Successor Features for Transfer in Reinforcement Learning

André Barreto [†††]
andrebarreto@google.com

Rémi Munos [†]
munos@google.com

Tom Schaul [†]
schaul@google.com

David Silver [†]
davidsilver@google.com

[†] Google DeepMind
London, United Kingdom

[††] Laboratório Nacional de
Computação Científica
Petrópolis, RJ, Brazil

Abstract

Transfer in reinforcement learning refers to the notion that generalization should occur not only within a task but also across tasks. Our focus is on transfer where the reward functions vary across tasks while the environment's dynamics remain the same. The method we propose rests on two key ideas: "successor features," a value function representation that decouples the dynamics of the environment from the rewards, and "generalized policy improvement," a generalization of dynamic programming's policy improvement step that considers a set of policies rather than a single one. Put together, the two ideas lead to an approach that integrates seamlessly within the reinforcement learning framework and allows transfer to take place between tasks without any restriction. The proposed method also provides performance guarantees for the transferred policy even before any learning has taken place. We derive two theorems that set our approach in firm theoretical ground and present experiments that show that it successfully promotes transfer in practice.

1 Introduction

Reinforcement learning (RL) provides a framework for the development of situated agents that learn how to behave while interacting with the environment [21]. The basic RL loop is defined in an abstract way so as to capture only the essential aspects of such an interaction: an agent receives observations and selects actions to maximize a reward signal. This setup is generic enough to describe tasks of different levels of complexity that may unroll at distinct time scales. For example, in the task of driving a car, an action can be to turn the wheel, to make a right turn, or to drive to a given location.

Clearly, from the point of view of the designer it is desirable to describe a task at the highest level of abstraction possible. However, by doing so one may overlook behavioral patterns and inadvertently make the task more difficult than it really is. The action of driving to a location clearly encompasses the action of making a right turn, which in turn encompasses the action of turning the wheel. In learning how to drive an agent should be able to identify and to exploit such interdependencies. More generally, the agent should be able to break a task in smaller subtasks and use knowledge accumulated in any subset of those to speed up learning in related tasks. This process of leveraging knowledge acquired in one task to improve performance on another task is usually referred to as *transfer*.

Transfer in reinforcement learning can be defined in many different ways, but in general it refers to the notion that generalization should occur not only within a task but also across tasks [23]. In this paper we look at one specific type of transfer, namely, when the subtasks involved correspond to different reward functions defined in the same environment. This setup is flexible enough to allow transfer to happen at different levels. In particular, since rewards are a generic device to define the agent's objective, by appropriately defining them one can induce different task decompositions. For instance, the type of hierarchical decomposition involved in the driving example above can be induced by changing the frequency at which rewards are delivered to the agent: a positive reinforcement can be given after each maneuver that is well executed or only at the final destination. It is not difficult to see that one can also decompose a task in subtasks that are fairly independent of each other or whose dependency is strictly temporal (that is, when the tasks must be executed in a certain order but no single task is clearly "contained" within another).

These types of task decomposition potentially allow the agent to tackle more complex problems than would be possible were the tasks modeled as a single monolithic challenge. However, in order to exploit this structure to its full extent the agent should have an explicit mechanism to promote transfer between tasks. Ideally, we want a transfer approach to have two important properties. First, the flow of information between tasks should not be dictated by a rigid diagram that reflects the relationship between the tasks themselves, such as hierarchical or temporal dependencies. On the contrary, information should be exchanged between tasks whenever useful. Second, rather than being posed as a separate problem, transfer should be integrated into the RL framework as much as possible, preferably in a way that is almost transparent to the agent.

In this paper we propose an approach to implement transfer that has the two properties above. Our method builds on two basic ideas that complement each other. The first one is a generalization of a concept proposed by Dayan [7] called *successor representation*. As the name suggests, in this representation scheme each state is described by a prediction about the future occurrence of all other states under a fixed policy. We present a generalization of Dayan's idea which extends the original scheme to continuous spaces and also facilitates the incorporation of function approximation. We call the resulting scheme *successor features*. As will be shown, successor features lead to a representation of the value function that naturally decouples the dynamics of the environment from the rewards, which makes them particularly suitable for transfer.

In order to actually put transfer into effect with successor features, we present two theoretical results that provide the foundation of our approach. The first one is a generalization of Bellman's [4] classic policy improvement theorem that extends the original result from one to multiple decision policies. This result shows how knowledge about a set of tasks can be transferred to a new task in a way that is completely integrated with reinforcement learning. It also provides performance guarantees on the new task *before* any learning has taken place. The second theoretical result is a theorem that formalizes the notion that an agent should be able to perform well on a task if it has seen a similar task before—something clearly desirable in the context of transfer. Combined, the two results above not only set our approach in firm ground but also outline the mechanics of how to actually implement transfer. We build on this knowledge to propose a concrete method and evaluate it in experiments that illustrate the benefits of transfer in practice.

2 Background and Problem Formulation

We consider the framework of RL outlined in the introduction: an agent interacts with an environment and selects actions in order to maximize the expected amount of reward received in the long run [21]. As usual, we assume that this interaction can be modeled as a *Markov decision process* (MDP, Puterman, [17]). An MDP is defined as a tuple $M \equiv (\mathcal{S}, \mathcal{A}, p, r, \gamma)$. The sets $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively; here we assume that $\mathcal{S}$ and $\mathcal{A}$ are finite whenever such an assumption facilitates the presentation, but most of the ideas readily extend to continuous spaces. For each $s \in \mathcal{S}$ and $a \in \mathcal{A}$ the function $p(\cdot|s, a)$ gives the next-state distribution upon taking action a in state s. We will often refer to $p(\cdot|s, a)$ as the *dynamics* of the MDP. The reward received at transition $s \xrightarrow{a} s'$ is given by $r(s, a, s')$; usually one is interested in the expected reward resulting from the execution of a in s, which is given by $r(s, a) = E_{S' \sim p(\cdot|s,a)}[r(s, a, S')]$. The discount factor $\gamma \in [0, 1)$ gives smaller weights to rewards received further in the future.

The goal of the agent in RL is to find a policy π—a mapping from states to actions—that maximizes the expected discounted sum of rewards, also called the *return* $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}$. One way to address this problem is to use methods derived from *dynamic programming* (DP), which heavily rely on the concept of a *value function* [17]. The *action-value function* of a policy π is defined as

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

where $E^\pi[\cdot]$ denotes expected value when following policy π. Once the action-value function of a particular policy π is known, we can derive a new policy π' which is *greedy* with respect to $Q^\pi(s,a)$, that is, $\pi'(s) \in \text{argmax}_a Q^\pi(s,a)$. Policy π' is guaranteed to be at least as good as (if not better than) policy π. These two steps, *policy evaluation* and *policy improvement*, define the basic mechanics of RL algorithms based on DP; under certain conditions their successive application leads to an optimal policy π^* that maximizes the expected return from every state in $\mathcal{S}$ [21].

As mentioned, in this paper we are interested in the problem of *transfer*. Here we adopt the following definition: given two sets of tasks T and T' such that $T \subset T'$, after being exposed to T' the agent should perform no worse, and preferably better, than it would had it been exposed to T only. Note that T can be the empty set. In this paper a task will be a specific reward function $r(s,a)$ for a given MDP. In Section 4 we will revisit this definition and make it more formal, and we will also clarify the measure used to compare performance. Before doing that, though, we will present a core concept for this paper whose interest is not restricted to transfer learning.

3 Successor Features

In this section we present the concept that will serve as a cornerstone for the rest of the paper. We start by presenting a simple reward model and then show how it naturally leads to a generalization of Dayan's [7] successor representation (SR).

Suppose that the one-step expected reward associated with state-action pair (s,a) is given by

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

where $\boldsymbol{\phi}(s,a) \in \mathbb{R}^d$ are features of (s,a) and $\mathbf{w} \in \mathbb{R}^d$ are weights. Supposing that (2) is true is not restrictive since we are not making any assumptions about $\boldsymbol{\phi}(s,a)$: if we have $\boldsymbol{\phi}_i(s,a) = r(s,a)$ for some i, for example, we can clearly recover any reward function exactly. To simplify the notation, let $\boldsymbol{\phi}_t = \boldsymbol{\phi}(s_t, a_t)$. Then, by simply rewriting the definition of the action-value function in (1) we have

$$\begin{aligned} Q^\pi(s,a) &= E^\pi \left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \,|\, S_t = s, A_t = a \right] \\ &= E^\pi \left[\boldsymbol{\phi}_{t+1}^\top \mathbf{w} + \gamma \boldsymbol{\phi}_{t+2}^\top \mathbf{w} + \gamma^2 \boldsymbol{\phi}_{t+3}^\top \mathbf{w} + ... \,|\, S_t = s, A_t = a \right] \\ &= E^\pi \left[\textstyle\sum_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\phi}_{i+1} \,|\, S_t = s, A_t = a \right]^\top \mathbf{w} = \boldsymbol{\psi}^\pi(s,a)^\top \mathbf{w}. \end{aligned} \tag{3}$$

We call $\boldsymbol{\psi}^\pi(s,a) \equiv E^\pi[\sum\limits_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\phi}_{i+1} | S_t = s, A_t = a]$ the *successor features* (SFs) of (s,a) under π.

The i$^{\text{th}}$ component of $\boldsymbol{\psi}^\pi(s,a)$ gives the discounted sum of $\boldsymbol{\phi}_i$ when following policy π starting from (s,a). In the particular case where $\mathcal{S}$ and $\mathcal{A}$ are finite and $\boldsymbol{\phi}$ is a tabular representation of $\mathcal{S} \times \mathcal{A}$—that is, $\boldsymbol{\phi}(s,a)$ is a "one-hot" vector in $\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$—$\boldsymbol{\psi}^\pi(s,a)$ is the discounted sum of occurrences of each state-action pair under π. This is essentially the concept of SR extended from the space $\mathcal{S}$ to the set $\mathcal{S} \times \mathcal{A}$ [7]. One of the points here is precisely to generalize SR to be used with function approximation, but the exercise of deriving the concept as above provides insights already in the tabular case. To see why this is so, note that in the tabular case the entries of $\mathbf{w} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ are the function $r(s,a)$ and suppose that $r(s,a) \neq 0$ in only a small subset $\mathcal{W} \subset \mathcal{S} \times \mathcal{A}$. From (2) and (3), it is clear that the cardinality of $\mathcal{W}$, and not of $\mathcal{S} \times \mathcal{A}$, is what effectively defines the dimension of the representation $\boldsymbol{\psi}^\pi$, since there is is no point in having $d > |\mathcal{W}|$. Although this fact is hinted at in Dayan's [7] paper, it becomes much more apparent when we look at SR as a particular case of SFs.

SFs extend Dayan's [7] SR in two ways. First, the concept readily applies to continuous state and action spaces without any modification. Second, by explicitly casting (2) and (3) as inner products involving feature vectors, SFs make it evident how to incorporate function approximation, since these vectors can clearly be learned from data. For reasons that will become apparent shortly, in this paper we are mostly interested in learning $\mathbf{w}$ and $\boldsymbol{\psi}^{\pi}(s,a)$. The extension to the scenario where $\boldsymbol{\phi}(s,a)$ must also be learned should not be difficult, though, and will also be discussed.

The SFs $\boldsymbol{\psi}^{\pi}$ are a way of summarizing the dynamics induced by π in a given environment. As shown in (3), this allows for a modular representation of Q^{π} in which the MDP's dynamics are decoupled from its rewards, which are captured by $\mathbf{w}$. One potential benefit of having such a decoupled representation is that only the relevant module must be relearned when either the dynamics or the reward changes. We come back to this point after describing how exactly each module can be learned.

The representation in (3) requires two components to be learned, $\mathbf{w}$ and $\boldsymbol{\psi}^{\pi}$. Since the latter is the expected discounted sum of $\boldsymbol{\phi}$ under π, we either know $\boldsymbol{\phi}$ or must learn it as well. Note that $r(s,a) \approx \boldsymbol{\phi}(s,a)^{\top}\mathbf{w}$ is a supervised learning problem, so one can resort to one of the many well-understood techniques from the field to learn $\mathbf{w}$ (and potentially $\boldsymbol{\phi}$, too)[8]. As for $\boldsymbol{\psi}^{\pi}$, we note that

$$\boldsymbol{\psi}^{\pi}(s,a) = \boldsymbol{\phi}_{t+1} + \gamma E^{\pi}[\boldsymbol{\psi}^{\pi}(S_{t+1}, \pi(S_{t+1})) \,|\, S_t = s, A_t = a], \tag{4}$$

that is, SFs satisfy a Bellman equation in which $\boldsymbol{\phi}_i$ play the role of rewards—something also noted by Dayan [7] regarding SR. Therefore, in principle *any* RL method can be used to compute $\boldsymbol{\psi}^{\pi}$ [21, 6].[1]

Now that we have described how to learn $\boldsymbol{\psi}^{\pi}$ and $\mathbf{w}$, we can resume the discussion on the potential benefits of doing so. Suppose that we have learned the value function of a given policy π using the scheme shown in (3). It should be clear that whenever the reward function $r(s,a)$ changes we only need to learn a new $\mathbf{w}$. Similarly, whenever the dynamics of the MDP $p(\cdot|s,a)$ change we can relearn $\boldsymbol{\psi}^{\pi}$ while retaining in $\mathbf{w}$ the information that has remained the same. But SFs may be useful even if we restrict ourselves to the setting usually considered in RL, in which $r(s,a)$ and $p(\cdot|s,a)$ are fixed. Note that the dynamics that determine Q^{π}, and thus $\boldsymbol{\psi}^{\pi}$, depend on both $p(\cdot|s,a)$ and π. Hence, even when the former is fixed, the possibility of only relearning ψ^{π} may be advantageous when π is changing, as is the case in the usual RL loop, since information regarding the reward function is preserved in $\mathbf{w}$. This helps explain the good performance of SR in Dayan's [7] experiments and may also serve as an argument in favor of adopting SFs as a general approximation scheme for RL. However, in this paper we focus on a scenario where the decoupled value-function approximation provided by SFs is exploited to its full extent, as we discuss next.

4 Transfer Via Successor Features

In this section we return to our discussion about transfer in RL. As described, we are interested in the scenario where all components of an MDP are fixed, except for the reward function. One way of formalizing this model is through (2): if we suppose that $\boldsymbol{\phi} \in \mathbb{R}^d$ is fixed, any $\mathbf{w} \in \mathbb{R}^d$ gives rise to a new MDP. Based on this observation, we define

$$\mathcal{M}^{\phi}(\mathcal{S}, \mathcal{A}, p, \gamma) \equiv \{M(\mathcal{S}, \mathcal{A}, p, r, \gamma) \,|\, r(s,a) = \boldsymbol{\phi}(s,a)^{\top}\mathbf{w}, \text{ with } \mathbf{w} \in \mathbb{R}^d\}, \tag{5}$$

that is, $\mathcal{M}^{\phi}$ is the set of MDPs induced by $\boldsymbol{\phi}$ through all possible instantiations of $\mathbf{w}$. Since what differentiates the MDPs in $\mathcal{M}^{\phi}$ is essentially the agent's goal, we will refer to $M_i \in \mathcal{M}^{\phi}$ as a *task*. The assumption is that we are interested in solving (a subset of) the tasks in the environment $\mathcal{M}^{\phi}$.

Unlike (2), which is not restrictive at all, supposing that a family of tasks of interest fit in the definition (5) will in general be a restrictive assumption.[2] Despite the fact that similar assumptions have been made in the literature [1], we now describe some illustrative examples that suggest that our formulation of $\mathcal{M}^{\phi}$ is a natural way of modeling some scenarios of interest.

Perhaps the best way to motivate (5) is to note that some aspects of real environments which we clearly associate with specific features change their appeal over time. Think for example how much the desirability of water or food changes depending on whether an animal is thirsty or hungry. One way to model this type of preference shifting, which should probably occur with some types of

[1] Yao et al. [25] discuss the properties of (3) when $\mathbf{w}$ and $\boldsymbol{\psi}^{\pi}$ are approximations learned in one specific way. However, the ideas presented here are not tightly coupled with any formulation of the two learning sub-problems.

[2] It is not restrictive when d is greater than or equal to the number of MDPs we are interested in or when $d \geq |\mathcal{S}||\mathcal{A}|$. In the first case we can simply make the i$^{\text{th}}$ dimension of $\boldsymbol{\phi}(s,a)$ equal to the reward function of the i$^{\text{th}}$ MDP. As for the second case, if we can afford to use SR—that is, if $d = |\mathcal{S}||\mathcal{A}|$—$\mathcal{M}^{\phi}$ will include all possible reward functions over $\mathcal{S} \times \mathcal{A}$.

artificial agents as well, is to suppose that the vector $\mathbf{w}$ appearing in (2) reflects the taste of the agent at any given point in time. For a more concrete example, imagine that the agent's goal is to produce and sell a combination of goods whose production line is relatively stable but whose prices vary considerably over time. In this case updating the price of the products corresponds to picking a new $\mathbf{w}$. Another intuitive example, which we will explore in the experimental section, is to imagine that the agent is navigating in a fixed environment but the goal location changes from time to time.

In all the examples above it is desirable for the agent to build on previous experience to improve its performance on a new setup. More concretely, if the agent knows good policies for the set of tasks $\mathcal{M} \equiv \{M_1, M_2, ..., M_n\}$, with $M_i \in \mathcal{M}^\phi$, it should be able to leverage this knowledge somehow to improve its behavior on a new task M_{n+1}—that is, it should perform better than it would had it been exposed to only a subset of the original tasks, $\mathcal{M}' \subset \mathcal{M}$. Here we assess the performance of an agent on M_{n+1} based on the value function of the policy computed by the agent after receiving the new $\mathbf{w}_{n+1}$ but *before* any learning has taken place in M_{n+1}.[3] More precisely, suppose that an agent agt has performed a number of transitions in each one of the tasks $M_i \in \mathcal{M}'$. Based on this experience and on the new $\mathbf{w}_{n+1}$, agt computes a policy π' that will define its initial behavior in M_{n+1}. Now, if we repeat the experience replacing $\mathcal{M}'$ with $\mathcal{M}$, the value of the resulting policy π should be such that $Q^\pi(s,a) \geq Q^{\pi'}(s,a)$ for all $s \in \mathcal{S}$ and all $a \in \mathcal{A}$.

Now that our setup is clear we can start to describe our solution for the transfer problem described above. We do so in two stages. First, we present a generalization of DP's notion of policy improvement whose interest may go beyond the current work. We then show how SFs can be used to implement this generalized form of policy improvement in an efficient and elegant way.

4.1 Generalized Policy Improvement

One of the key results in DP is Bellman's [4] policy improvement theorem. Basically, the theorem states that acting greedily with respect to a policy's value function gives rise to another policy whose performance is no worse than the former's. This is the driving force behind DP, and any RL algorithm that uses the notion of a value function is exploiting Bellman's result in one way or another.

In this section we extend the policy improvement theorem to the scenario where the new policy is to be computed based on the value functions of a *set* of policies. We show that this extension can be done in a very natural way, by simply acting greedily with respect to the maximum over the value functions available. Our result is summarized in the theorem below.

Theorem 1. (Generalized Policy Improvement) *Let π_1, π_2, ..., π_n be n decision policies and let $\tilde{Q}^{\pi_1}$, $\tilde{Q}^{\pi_2}$, ..., $\tilde{Q}^{\pi_n}$ be approximations of their respective action-value functions such that*

$$|Q^{\pi_i}(s,a) - \tilde{Q}^{\pi_i}(s,a)| \leq \epsilon \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}, \text{ and } i \in \{1, 2, ..., n\}. \quad (6)$$

Define $\pi(s) \in \mathrm{argmax}_a \max_i \tilde{Q}^{\pi_i}(s,a)$. Then,

$$Q^\pi(s,a) \geq \max_i Q^{\pi_i}(s,a) - \frac{2}{1-\gamma}\epsilon \quad (7)$$

for any $s \in \mathcal{S}$ and any $a \in \mathcal{A}$, where Q^π is the action-value function of π.

The proofs of our theoretical results are in Appendix A. As one can see, our theorem covers the case in which the policies' value functions are not computed exactly, either because function approximation is used or because some exact algorithm has not be run to completion. This error is captured by ϵ in (6), which of course re-appears as a "penalty" term in the lower bound (7). Such a penalty is inherent to the presence of approximation in RL, and in fact it is identical to the penalty incurred in the single-policy case (see *e.g.* Bertsekas and Tsitsiklis's Proposition 6.1 [5]).

In order to contextualize our result within the broader scenario of DP, suppose for a moment that $\epsilon = 0$. In this case Theorem 1 states that π will perform no worse than *all* of the policies $\pi_1, \pi_2, ..., \pi_n$. This is interesting because in general $\max_i Q^{\pi_i}$—the function used to induce π—is not the value function of any particular policy. It is not difficult to see that π will be strictly better than all previous policies if $\mathrm{argmax}_i \max_a \tilde{Q}^{\pi_i}(s,a) \cap \mathrm{argmax}_i \max_a \tilde{Q}^{\pi_i}(s',a) = \emptyset$ for any $s, s' \in \mathcal{S}$, that is, if no

[3] Of course $\mathbf{w}_{n+1}$ can, and will be, learned, as discussed in Section 4.2 and illustrated in Section 5. Here though we assume that $\mathbf{w}_{n+1}$ is given to make the definition of our performance criterion as clear as possible.

single policy dominates all other policies. If one policy does dominate all others, Theorem 1 reduces to the original policy improvement theorem. Note that this will always be the case if one of the optimal policies π^* belongs to the set $\{\pi_1, \pi_2, ..., \pi_n\}$.

If we consider the usual DP loop, in which policies of increasing performance are computed in sequence, our result is not of much use because the most recent policy will always dominate all others. Another way of putting it is to say that after Theorem 1 is applied once adding the resulting π to the set $\{\pi_1, \pi_2, ..., \pi_n\}$ will reduce the next improvement step to standard policy improvement, and thus the policies $\pi_1, \pi_2, ..., \pi_n$ can be simply discarded.

There are however two situations in which our result may be of interest. One is when we have many policies π_i being evaluated in parallel. In this case Theorem 1 provides a principled strategy for combining these policies. This could be used for example as an initialization scheme for policy iteration when it is possible to evaluate many policies simultaneously. This possibility may also be useful in RL when one consider the multi-agent setting. The other situation in which our result may be useful is when the underlying MDP changes, as we discuss next.

4.2 Generalized Policy Improvement with Successor Features

We start this section by extending our notation slightly to make it easier to refer to the quantities involved in transfer learning. Let M_i be a task in $\mathcal{M}^\phi$ defined by $\mathbf{w}_i \in \mathbb{R}^d$. We will use π_i^* to refer to an optimal policy of MDP M_i and use $Q_i^{\pi_i^*}$ to refer to its value function. The value function of π_i^* when executed in $M_j \in \mathcal{M}^\phi$ will be denoted by $Q_j^{\pi_i^*}$.

Suppose now that an agent agt has computed optimal policies for the tasks $M_1, M_2, ..., M_n \in \mathcal{M}^\phi$. Suppose further that when exposed to a new task M_{n+1} the agent computes $Q_{n+1}^{\pi_i^*}$—the value functions of the policies π_i^* under the new reward function induced by $\mathbf{w}_{n+1}$. In this case, applying Theorem 1 to the newly-computed set of value functions $\{Q_{n+1}^{\pi_1^*}, Q_{n+1}^{\pi_2^*}, ..., Q_{n+1}^{\pi_n^*}\}$ will give rise to a policy that performs at least as well as a policy computed based on any subset of the set above, including the empty set (except of course in the unlikely event that one starts with a randomly-generated policy that performs well). Thus, this strategy satisfies our definition of successful transfer.

There is a caveat, though. Why would one waste time computing the value functions of $\pi_1^*, \pi_2^*, ...,$ π_n^*, whose performance in M_{n+1} may be mediocre, if the same amount of resources can be allocated to compute a sequence of n policies with increasing performance? This is where SFs come into play. Suppose that we have learned the functions $Q^{\pi_i^*}$ using the approximation scheme shown in (3). Now, if the reward changes to $r_{n+1}(s, a) = \boldsymbol{\phi}(s, a)^\top \mathbf{w}_{n+1}$, as long as we have $\mathbf{w}_{n+1}$ we can compute the new value function of π_i^* by simply making $Q_{n+1}^{\pi_i^*}(s, a) = \boldsymbol{\psi}^{\pi_i^*}(s, a)^\top \mathbf{w}_{n+1}$. This reduces the computation of all $Q_{n+1}^{\pi_i^*}$ to the much simpler supervised learning problem of computing $\mathbf{w}_{n+1}$.

Once $Q_{n+1}^{\pi_i^*}$ have been computed, we can apply Theorem 1 to derive a policy π whose performance on M_{n+1} is no worse than the performance of $\pi_1^*, \pi_2^*, ..., \pi_n^*$ on the same task. A question that arises in this case is whether we can provide stronger guarantees on the performance of π by exploiting the structure shared by the tasks in $\mathcal{M}^\phi$. The following theorem answers this question in the affirmative.

Theorem 2. *Let $M_i \in \mathcal{M}^\phi$ and let $Q_i^{\pi_j^*}$ be the value function of an optimal policy of $M_j \in \mathcal{M}^\phi$ when executed in M_i. Given approximations $\{\tilde{Q}_i^{\pi_1^*}, \tilde{Q}_i^{\pi_2^*}, ..., \tilde{Q}_i^{\pi_n^*}\}$ such that*

$$\left| Q_i^{\pi_j^*}(s, a) - \tilde{Q}_i^{\pi_j^*}(s, a) \right| \leq \epsilon \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}, \text{ and } j \in \{1, 2, ..., n\}, \tag{8}$$

let $\pi(s) \in \mathrm{argmax}_a \max_j \tilde{Q}_i^{\pi_j^*}(s, a)$. Finally, let $\boldsymbol{\phi}_{\max} = \max_{s,a} ||\boldsymbol{\phi}(s, a)||$, where $|| \cdot ||$ is the norm induced by the inner product adopted. Then,

$$Q_i^{\pi_i^*}(s, a) - Q_i^{\pi}(s, a) \leq \frac{2}{1 - \gamma} \left(\boldsymbol{\phi}_{\max} \min_j ||\mathbf{w}_i - \mathbf{w}_j|| + \epsilon \right). \tag{9}$$

Note that we used "M_i" rather than "M_{n+1}" in the theorem's statement to remove any suggestion of order among the tasks. This also makes it explicit that the result also applies when $i \in \{1, 2, ..., n\}$. Theorem 2 is a specialization of Theorem 1 for the case where the set of value functions used to compute π are associated with tasks in the form of (5). As such, it provides stronger guarantees than its precursor: instead of comparing the performance of π with that of the previously-computed policies π_j, Theorem 2 quantifies the loss incurred by following π as opposed to of one of M_i's optimal policies.

As shown in (9), the loss $Q_i^{\pi_i^*}(s,a) - Q_i^{\pi}(s,a)$ is upper-bounded by two terms. As before, $2\epsilon/(1-\gamma)$ is a "penalty" term that shows up in the bound due to the use of approximations $\tilde{Q}_i^{\pi_j^*}$ instead of the true value functions $Q_i^{\pi_j^*}$. The term $2\boldsymbol{\phi}_{\max}\min_j||\mathbf{w}_i - \mathbf{w}_j||/(1-\gamma)$ is of more interest here because it reflects the structure of $\mathcal{M}^{\phi}$. This term is a multiple of the distance between $\mathbf{w}_i$, the vector describing the task we are currently interested in, and the closest $\mathbf{w}_j$ for which we have computed a policy. This formalizes the intuition that the agent should perform well in task $\mathbf{w}_i$ if it has solved a similar task before. More generally, the term in question relates the concept of distance in $\mathbb{R}^d$ with difference in performance in $\mathcal{M}^{\phi}$, which allows for interesting extrapolations. For example, if we assume that the tasks $\mathbf{w}_j$ are sampled from a distribution over $\mathbb{R}^d$, it might be possible to derive probabilistic performance guarantees whose probability of failure goes to zero as $n \to \infty$.

Although Theorem 2 is inexorably related to the characterization of $\mathcal{M}^{\phi}$ in (5), it does not depend on the definition of SFs in any way. Here SFs are the *mechanism* used to efficiently apply the protocol suggested by Theorem 2. When SFs are used the value function approximations are given by $\tilde{Q}_i^{\pi_j^*}(s,a) = \tilde{\boldsymbol{\psi}}^{\pi_j^*}(s,a)^{\top}\tilde{\mathbf{w}}_i$. The modules $\tilde{\boldsymbol{\psi}}^{\pi_j^*}$ are computed and stored when the agent is learning the tasks M_j; when faced with a new task M_i the agent computes an approximation of $\mathbf{w}_i$, which is a supervised learning problem, and then uses the policy π defined in Theorem 2 to learn $\tilde{\boldsymbol{\psi}}^{\pi_i^*}$. Note that we do not assume that either $\boldsymbol{\psi}^{\pi_j^*}$ or $\mathbf{w}_i$ is computed exactly: the effect of errors in $\tilde{\boldsymbol{\psi}}^{\pi_j^*}$ and $\tilde{\mathbf{w}}_i$ in the approximation of $Q_i^{\pi_j^*}(s,a)$ is accounted for by the term ϵ appearing in (8). As shown in (9), if ϵ is small and the agent has seen enough tasks the performance of π on M_i should already be good, which suggests that it will also speed up the process of learning $\tilde{\boldsymbol{\psi}}^{\pi_i^*}$. In the next section we verify empirically how these effects manifest in practice.

5 Experiments

In this section we use experiments to illustrate how the transfer promoted by the combination of generalized policy iteration and SFs actually takes place in practice. In order to do so we introduce a generalized version of a classic RL task known as the "puddle world" [20]. The puddle world is a simple two-dimensional problem with a goal position and two elliptical "puddles," one vertical and one horizontal [20]. The four actions available move the agent up, down, left, or right. An action fails with probability 0.1, in which case an action selected uniformly at random is executed. The objective is to reach the goal while avoiding the puddles along the way.

We generalized the puddle world task by letting the position of the puddles and of the goal state to change at arbitrary time steps. More specifically, we implemented the task as a 15×15 grid and restricted the position of the elements to a subset of the cells: the goal is only allowed to be in one of the four corners and the two puddles are restricted to the set $\{3,5,7,9,11\} \times \{3,5,7,9,11\}$. This gives rise to 2500 possible configurations of the task, which is our set $\mathcal{M}^{\phi}$. Following (5), the reward function for the i^{th} task was defined as $r(s,a) = \boldsymbol{\phi}(s,a)^{\top}\mathbf{w}_i$. Here $\boldsymbol{\phi}(s,a)$ is a binary vector in $\mathbb{R}^{54}$ that indicates whether the state that (s,a) most likely leads to corresponds to a puddle or a goal. Specifically, if the i^{th} entry of $\boldsymbol{\phi}(s,a)$ is associated with, say, one of the possible 25 locations of the horizontal puddle, it will be equal to 1 if and only if (s,a) has as its most likely outcome the state at that location. The goal and puddles that are present in the i^{th} task are indicated by three nonzero elements in $\mathbf{w}_i$: a $+1$ entry associated with the goal and a -1 entry associated with each puddle.

We focus on the online RL scenario where the agent must learn while interacting with the environment $\mathcal{M}^{\phi}$. The task changes at every k transitions, with a new $\mathbf{w}_i$ selected uniformly at random from the set described above. We adopted Watkins and Dayan's [24] Q-learning as our basic algorithm and combined it with different representation schemes to show their potential for transfer. In particular, we compared four versions of Q-learning: using a tabular representation (QL), using a tabular representation that is reinitialized to zero whenever the task changes (QLR), using SR, and using SFs. All versions of the algorithm used an ϵ-greedy policy to explore the environment, with $\epsilon = 0.15$ [21].

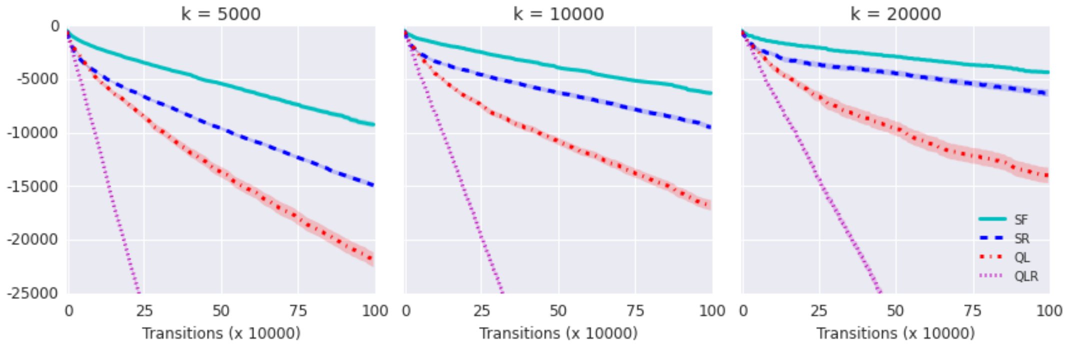

Figure 1: Cumulative return on the moving-puddles task. The parameter k above each plot indicates how often the reward function changes. Shadowed regions represent one standard error over 30 runs.

The SR and SF agents were implemented in the following way. The action-value function was represented as $\tilde{Q}^{\pi}(s,a) = \tilde{\boldsymbol{\psi}}_i^{\pi}(s,a)^{\top}\tilde{\mathbf{w}}$, where $\tilde{\boldsymbol{\psi}}_i^{\pi}(s,a)$ is associated with the i^{th} task. Both $\tilde{\boldsymbol{\psi}}_i^{\pi}$ and $\tilde{\mathbf{w}}$ were learned online. The former was learned using temporal-difference updates to solve (4) [21], while the latter was learned as a least-squares minimization of the difference between the two sides of (2). Every time the task changed the current $\tilde{\boldsymbol{\psi}}_i^{\pi}(s,a)$ was stored, a new $\tilde{\boldsymbol{\psi}}_{i+1}^{\pi}(s,a)$ was created, and $\mathbf{w}$ was reinitialized to $\mathbf{0}$. The agent followed a 0.15-greedy policy with respect to $\operatorname{argmax}_a \max_{j\in\{1,2,\ldots,i+1\}} \tilde{\boldsymbol{\psi}}_j^{\pi}(s,a)^{\top}\tilde{\mathbf{w}}$ (hence the policy π that induces (4) was constantly changing). In the case of SR, $\tilde{\boldsymbol{\psi}}_i^{\pi}(s,a)$ were vectors in $\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$, as usual (here $|\mathcal{S}||\mathcal{A}| = 900$). The SF agent received with each (s,a) the corresponding vector $\boldsymbol{\phi}(s,a)$. So, in this case $\tilde{\boldsymbol{\psi}}_i^{\pi}(s,a) \in \mathbb{R}^{54}$.

The results of our experiments are shown in Figure 1. Several interesting observations can be made regarding the figure. First, note that QLR is unable to learn the task. If we compare it with QL, the difference in performance suggests that in this environment starting from an actual value function $Q^{\pi}(s,a)$ leads to better results than starting from a function that is zero everywhere, regardless of the policy π associated with $Q^{\pi}(s,a)$. Also note that there is a clear improvement on the algorithms' performance as the interval k to change the reward increases, which is not surprising. However, the most important point to be highlighted here is that both SR and SF significantly outperform the standard version of Q-learning. This is an illustration of our theoretical results and a demonstration that the proposed approach is indeed able to successfully transfer knowledge across tasks. Finally, note that SF outperforms SR by a considerable margin. This is also expected: since the former uses a vector $\mathbf{w}$ that is in $\mathbb{R}^{54}$ rather than in $\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$, the nonzero elements of this vector will be updated whenever the agent encounters a puddle or a goal, regardless of the specific (s,a) pair that lead to that state. This shows how, unlike its precursor, SFs allows for generalization.

6 Related Work

In this paper we present SFs, a representation scheme for value functions, and show how they provide a natural framework for implementing transfer in RL. Both representation and transfer are active areas of research in RL; in what follows we briefly describe the previous work we consider to be more closely related to ours and take advantage of the relevant connections to point out some interesting directions for future research.

When it comes to representation, a lot of effort in previous research has been directed towards the development of methods to automatically compute good features to represent the value function [14, 10, 16, 15]. Many of these approaches build on the fact that, when S is finite, the value function of a policy π is given by $\mathbf{v}^{\pi} = \sum_{i=0}^{\infty}(\gamma\mathbf{P}^{\pi})^i\mathbf{r}^{\pi}$, where $p_{ij}^{\pi} = p(s_j|s_i,\pi(s_i))$ and $r_i^{\pi} = r(s_i,\pi(s_i))$. The idea is to exploit the structure in the definition of $\mathbf{v}^{\pi}$ to replace the set of vectors $(\gamma\mathbf{P}^{\pi})^i\mathbf{r}^{\pi} \in \mathbb{R}^{|S|}$, which is infinite, by a properly defined basis whose cardinality is preferably smaller than $|S|$. If we adopt the reward model in (2), we can rewrite the previous expression as $\mathbf{v}^{\pi} = \sum_{i=0}^{\infty}(\gamma\mathbf{P}^{\pi})^i\boldsymbol{\Phi}^{\pi}\mathbf{w}$, where $\boldsymbol{\Phi}$ is a vector in $\mathbb{R}^{|S|\times d}$ whose i^{th} row is $\boldsymbol{\phi}(s_i,\pi(s_i))$. If we then define $\mathbf{\Psi}^{\pi} = \sum_{i=0}^{\infty} (\gamma \mathbf{P}^{\pi})^{i} \mathbf{\Phi}^{\pi}$, it should be clear that the i$^{\text{th}}$ row of $\mathbf{\Psi}^{\pi}$ is $\psi^{\pi}(s_i, \pi(s_i))$. This shows that $\mathbf{\Psi}^{\pi}$ arises as a natural basis when (2) is adopted, which is neither very surprising nor very useful, since $\mathbf{\Psi}^{\pi}$ lives in $\mathbb{R}^{|S| \times d}$. What is perhaps more interesting is the observation that, since the definitions of $\mathbf{v}^{\pi}$ and $\mathbf{\Psi}^{\pi}$ are very similar, the methods cited above could in principle also be used to find good features to represent $\mathbf{\Psi}^{\pi}$ itself.

Although the methods above decouple the construction of features from the actual RL problem, it is also possible to tackle both problems concomitantly, using general nonlinear function approximators to incrementally learn $\boldsymbol{\psi}^{\pi}(s, a)$ [12]. Another interesting possibility is the definition of a clear protocol to also learn $\boldsymbol{\phi}(s, a)$, which is closely related to the problem known as "multi-task feature learning" [1]. Here again the use of nonlinear approximators may be useful, since with them it may be possible to embed an arbitrary family of MDPs into a model $\mathcal{M}^{\phi}$ with the structure shown in (5) [11].

Still on the subject of representation, a scheme that also relates to SFs is Littman et al.'s [9] predictive state representation (PSR). PSRs are similar to SFs in the sense that they also have a prediction at the core of their representation. Unlike the latter, though, the former tries to use such predictions to summarize the dynamics of the entire environment rather than of a single policy π. A scheme that is perhaps closer to SFs is the value function representation sometimes adopted in inverse RL [13]. The scenario considered in this case is considerably different, though, since the focus is in finding a $\mathbf{w}$ that induces a predefined policy π.

We now turn our attention to previous work related to the use of SFs for transfer. As mentioned in the introduction, the problem of transfer has many definitions in the literature [23]. When we focus on the scenario considered here, in which the agent must perform well on a family of MDPs that differ only in the reward function, two approaches are possible. One of them is to learn a model of the MDPs' dynamics [3]. Another alternative, which is more in-line with our approach, is to summarize the experience using policies or value functions—which in some sense represent a "partial model" of the environment. Among these, Schaul et al.'s [18] *universal value function approximators* (UVFAs) are particularly relevant to our work. UVFAs extend the notion of value function to also include as an argument a representation of a goal. We note that the function $\max_j \tilde{\boldsymbol{\psi}}^{\pi_j^*}(s, a)^{\top} \mathbf{w}$ used in our generalized policy improvement framework can be seen as a function of s, a, and $\mathbf{w}$—the latter a generic way of representing a "goal." Thus, in some sense the approximation scheme proposed here *is* a UVFA, in which $\mathbf{w}$ corresponds to the learned goal embedding.

As discussed, one possible interpretation of the scenario studied here is that there is one main task that has been decomposed in many sub-tasks. This view of transfer highlights an interesting connection between our approach and temporal abstraction. In fact, if we look at $\boldsymbol{\psi}^{\pi}$ as instances of Sutton et al.'s [22] *options*, acting greedily with respect to the maximum over their value functions corresponds in some sense to planning at a higher level of temporal abstraction. This is the view adopted by Yao et al. [25], whose *universal option model* closely resembles our approach in some aspects. The main difference is that, unlike in our method, in Yao et al.'s [25] approach options are not used to learn new options.

7 Conclusion

This paper builds on two concepts, both of which are generalizations of previous ideas. The first one is *SFs*, a generalization of Dayan's [7] SR that extends the original definition from discrete to continuous spaces and also facilitates the incorporation of function approximation. The second concept is *generalized policy improvement*, formalized in Theorem 1. As the name suggests, this result extends Bellman's [4] classic policy improvement theorem from a single to multiple policies.

Although SFs and generalized policy improvement are of interest on their own, in this paper we focus on their combination to induce transfer. The resulting framework is an elegant extension of DP's basic setting that provides a solid foundation for transfer in RL. We derived a theoretical result, Theorem 2, that formalizes the intuition that an agent should perform well on a novel task if it has seen a similar task before. We also illustrated how this effect manifests in practice using experiments.

We believe the ideas presented in this paper lay out a general framework for transfer in RL. By specializing the basic components presented here one can build on our results to derive agents able to perform well across a wide variety of tasks, and thus handle environments of considerable complexity.

Acknowledgments

The authors would like to thank Joseph Modayil and Hado van Hasselt for the invaluable discussions during the development of the ideas described in this paper. We also thank Peter Dayan, Matt Botvinick, Marc Bellemare, and Guy Lever for all the excellent comments. Finally, we thank Dan Horgan and Alexander Pritzel for their help with the experiments.

References

[1] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, 2008.

[2] Mehran Asadi and Manfred Huber. Effective control knowledge transfer through learning skill and representation hierarchies. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2054–2059, 2007.

[3] Christopher G. Atkeson and J. Santamaria. A comparison of direct and model-based reinforcement learning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 4, pages 3557–3564, 1997.

[4] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[5] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[6] Justin A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49:233–246, 2002.

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

[8] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2002.

[9] Michael L. Littman, Richard S. Sutton, and Satinder Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1555–1561, 2001.

[10] Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, 8:2169–2231, 2007.

[11] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, 2013.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[13] Andrew Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 663–670, 2000.

[14] Ronald Parr, Christopher Painter-Wakefield, Lihong Li, and Michael Littman. Analyzing feature generation for value-function approximation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 737–744, 2007.

[15] Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 752–759, 2008.

[16] Marek Petrik. An analysis of laplacian methods for value function approximation in MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2574–2579, 2007.

[17] Martin L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[18] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal Value Function Approximators. In *International Conference on Machine Learning (ICML)*, 2015.

[19] Alexander L. Strehl and Michael L. Littman. A theoretical analysis of model-based interval estimation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 857–864, 2005.

[20] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1038–1044, 1996.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[22] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112: 181–211, August 1999.

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

[24] Christopher Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[25] Hengshuai Yao, Csaba Szepesvari, Richard S Sutton, Joseph Modayil, and Shalabh Bhatnagar. Universal option models. In *Advances in Neural Information Processing Systems (NIPS)*, pages 990–998, 2014.

A Proofs

Theorem 1. (Generalized Policy Improvement) *Let π_1, π_2, ..., π_n be n decision policies and let $\tilde{Q}^{\pi_1}$, $\tilde{Q}^{\pi_2}$, ..., $\tilde{Q}^{\pi_n}$ be approximations of their respective action-value functions such that*

$$|Q^{\pi_i}(s,a) - \tilde{Q}^{\pi_i}(s,a)| \le \epsilon \text{ for all } s \in S, a \in A, \text{ and } i \in \{1,2,...,n\}.$$

Define

$$\pi(s) \in \operatorname*{argmax}_a \max_i \tilde{Q}^{\pi_i}(s,a).$$

Then,

$$Q^{\pi}(s,a) \ge \max_i Q^{\pi_i}(s,a) - \frac{2}{1-\gamma}\epsilon$$

for any $s \in S$ and any $a \in A$, where Q^{π} is the action-value function of π.

Proof. To simplify the notation, let

$$Q_{\max}(s,a) = \max_i Q^{\pi_i}(s,a) \quad \text{and} \quad \tilde{Q}_{\max}(s,a) = \max_i \tilde{Q}^{\pi_i}(s,a).$$

We start by noting that for any $s \in S$ and any $a \in A$ the following holds:

$$|Q_{\max}(s,a) - \tilde{Q}_{\max}(s,a)| = |\max_i Q^{\pi_i}(s,a) - \max_i \tilde{Q}^{\pi_i}(s,a)| \le \max_i |Q^{\pi_i}(s,a) - \tilde{Q}^{\pi_i}(s,a)| \le \epsilon.$$

For all $s \in S$, $a \in A$, and $i \in \{1,2,...,n\}$ we have

$$\begin{aligned}
T^{\pi}\tilde{Q}_{\max}(s,a) &= r(s,a) + \gamma \sum_{s'} p(s'|s,a)\tilde{Q}_{\max}(s',\pi(s')) \\
&= r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_b \tilde{Q}_{\max}(s',b) \\
&\ge r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_b Q_{\max}(s',b) - \gamma\epsilon \\
&\ge r(s,a) + \gamma \sum_{s'} p(s'|s,a) Q_{\max}(s',\pi_i(s')) - \gamma\epsilon \\
&\ge r(s,a) + \gamma \sum_{s'} p(s'|s,a) Q^{\pi_i}(s',\pi_i(s')) - \gamma\epsilon \\
&= T^{\pi_i}Q^{\pi_i}(s,a) - \gamma\epsilon \\
&= Q^{\pi_i}(s,a) - \gamma\epsilon.
\end{aligned}$$

Since $T^{\pi}\tilde{Q}_{\max}(s,a) \geq Q^{\pi_i}(s,a) - \gamma\epsilon$ for any i, it must be the case that

$$\begin{aligned}
T^{\pi}\tilde{Q}_{\max}(s,a) &\geq \max_i Q^{\pi_i}(s,a) - \gamma\epsilon \\
&= Q_{\max}(s,a) - \gamma\epsilon \\
&\geq \tilde{Q}_{\max}(s,a) - \epsilon - \gamma\epsilon.
\end{aligned}$$

Let $e(s,a) = 1$ for all $s,a \in S \times A$. It is well known that $T^{\pi}(\tilde{Q}_{\max}(s,a) + ce(s,a)) = T^{\pi}\tilde{Q}_{\max}(s,a) + \gamma c$ for any $c \in \mathbb{R}$. Using this fact together with the monotonicity and contraction properties of the Bellman operator T^{π}, we have

$$\begin{aligned}
Q^{\pi}(s,a) &= \lim_{k\to\infty}(T^{\pi})^k\tilde{Q}_{\max}(s,a) \\
&\geq \tilde{Q}_{\max}(s,a) - \frac{1+\gamma}{1-\gamma}\epsilon \\
&\geq Q_{\max}(s,a) - \epsilon - \frac{1+\gamma}{1-\gamma}\epsilon.
\end{aligned}$$

□

Lemma 1. *Let* $\delta_{ij} = \max_{s,a}|r_i(s,a) - r_j(s,a)|$. *Then,*

$$Q_i^{\pi_i^*}(s,a) - Q_i^{\pi_j^*}(s,a) \leq \frac{2\delta_{ij}}{1-\gamma}.$$

Proof. To simplify the notation, let $Q_i^j(s,a) \equiv Q_i^{\pi_j^*}(s,a)$. Then,

$$\begin{aligned}
Q_i^i(s,a) - Q_i^j(s,a) &= Q_i^i(s,a) - Q_j^j(s,a) + Q_j^j(s,a) - Q_i^j(s,a) \\
&\leq |Q_i^i(s,a) - Q_j^j(s,a)| + |Q_j^j(s,a) - Q_i^j(s,a)|. \qquad (10)
\end{aligned}$$

Our strategy will be to bound $|Q_i^i(s,a) - Q_j^j(s,a)|$ and $|Q_j^j(s,a) - Q_i^j(s,a)|$. Note that $|Q_i^i(s,a) - Q_j^j(s,a)|$ is the difference between the value functions of two MDPs with the same transition function but potentially different rewards. Let $\Delta_{ij} = \max_{s,a}|Q_i^i(s,a) - Q_j^j(s,a)|$. Then, [4]

$$\begin{aligned}
|Q_i^i(s,a) - Q_j^j(s,a)| &= \left| r_i(s,a) + \gamma\sum_{s'} p(s'|s,a)\max_b Q_i^i(s',b) - r_j(s,a) - \gamma\sum_{s'} p(s'|s,a)\max_b Q_j^j(s',b)\right| \\
&= \left| r_i(s,a) - r_j(s,a) + \gamma\sum_{s'} p(s'|s,a)\left(\max_b Q_i^i(s',b) - \max_b Q_j^j(s',b)\right)\right| \\
&\leq |r_i(s,a) - r_j(s,a)| + \gamma\sum_{s'} p(s'|s,a)\left|\max_b Q_i^i(s',b) - \max_b Q_j^j(s',b)\right| \\
&\leq |r_i(s,a) - r_j(s,a)| + \gamma\sum_{s'} p(s'|s,a)\max_b\left|Q_i^i(s',b) - Q_j^j(s',b)\right| \\
&\leq \delta_{ij} + \gamma\Delta_{ij}. \qquad (11)
\end{aligned}$$

Since (11) is valid for any $s,a \in S \times A$, we have shown that $\Delta_{ij} \leq \delta_{ij} + \gamma\Delta_{ij}$. Solving for Δ_{ij} we get

$$\Delta_{ij} \leq \frac{1}{1-\gamma}\delta_{ij}. \qquad (12)$$

[4] We follow the steps of Strehl and Littman [19].

We now turn our attention to $|Q_j^j(s,a) - Q_i^j(s,a)|$. Following the previous steps, define $\Delta'_{ij} = \max_{s,a} |Q_i^i(s,a) - Q_i^j(s,a)|$. Then,

$$\begin{aligned}
|Q_j^j(s,a) - Q_i^j(s,a)| &= \left| r_j(s,a) + \gamma \sum_{s'} p(s'|s,a) Q_j^j(s', \pi_j^*(s')) - r_i(s,a) - \gamma \sum_{s'} p(s'|s,a) Q_i^j(s', \pi_j^*(s')) \right| \\
&= \left| r_i(s,a) - r_j(s,a) + \gamma \sum_{s'} p(s'|s,a) \left(Q_j^j(s', \pi_j^*(s')) - Q_i^j(s', \pi_j^*(s')) \right) \right| \\
&\leq |r_i(s,a) - r_j(s,a)| + \gamma \sum_{s'} p(s'|s,a) \left| Q_j^j(s', \pi_j^*(s')) - Q_i^j(s', \pi_j^*(s')) \right| \\
&\leq \delta_{ij} + \gamma \Delta'_{ij}.
\end{aligned}$$

Solving for Δ'_{ij}, as above, we get

$$\Delta'_{ij} \leq \frac{1}{1-\gamma} \delta_{ij}. \tag{13}$$

Plugging (12) and (13) back in (10) we get the desired result. □

Theorem 2. *Let $M_i \in \mathcal{M}^\phi$ and let $Q_i^{\pi_j^*}$ be the value function of an optimal policy of $M_j \in \mathcal{M}^\phi$ when executed in M_i. Given the set $\{\tilde{Q}_i^{\pi_1^*}, \tilde{Q}_i^{\pi_2^*}, ..., \tilde{Q}_i^{\pi_n^*}\}$ such that*

$$\left| Q_i^{\pi_j^*}(s,a) - \tilde{Q}_i^{\pi_j^*}(s,a) \right| \leq \epsilon \text{ for all } s \in S, a \in A, \text{ and } j \in \{1, 2, ..., n\},$$

let

$$\pi(s) \in \underset{a}{\operatorname{argmax}} \max_j \tilde{Q}_i^{\pi_j^*}(s,a).$$

Finally, let $\boldsymbol{\phi}_{\max} = \max_{s,a} ||\boldsymbol{\phi}(s,a)||$, where $||\cdot||$ is the norm induced by the inner product adopted. Then,

$$Q_i^*(s,a) - Q_i^\pi(s,a) \leq \frac{2}{1-\gamma} \left(\boldsymbol{\phi}_{\max} \min_j ||\mathbf{w}_i - \mathbf{w}_j|| + \epsilon \right).$$

Proof. The result is a direct application of Theorem 1 and Lemma 1. For any $j \in \{1, 2, ..., n\}$, we have

$$\begin{aligned}
Q_i^*(s,a) - Q_i^\pi(s,a) &\leq Q_i^*(s,a) - Q_i^{\pi_j^*}(s,a) + \frac{2}{1-\gamma}\epsilon && \text{(Theorem 1)} \\
&\leq \frac{2}{1-\gamma} \max_{s,a} |r_i(s,a) - r_j(s,a)| + \frac{2}{1-\gamma}\epsilon && \text{(Lemma 1)} \\
&= \frac{2}{1-\gamma} \max_{s,a} |\boldsymbol{\phi}(s,a)^\top \mathbf{w}_i - \boldsymbol{\phi}(s,a)^\top \mathbf{w}_j| + \frac{2}{1-\gamma}\epsilon \\
&= \frac{2}{1-\gamma} \max_{s,a} |\boldsymbol{\phi}(s,a)^\top (\mathbf{w}_i - \mathbf{w}_j)| + \frac{2}{1-\gamma}\epsilon \\
&\leq \frac{2}{1-\gamma} \max_{s,a} ||\boldsymbol{\phi}(s,a)|| \, ||\mathbf{w}_i - \mathbf{w}_j|| + \frac{2}{1-\gamma}\epsilon && \text{(Cauchy-Schwarz's inequality)} \\
&= \frac{2\boldsymbol{\phi}_{\max}}{1-\gamma} ||\mathbf{w}_i - \mathbf{w}_j|| + \frac{2}{1-\gamma}\epsilon.
\end{aligned}$$

□