ the agent computes $Q_{n+1}^{\pi_i}$—the value functions of the policies π_i under the reward function induced by $\mathbf{w}_{n+1}$. In this case, applying GPI to the set $\{Q_{n+1}^{\pi_1}, Q_{n+1}^{\pi_2}, ..., Q_{n+1}^{\pi_n}\}$ will result in a policy that performs at least as well as any of the policies π_i.

Clearly, the approach above is appealing only if we have a way to quickly compute the value functions of the policies π_i on the task M_{n+1}. This is where SFs come in handy. SFs make it possible to compute the value of a policy π on *any* task $M_i \in \mathcal{M}^{\phi}$ by simply plugging in the representation the vector $\mathbf{w}_i$ defining the task. Specifically, if we substitute (2) in (1) we have

$$Q_i^{\pi}(s,a) = \mathrm{E}^{\pi}\left[\sum_{i=t}^{\infty}\gamma^{i-t}\boldsymbol{\phi}_{i+1} \,|\, S_t = s, A_t = a\right]^{\top}\mathbf{w}_i \equiv \boldsymbol{\psi}^{\pi}(s,a)^{\top}\mathbf{w}_i, \quad (4)$$

where $\boldsymbol{\phi}_t = \boldsymbol{\phi}(s_t, a_t, s_{t+1})$ and $\boldsymbol{\psi}^{\pi}(s,a)$ are the SFs of (s,a) under policy π. As one can see, SFs decouple the dynamics of the MDP M_i from its rewards (Dayan, 1993). One benefit of doing so is that if we replace $\mathbf{w}_i$ with $\mathbf{w}_j$ in (4) we immediately obtain the evaluation of π on task M_j. It is also easy to show that

$$\boldsymbol{\psi}^{\pi}(s,a) = \mathrm{E}^{\pi}[\boldsymbol{\phi}_{t+1} + \gamma\boldsymbol{\psi}^{\pi}(S_{t+1}, \pi(S_{t+1})) \,|\, S_t = s, A_t = a], \quad (5)$$

that is, SFs satisfy a Bellman equation in which ϕ_i play the role of rewards. Therefore, SFs can be learned using any conventional RL method (Szepesvári, 2010).

The combination of SFs and GPI provides a general framework for transfer in environments of the form (3). Suppose that we have learned the functions $Q_i^{\pi_i}$ using the representation scheme (4). When exposed to the task defined by $r_{n+1}(s,a,s') = \boldsymbol{\phi}(s,a,s')^{\top}\mathbf{w}_{n+1}$, as long as we have $\mathbf{w}_{n+1}$ we can immediately compute $Q_{n+1}^{\pi_i}(s,a) = \boldsymbol{\psi}^{\pi_i}(s,a)^{\top}\mathbf{w}_{n+1}$. This reduces the computation of all $Q_{n+1}^{\pi_i}$ to the problem of determining $\mathbf{w}_{n+1}$, which can be posed as a supervised learning problem whose objective is to minimise some loss derived from (2). Once $Q_{n+1}^{\pi_i}$ have been computed, we can apply GPI to derive a policy π that is no worse, and possibly better, than $\pi_1, ..., \pi_n$ on task M_{n+1}.

3. Extending the notion of environment

Barreto et al. (2017) focus on environments of the form (3). In this paper we propose a more general notion of environment:

$$\mathcal{M}(\mathcal{S}, \mathcal{A}, p, \gamma) \equiv \{M(\mathcal{S}, \mathcal{A}, p, \cdot, \gamma)\}. \quad (6)$$

$\mathcal{M}$ contains *all* MDPs that share the same $\mathcal{S}$, $\mathcal{A}$, p, and γ, regardless of whether their rewards can be computed as a linear combination of the features $\boldsymbol{\phi}$. Clearly, $\mathcal{M} \supset \mathcal{M}^{\phi}$.

We want to devise a transfer framework for environment $\mathcal{M}$. Ideally this framework should have two properties. First, it should be principled, in the sense that we should be able to provide theoretical guarantees regarding the performance of the transferred policies. Second, it should give rise to simple methods that are applicable in practice, preferably in combination with deep learning. Surprisingly, we can have both of these properties by simply looking at SF&GPI from a slightly different point of view, as we show next.

3.1. Guarantees on the extended environment

Barreto et al. (2017) provide theoretical guarantees on the performance of SF&GPI applied to any task $M \in \mathcal{M}^{\phi}$. In this section we show that it is in fact possible to derive guarantees for any task in $\mathcal{M}$. Our main result is below:

Proposition 1. *Let $M \in \mathcal{M}$ and let $Q_i^{\pi_j^*}$ be the action-value function of an optimal policy of $M_j \in \mathcal{M}$ when executed in $M_i \in \mathcal{M}$. Given approximations $\{\tilde{Q}_i^{\pi_1}, \tilde{Q}_i^{\pi_2}, ..., \tilde{Q}_i^{\pi_n}\}$ such that $\left|Q_i^{\pi_j^*}(s,a) - \tilde{Q}_i^{\pi_j}(s,a)\right| \leq \epsilon$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, and $j \in \{1, 2, ..., n\}$, let*

$$\pi(s) \in \mathrm{argmax}_a \mathrm{max}_j \tilde{Q}_i^{\pi_j}(s,a). \quad (7)$$

Then,

$$\|Q^* - Q^{\pi}\|_{\infty} \leq \frac{2}{1-\gamma}\left(\|r - r_i\|_{\infty} + \min_j \|r_i - r_j\|_{\infty} + \epsilon\right), \quad (8)$$

where Q^* is the optimal value function of M, Q^{π} is the value function of π in M, and $\|f - g\|_{\infty} = \max_{s,a}|f(s,a) - g(s,a)|$.

The proof of Proposition 1 is in the supplementary material. Our result provides guarantees on the performance of the GPI policy (7) applied to an MDP M with *arbitrary* reward function $r(s,a)$. Note that, although the theorem does not restrict any of the tasks to be in $\mathcal{M}^{\phi}$, in order to compute the GPI policy (7) we still need an efficient way of evaluating the policies π_j on task M_i. As explained in Section 2, one way to accomplish that is to assume that M_i and all M_j appearing in the statement of the theorem belong to $\mathcal{M}^{\phi}$; this allows us to use SFs to quickly compute $\tilde{Q}_i^{\pi_j}(s,a)$.

Let us thus take a closer look at Proposition 1 under the assumption that all MDPs belong to $\mathcal{M}^{\phi}$, except perhaps for M. The proposition is based on a reference MDP $M_i \in \mathcal{M}^{\phi}$. If $M_i = M_j$ for some j, the second term of (8) disappears, and we end up with a lower bound on the performance of a policy computed for M_i when executed in M. More generally, one should think of M_i as the MDP in $\mathcal{M}^{\phi}$ that is "closest" to M in some sense (so it may be that $M_i \neq M_j$ for all j). Specifically, if $r_i(s,a,s') = \boldsymbol{\phi}(s,a,s')^{\top}\mathbf{w}_i$, we can think of $\mathbf{w}_i$ as the vector that provides the best approximation $\boldsymbol{\phi}(s,a,s')^{\top}\mathbf{w}_i \approx r(s,a,s')$ under some well-defined criterion. The first term of (8) can thus be seen as the

"distance" between M and $\mathcal{M}^{\phi}$, which suggests that the performance of SF&GPI should degrade gracefully as we move away from the original environment $\mathcal{M}^{\phi}$. In the particular case where $M \in \mathcal{M}^{\phi}$ the first term of (8) vanishes, and we recover Barreto et al.'s (2017) Theorem 2.

3.2. Uncovering the structure of the environment

We want to solve a subset of the tasks in $\mathcal{M}$ and use GPI to promote transfer between these tasks. In order to do so efficiently, we need a function $\boldsymbol{\phi}$ that covers the tasks of interest as much as possible. If we were to rely on Barreto et al.'s (2017) original results, we would have guarantees on the performance of the GPI policy only if $\boldsymbol{\phi}$ spanned *all* the tasks of interest. Proposition 1 allows us to remove this requirement, since now we have theoretical guarantees for any choice of $\boldsymbol{\phi}$. In practice, though, we want features $\boldsymbol{\phi}$ such that the first term of (8) is small for all tasks of interest.

There might be contexts in which we have direct access to features $\boldsymbol{\phi}(s, a, s')$ that satisfy (2), either exactly or approximately. Here though we are interested in the more general scenario where this structure is not available, nor given to the agent in any form. In this case the use of SFs requires a way of unveiling such a structure.

Barreto et al. (2017) assume the existence of a $\boldsymbol{\phi} \in \mathbb{R}^d$ that satisfy (2) exactly and formulate the problem of computing an approximate $\tilde{\boldsymbol{\phi}}$ as a multi-task learning problem. The problem is decomposed into D regressions, each one associated with a task. For reasons that will become clear shortly we will call these tasks *base tasks* and denote them by $\hat{\mathcal{M}} \equiv \{M_1, M_2, ..., M_D\} \subset \mathcal{M}^{\phi}$. The multi-task problem thus consists in solving the approximations

$$\tilde{\boldsymbol{\phi}}(s, a, s')^{\top} \tilde{\mathbf{w}}_i \approx r_i(s, a, s'), \text{ for } i = 1, 2, ..., D, \quad (9)$$

where r_i is the reward of M_i (Caruana, 1997; Baxter, 2000).

In this section we argue that (9) can be replaced by a much simpler approximation problem. Suppose for a moment that we know a function $\boldsymbol{\phi}$ and D vectors $\mathbf{w}_i$ that satisfy (9) exactly. If we then stack the vectors $\mathbf{w}_i$ to obtain a matrix $\mathbf{W} \in \mathbb{R}^{D \times d}$, we can write $\mathbf{r}(s, a, s') = \mathbf{W}\boldsymbol{\phi}(s, a, s')$, where the i^{th} element of $\mathbf{r}(s, a, s') \in \mathbb{R}^D$ is $r_i(s, a, s')$. Now, as long as we have d linearly independent tasks $\mathbf{w}_i$, we can write $\boldsymbol{\phi}(s, a, s') = (\mathbf{W}^{\top}\mathbf{W})^{-1}\mathbf{W}^{\top}\mathbf{r}(s, a, s') = \mathbf{W}^{\dagger}\mathbf{r}(s, a, s')$. Since $\boldsymbol{\phi}$ is given by a linear transformation of $\mathbf{r}$, any task representable by the former can also be represented by the latter. To see why this is so, note that for any task in $\mathcal{M}^{\phi}$ we have $r(s, a, s') = \mathbf{w}^{\top}\boldsymbol{\phi}(s, a, s') = \mathbf{w}^{\top}\mathbf{W}^{\dagger}\mathbf{r}(s, a, s') = (\mathbf{w}')^{\top}\mathbf{r}(s, a, s')$. Therefore, we can use the rewards $\mathbf{r}$ themselves as features, which means that we can replace (9) with the much simpler approximation

$$\tilde{\boldsymbol{\phi}}(s, a, s') = \tilde{\mathbf{r}}(s, a, s') \approx \mathbf{r}(s, a, s'). \quad (10)$$

One potential drawback of directly approximating $\mathbf{r}(s, a, s')$ is that we no longer have the flexibility of distinguishing between d, the dimension of the environment $\mathcal{M}^{\phi}$, and D, the number of base tasks in $\hat{\mathcal{M}}$. Since in general we will solve (9) or (10) based on data, having $D > d$ may lead to a better approximation. On the other hand, using $\tilde{\mathbf{r}}(s, a, s')$ as features has a number of advantages that in many scenarios of interest should largely outweigh the loss in flexibility.

One such advantage becomes clear when we look at the scenario above from a different perspective. Instead of assuming we know a $\boldsymbol{\phi}$ and a $\mathbf{W}$ that satisfy (9), we can note that, given any set of D tasks $r_i(s, a, s')$, $k \leq D$ of them will be linearly independent. Thus, the reasoning above applies without modification. Specifically, if we use the tasks's rewards as features, as in (10), we can think of them as *inducing* a $\boldsymbol{\phi}(s, a, s') \in \mathbb{R}^k$—and, as a consequence, an environment $\mathcal{M}^{\phi}$. This highlights a benefit of replacing (9) with (10): the fact that we can work directly in the space of tasks, which in many cases may be more intuitive. When we think of $\boldsymbol{\phi}$ as a non-observable, abstract, quantity, it can be difficult to define what exactly the base tasks should be. In contrast, when we look at $\mathbf{r}$ directly the question we are trying to answer is: is it possible to approximate the reward function of all tasks of interest as a linear combination of the functions r_i? As one can see, the set $\hat{\mathcal{M}}$ works exactly as a basis, and thus the name "base tasks".

Another interesting consequence of using an approximation $\tilde{\boldsymbol{\phi}}(s, a, s') \approx \mathbf{r}(s, a, s')$ as features is that the resulting SFs are nothing but ordinary action-value functions. Specifically, the j^{th} component of $\tilde{\boldsymbol{\psi}}^{\pi_i}(s, a)$ is simply $\tilde{Q}_j^{\pi_i}(s, a)$. Next we discuss how this gives rise to a simple approach that can be combined with deep learning in a stable way.

4. Transfer in deep reinforcement learning

As described in the introduction, we are interested in using SF&GPI to build scalable agents that can be combined with deep learning in a stable way. Since deep learning generally involves vast amounts of data whose storage is impractical, we will be mostly focusing on methods that work online. As a motivating example for this section we show in Algorithm 1 how SFs and GPI can be combined with Watkins & Dayan's (1992) Q-learning.[1]

Algorithm 1 differs from standard Q-learning in two main ways. First, instead of selecting actions based on the value function being learned, the behaviour of the agent is determined by GPI (line 7). Depending on the set of SFs $\tilde{\Psi}$ used by the algorithm, this can be a significant improvement over the greedy policy induced by $\tilde{Q}^{\pi_{n+1}}$, which usually is the main determinant of a Q-learning agent's behaviour.

[1] We use $x \xleftarrow{\alpha} y$ meaning $x \leftarrow x + \alpha y$. We also use $\nabla_{\boldsymbol{\theta}} f(x)$ to denote the gradient of $f(x)$ with respect to the parameters $\boldsymbol{\theta}$.

Algorithm 1 SF&GPI with ϵ-greedy Q-learning

Require: $\tilde{\boldsymbol{\phi}}$, $\tilde{\Psi} \equiv \{\tilde{\boldsymbol{\psi}}^{\pi_1}, ..., \tilde{\boldsymbol{\psi}}^{\pi_n}\}$ features, SFs; extend_basis learn a new SF?; $\alpha_\psi, \alpha_w, \epsilon$, ns hyper-parameters

1: **if** extend_basis **then**
2: create $\tilde{\boldsymbol{\psi}}^{\pi_{n+1}}$ parametrised by $\boldsymbol{\theta}_\psi$
3: $\tilde{\Psi} \leftarrow \tilde{\Psi} \cup \{\tilde{\boldsymbol{\psi}}^{\pi_{n+1}}\}$
4: select initial state $s \in \mathcal{S}$
5: **for** ns steps **do**
6: **if** Bernoulli(ϵ)=1 **then** $a \leftarrow$ Uniform($\mathcal{A}$) // exploration
7: **else** $a \leftarrow \text{argmax}_b \max_i \tilde{\boldsymbol{\psi}}^{\pi_i}(s,b)^\top \tilde{\mathbf{w}}$ // GPI
8: Execute action a and observe r and s'
9: $\tilde{\mathbf{w}} \xleftarrow{\alpha_w} \left[r - \tilde{\boldsymbol{\phi}}(s,a,s')^\top \tilde{\mathbf{w}}\right] \tilde{\boldsymbol{\phi}}(s,a,s')$ // learn $\tilde{\mathbf{w}}$
10: **if** extend_basis **then** // will learn new SFs
11: $a' \leftarrow \text{argmax}_b \tilde{\boldsymbol{\psi}}^{\pi_{n+1}}(s,b)^\top \tilde{\mathbf{w}}$
12: **for** $i \leftarrow 1, 2, ..., d$ **do**
13: $\delta \leftarrow \tilde{\boldsymbol{\phi}}_i(s,a,s') + \gamma \tilde{\boldsymbol{\psi}}_i^{\pi_{n+1}}(s',a') - \tilde{\boldsymbol{\psi}}_i^{\pi_{n+1}}(s,a)$
14: $\boldsymbol{\theta}_\psi \xleftarrow{\alpha_\psi} \delta \nabla_{\boldsymbol{\theta}_\psi} \tilde{\boldsymbol{\psi}}_i^{\pi_{n+1}}(s,a)$ // learn $\tilde{\boldsymbol{\psi}}^{\pi_{n+1}}$
15: **if** s' is not terminal **then** $s \leftarrow s'$
16: **else** select initial state $s \in \mathcal{S}$

Algorithm 1 also deviates from conventional Q-learning in the way a policy is learned. There are two possibilities here. One of them is for the agent to rely exclusively on the GPI policy computed over $\tilde{\Psi}$ (when the variable extend_basis is set to false). In this case no specialised policy is learned for the current task, which reduces the RL problem to the supervised problem of determining $\tilde{\mathbf{w}}$ (solved in line 9 as a least-squares regression).

Another possibility is to use data collected by the GPI policy to learn a policy π_{n+1} specifically tailored for the task. As shown in lines 13 and 14, this comes down to solving equation (5). When π_{n+1} is learned the function $\tilde{Q}^{\pi_{n+1}}(s,a) = \tilde{\boldsymbol{\psi}}^{\pi_{n+1}}(s,a)^\top \tilde{\mathbf{w}}$ also takes part in GPI. This means that, if the approximations $\tilde{Q}^{\pi_i}(s,a)$ are reasonably accurate, the policy computed by Algorithm 1 should be strictly better than Q-learning's counterpart. The SFs $\tilde{\boldsymbol{\psi}}^{\pi_{n+1}}$ can then be added to the set $\tilde{\Psi}$, and therefore a subsequent execution of Algorithm 1 will result in an even stronger agent. The ability to build and continually refine a set of skills is widely regarded as a desirable feature for continual (or lifelong) learning (Thrun, 1996).

4.1. Challenges involved in building features

In order to use Algorithm 1, or any variant of the SF&GPI framework, we need the features $\tilde{\boldsymbol{\phi}}(s,a,s')$. A natural way of addressing the computation of $\tilde{\boldsymbol{\phi}}(s,a,s')$ is to see it as a multi-task problem, as in (9). We now discuss the difficulties involved in solving this problem online at scale.

The solution of (9) requires data coming from D base tasks. In principle, we could look at the collection of sample transitions as a completely separate process. However, here we are assuming that the base tasks $\hat{\mathcal{M}}$ are part of the RL problem, that is, we want to collect data using policies that are competent in $\hat{\mathcal{M}}$. In order to maximise the potential for transfer, while learning the policies π_i for the base tasks M_i we should also learn the associated SFs $\tilde{\boldsymbol{\psi}}^{\pi_i}$; this corresponds to building the initial set $\tilde{\Psi}$ used by Algorithm 1. Unfortunately, learning value functions in the form (4) while solving (9) can be problematic. Since the SFs $\tilde{\boldsymbol{\psi}}^{\pi_i}$ depend on $\tilde{\boldsymbol{\phi}}$, learning the former while refining the latter can clearly lead to undesirable solutions (this is akin to having a non-stationary reward function). On top of that, the computation of $\tilde{\boldsymbol{\phi}}$ itself depends on the SFs $\tilde{\boldsymbol{\psi}}^{\pi_i}$, for they ultimately define the data distribution used to solve (9). This circular dependency can make the process of concurrently learning $\tilde{\boldsymbol{\phi}}$ and $\tilde{\boldsymbol{\psi}}^{\pi_i}$ unstable—something we observed in practice.

One possible solution is to use a conventional value function representation while solving (9) and only learn SFs for the subsequent tasks (Barreto et al., 2017). This has the disadvantage of not reusing the policies learned in $\hat{\mathcal{M}}$ for transfer. Alternatively, one can store all the data and learn the SFs associated with the base tasks only *after* $\tilde{\boldsymbol{\phi}}$ has been learned, but this may be difficult in scenarios involving large amounts of data. Besides, any approximation errors in $\tilde{\boldsymbol{\phi}}$ will already be reflected in the initial $\tilde{\Psi}$ computed in $\hat{\mathcal{M}}$.

4.2. Learning features online while retaining transferable knowledge

To recapitulate, we are interested in solving D base tasks M_i and, while doing so, build $\tilde{\boldsymbol{\phi}}$ and the initial set $\tilde{\Psi}$ to be used by Algorithm 1. We want $\tilde{\boldsymbol{\phi}}$ and $\tilde{\Psi}$ to be learned concurrently, so we do not have to store transitions, and preferably $\tilde{\Psi}$ should not reflect approximation errors in $\tilde{\boldsymbol{\phi}}$.

We argue that one can accomplish all of the above by replacing (9) with (10), that is, by directly approximating $\mathbf{r}(s,a,s')$, which is observable, and adopting the resulting approximation as the features $\tilde{\boldsymbol{\phi}}$ required by Algorithm 1.

As discussed in Section 3.2, when using rewards as features the resulting SFs are collections of value functions: $\tilde{\boldsymbol{\psi}}^{\pi_i} = \tilde{\mathbf{Q}}^{\pi_i} \equiv [\tilde{Q}_1^{\pi_i}, \tilde{Q}_2^{\pi_i}, ..., \tilde{Q}_D^{\pi_i}]$. This leads to a particularly simple way of building the features $\tilde{\boldsymbol{\phi}}$ while retaining transferable knowledge in $\tilde{\Psi}$. Given a set of D base tasks M_i, while solving them we only need to carry out two extra operations: compute approximations $\tilde{r}_i(s,a,s')$ of the functions $r_i(s,a,s')$, to be used as $\tilde{\boldsymbol{\phi}}$, and evaluate the resulting policies on all tasks—*i.e.*, compute $\tilde{Q}_j^{\pi_i}$—to build $\tilde{\Psi}$.

Before providing a practical method to compute $\tilde{\boldsymbol{\phi}}$ and $\tilde{\Psi}$, we note that, although the approximations $\tilde{r}_i(s,a,s')$ can be learned independently from each other, the computation of $\tilde{\mathbf{Q}}^{\pi_i}$ requires policy π_i to be evaluated under different

reward signals. This can be accomplished in different ways; here we assume that the agent is able to interact with the tasks M_i in parallel. We can consider that at each transition the agent observes rewards from all the base tasks, $\mathbf{r} \in \mathbb{R}^D$, or a single scalar r_i associated with one of them. We will assume the latter, but our solution readily extends to the scenario where the agent simultaneously observes D tasks.

Algorithm 2 shows a possible way of implementing our solution, again using Q-learning as the basic RL method. We highlight the fact that GPI can already be used in this phase, as shown in line 4, which means that the policies π_i can "cooperate" to solve each task M_i.

Algorithm 2 Build SF&GPI basis with ϵ-greedy Q-learning

Require: $\begin{cases} M_1, M_2, ..., M_D & \text{base tasks} \\ \alpha_Q, \alpha_r, \epsilon, \text{ns} & \text{hyper-parameters} \end{cases}$

1: **for** ns steps **do**
2: select a task $t \in \{1, 2, ..., D\}$ and a state $s \in \mathcal{S}$
3: **if** Bernoulli(ϵ)=1 **then** $a \leftarrow$ Uniform($\mathcal{A}$) // exploration
4: **else** $a \leftarrow \text{argmax}_b \max_i \tilde{Q}_t^{\pi_i}(s, b)$ // GPI
5: Execute action a in M_t and observe r and s'
6: $\boldsymbol{\theta}_r \xleftarrow{\alpha_r} [r - \tilde{r}_t(s, a, s')] \nabla_{\boldsymbol{\theta}_r} \tilde{r}_t(s, a, s')$
7: **for** $i \leftarrow 1, 2, ..., D$ **do**
8: $a' \leftarrow \text{argmax}_b \tilde{Q}_i^{\pi_i}(s, b)$ // $a' \equiv \pi_i(s)$
9: $\boldsymbol{\theta}_Q \xleftarrow{\alpha_Q} \left[r + \gamma \tilde{Q}_t^{\pi_i}(s', a') - \tilde{Q}_t^{\pi_i}(s, a)\right] \nabla_{\boldsymbol{\theta}_Q} \tilde{Q}_t^{\pi_i}(s, a)$
10: **return** $\tilde{\boldsymbol{\phi}} \equiv [\tilde{r}_1, ..., \tilde{r}_D]$ and $\tilde{\Psi} \equiv \{\tilde{\mathbf{Q}}^{\pi_1}, ..., \tilde{\mathbf{Q}}^{\pi_D}\}$

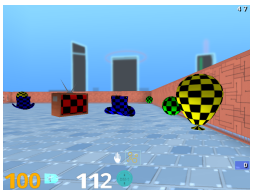
Note that if we were to learn general $\tilde{\boldsymbol{\psi}}^{\pi_i}(s, a)$ and $\tilde{\boldsymbol{\phi}}(s, a, s')$ in parallel we would necessarily have to use the latter to update the former, which means computing an approximation on top of another (see (4)). In contrast, when learning $\tilde{Q}_j^{\pi_i}(s, a)$ we can use the *actual* rewards $r_j(s, a, s')$, as opposed to the approximations $\tilde{r}_j(s, a, s')$ (line 9 of Algorithm 2). This means that $\tilde{\boldsymbol{\phi}}$ and $\tilde{\Psi}$ can be learned concurrently without the accumulation of approximation errors in $\tilde{\boldsymbol{\psi}}^{\pi_i}$, as promised.

5. Experiments

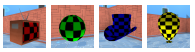
In this section we use experiments to assess whether the proposed approach can indeed promote transfer on large-scale domains. Here we focus on what we consider the most relevant aspects of our experiments; for further details and additional results please see the supplementary material.

5.1. Environment

The environment we consider is conceptually similar to one of the problems used by Barreto et al. (2017) to evaluate their framework: the agent has to navigate in a room picking up desirable objects while avoiding undesirable ones. Here



(a) Screenshot of environment



M_1	1	0	0	0
M_2	0	1	0	0
M_3	0	0	1	0
M_4	0	0	0	1

(b) Base tasks $\hat{\mathcal{M}}$

Figure 1. Environment and base tasks.

the problem is tackled in a particularly challenging format: instead of observing the state s_t at time step t, as in the original experiments, the agent interacts with the environment from a first-person perspective, only receiving as observation a 84×84 image o_t that is insufficient to disambiguate the actual underlying state of the MDP (see Figure 1(a)).

We used Beattie et al.'s (2016) DeepMind Lab platform to design our 3D environment, which works as follows. The agent finds itself in a room full of objects of different types. There are five instances of each object type: "TV", "ball", "hat", and "balloon". Whenever the agent hits an object it picks it up and another object of the same type appears at a random location in the room. This process goes on for a minute, after which a new episode starts.

The type of an object determines the reward associated with it; thus, a task is defined (and can be referred to) by four numbers indicating the rewards attached to each object type. For example, in task `1-100` the agent is interested in objects of the first type and should avoid objects of the second type, while the other object types are irrelevant. We defined base tasks that will be used by Algorithm 2 to build $\tilde{\boldsymbol{\phi}}$ and $\tilde{\Psi}$ (see Figure 1(b)). The transfer ability of the algorithms will be assessed on different, unseen, tasks, referred to as *test tasks*.

5.2. Agents

The SF&GPI agent adopted in the experiments is a variation of Algorithms 1 and 2 that uses Watkins's (1989) $Q(\lambda)$ to apply Q-learning with eligibility traces. The functions $\tilde{\boldsymbol{\phi}}$ and $\tilde{\Psi}$ are computed by a deep neural network whose architecture is shown in Figure 2. The network is composed of three parts. The first one uses the history of observations and actions up to time t, h_t, to compute a state signal $\tilde{s}_t = f(h_t)$. The construction of $\tilde{s}_t$ can itself be broken into two stages corresponding to specific functional modules: a convolutional network (CNN) to handle the pixel-based observation o_t and a long short-term network (LSTM) to compute $f(h_t)$ in a recursive way (LeCun et al., 1998; Hochreiter & Schmidhuber, 1997). The second part of the network is composed of $D + 1$ specialised blocks that receive $\tilde{s}_t$ as input and compute $\tilde{\boldsymbol{\phi}}(\tilde{s}_t, a)$ and $\tilde{\boldsymbol{\psi}}^{\pi_i}(\tilde{s}_t, a)$ for all $a \in \mathcal{A}$. Each one of these blocks is a multilayer perceptron (MLP) with a single hidden layer (Rumelhart et al., 1986). The third part of the network is simply $\tilde{\mathbf{w}}$, which combined with

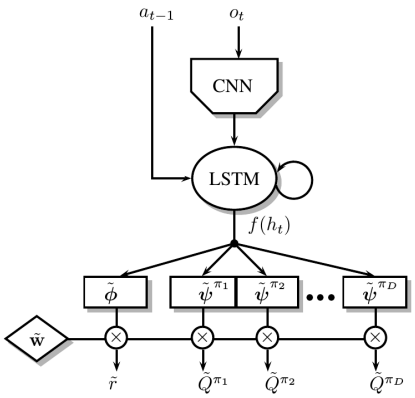


Figure 2. Deep architecture used. Rectangles represent MLPs. $\tilde{\phi}$ and $\tilde{\psi}^{\pi_i}$ will provide the final approximations.

The entire architecture was trained end-to-end through Algorithm 2 using the base tasks shown in Figure 1(b). After the SF&GPI agent had been trained it was tested on a test task, now using Algorithm 1 with the newly-learned $\tilde{\phi}$ and $\tilde{\Psi}$. In order to handle the large number of sample trajectories needed in our environment both Algorithms 1 and 2 used the IMPALA distributed architecture (Espeholt et al., 2018).

Algorithm 1 was run with and without the learning of a specialised policy (controlled via the variable $\mathrm{extend\_basis}$). We call the corresponding versions of the algorithm "SF&GPI-continual" and "SF&GPI-transfer", respectively. We compare SF&GPI with baseline agents that use the same network architecture, learning algorithm, and distributed data processing. The only difference is in the way the network shown in Figure 2 is updated and used during the test phase. Specifically, we ignore the MLPs used to compute $\tilde{\phi}$ and $\tilde{\psi}^{\pi_i}$ and instead add another MLP, with the exact same architecture, to be jointly trained with $\tilde{\mathbf{w}}$ through $Q(\lambda)$. We then distinguish three baselines. The first one uses the state signal $\tilde{s}_t = f(h_t)$ learned in the base tasks to compute an approximation $\tilde{Q}(\tilde{s}, a)$—that is, both the CNN and the LSTM are fixed. We will refer to this method simply as $Q(\lambda)$. The second baseline is allowed to modify $f(h_t)$ during test, so we call it "$DQ(\lambda)$ fine tuning" as a reference to its deep architecture. Finally, the third baseline, "$DQ(\lambda)$ from scratch", learns its own representation $f(h_t)$.

5.3. Results and discussion

Figure 3 shows the results of SF&GPI and the baselines on a selection of test tasks. The first thing that stands out in the figures is the fact that SF&GPI-transfer learns very good policies for the test tasks almost instantaneously. In fact, as this version of the algorithm is solving a simple supervised learning problem, its learning progress is almost imperceptible at the scale the RL problem unfolds. Since the baselines *are* solving the full RL problem, in some tasks their performance eventually reaches, or even surpasses, that of the transferred policies. SF&GPI-continual combines the desirable properties of both SF&GPI-transfer and the baselines. On one hand, it still benefits from the instantaneous transfer promoted by SF&GPI. On the other hand, its performance keeps improving, since in this case the transferred policy is used to learn a policy specialised to the current task. As a result, SF&GPI-continual outperforms the other methods in almost all of the tasks.[2]

Another interesting trend shown in Figures 3 is the fact that SF&GPI performs well on test tasks with negative rewards—in some cases considerably better than the alternative methods—, even though the agent only experienced positive rewards on the base tasks. This is an indication that the transferred GPI policy is combining the policies π_i for the base tasks in a non-trivial way (line 7 of Algorithm 1).

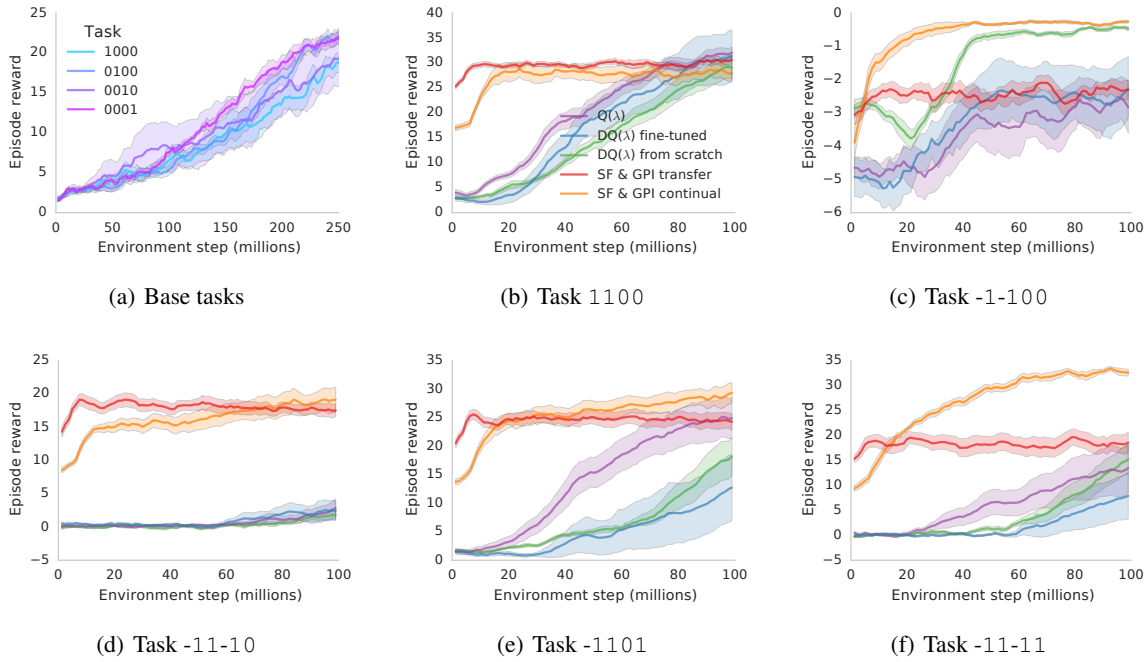
In this paper we argue that SF&GPI can be applied even if assumption (2) is not strictly satisfied. In order to illustrate this point, we reran the experiments with SF&GPI-transfer using a set of linearly-depend base tasks, $\hat{\mathcal{M}}' \equiv \{\mathtt{1000}, \mathtt{0100}, \mathtt{0011}, \mathtt{1100}\}$. Clearly, $\hat{\mathcal{M}}'$ can only represent tasks in which the rewards associated with the third and fourth object types are the same. We thus fixed the rewards associated with the first two object types and compared the results of SF&GPI-transfer using $\hat{\mathcal{M}}$ and $\hat{\mathcal{M}}'$ on several tasks where this is not the case. The comparison is shown in Figure 4. As shown in the figure, although using a linearly-dependent set of base tasks does hinder transfer in some cases, in general it does not have a strong impact on the results. This smooth degradation of the performance is in accordance with Proposition 1.

The result above also illustrates an interesting distinction between the space of reward functions and the associated space of policies. Although we want to be able to represent the reward functions of all tasks of interest, this does not guarantee that the resulting GPI policy will perform well. To see this, suppose we replace the positive rewards in $\hat{\mathcal{M}}$ with negative ones. Clearly, in this case we would still have a basis spanning the same space of rewards; however, since now a policy that stands still is optimal in all tasks M_i, we should not expect GPI to give rise to good policies in tasks with positive rewards. One can ask how to define a "behavioural basis" that leads to good policies across $\mathcal{M}$ through GPI. We leave this as an interesting open question.

6. Related work

Recently, there has been a resurgence of the subject of transfer in the deep RL literature. Teh et al. (2017) propose an

[2] A video of SF&GPI-transfer is included as a supplement, and can also be found on this link: https://youtu.be/-dTnqfwTRMI.



(a) Base tasks (b) Task `1100` (c) Task `-1-100`

(d) Task `-11-10` (e) Task `-1101` (f) Task `-11-11`

Figure 3. Average reward per episode on the base tasks and a selection of test tasks. The x axes have different scales because the amount of reward available changes across tasks. Shaded regions are one standard deviation over 10 runs.

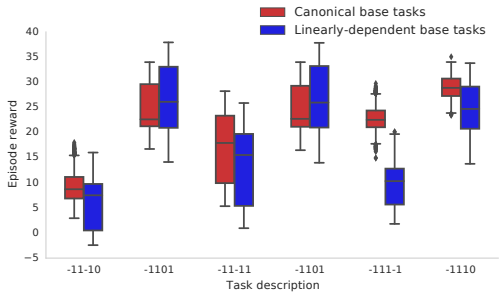


Figure 4. Performance of SF&GPI-transfer using base tasks $\hat{\mathcal{M}}$ and $\hat{\mathcal{M}}'$. The box plots summarise the distribution of the rewards received per episode between 50 and 100 million steps of learning.

approach for the multi-task problem—which in our case is the learning of base tasks—that uses a shared policy as a regulariser for specialised policies. Finn et al. (2017) propose to achieve fast transfer, which is akin to SF&GPI-transfer, by focusing on adaptability, rather than performance, during learning. Rusu et al. (2016) and Kirkpatrick et al. (2016) introduce neural-network architectures well-suited for continual learning. There have also been previous attempts to combine SFs and deep learning for transfer, but none of them used GPI (Kulkarni et al., 2016; Zhang et al., 2016).

Many works on the combination of deep RL and transfer propose modular network architectures that naturally induce a decomposition of the problem (Devin et al., 2016; Heess et al., 2017; Clavera et al., 2017). Among these, a recurrent theme is the existence of sub-networks specialised in different skills that are managed by another network (Heess et al., 2016; Frans et al., 2017; Oh et al., 2017). This highlights an interesting connection between transfer learning and hierarchical RL, which has also recently re-emerged in the deep RL literature (Vezhnevets et al., 2017; Bacon et al., 2017).

7. Conclusion

In this paper we extended the SF&GPI transfer framework in two ways. First, we showed that the theoretical guarantees supporting the framework can be extended to any set of MDPs that only differ in the reward function, regardless of whether their rewards can be computed as a linear combination of a set of features or not. In order to use SF&GPI in practice we still need the reward features, though; our second contribution is to show that these features can be the reward functions of a set of MDPs. This reinterpretation of the problem makes it possible to combine SF&GPI with deep learning in a stable way. We empirically verified this claim on a complex 3D environment that requires hundreds of millions of transitions to be solved. We showed that, by turning an RL task into a supervised learning problem, SF&GPI-transfer is able to provide skilful, non-trivial, policies almost instantaneously. We also showed how these policies can be used by SF&GPI-continual to learn specialised policies, which can then be added to the agent's set of skills. Together, these concepts can help endow an agent with the ability to build, refine, and use a set of skills while interacting with the environment.

Acknowledgements

The authors would like to thank Will Dabney and Hado van Hasselt for the invaluable discussions during the development of this paper. We also thank the anonymous reviewers, whose comments and suggestions helped to improve the paper considerably.

References

Bacon, P., Harb, J., and Precup, D. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1726–1734, 2017.

Barreto, A., Dabney, W., Munos, R., Hunt, J., Schaul, T., van Hasselt, H., and Silver, D. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Baxter, J. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Bertsekas, D. P. and Tsitsiklis, J. N. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

Bowling, M., Burch, N., Johanson, M., and Tammelin, O. Heads-up limit hold'em poker is solved. *Science*, 347 (6218):145–149, January 2015.

Caruana, R. Multitask learning. *Machine Learning*, 28(1): 41–75, 1997.

Clavera, I., Held, D., and Abbeel, P. Policy transfer via modularity and reward guiding. In *International Conference on Intelligent Robots and Systems (IROS)*, 2017.

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

Devin, C., Gupta, A., Darrell, T., Abbeel, P., and Levine, S. Learning modular neural network policies for multi-task and multi-robot transfer. *CoRR*, abs/1609.07088, 2016. URL http://arxiv.org/abs/1609.07088.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. URL http://arxiv.org/abs/1703.03400.

Frans, K., Ho, J., Chen, X., Abbeel, P., and Schulman, J. Meta learning shared hierarchies. *CoRR*, abs/1710.09767, 2017. URL http://arxiv.org/abs/1710.09767.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Heess, N., Wayne, G., Tassa, Y., Lillicrap, T. P., Riedmiller, M. A., and Silver, D. Learning and transfer of modulated locomotor controllers. *CoRR*, abs/1610.05182, 2016. URL http://arxiv.org/abs/1610.05182.

Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S. M. A., Riedmiller, M. A., and Silver, D. Emergence of locomotion behaviours in rich environments. *CoRR*, abs/1707.02286, 2017. URL http://arxiv.org/abs/1707.02286.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

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. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016. URL http://arxiv.org/abs/1612.00796.

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

Lazaric, A. *Transfer in Reinforcement Learning: A Framework and a Survey*, pp. 143–173. 2012.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

Oh, J., Singh, S. P., Lee, H., and Kohli, P. Zero-shot task generalization with multi-task deep reinforcement learning. *CoRR*, abs/1706.05064, 2017. URL http://arxiv.org/abs/1706.05064.

Puterman, M. L. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Parallel distributed processing: Explorations in the microstructure of cognition. chapter Learning Internal Representations by Error Propagation, pp. 318–362. MIT Press, Cambridge, MA, USA, 1986.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *CoRR*, abs/1606.04671, 2016. URL http://arxiv.org/abs/1606.04671.

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., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of Go without human knowledge. *Nature*, 550, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 1998. URL http://www-anw.cs.umass.edu/~rich/book/the-book.html.

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

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

Teh, Y. W., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 4499–4509, 2017.

Thrun, S. Is learning the N-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems (NIPS)*, pp. 640–646, 1996.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. FeUdal networks for hierarchical reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 3540–3549, 2017.

Watkins, C. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, England, 1989.

Watkins, C. and Dayan, P. Q-learning. *Machine Learning*, 8:279–292, 1992.

Zhang, J., Springenberg, J. T., Boedecker, J., and Burgard, W. Deep reinforcement learning with successor features for navigation across similar environments. *CoRR*, abs/1612.05533, 2016.