Successor Feature Sets: Generalizing Successor Representations Across Policies

Kianté Brantley,[1] **Soroush Mehri,** [2] **Geoffrey J. Gordon** [2]
[1] University of Maryland College Park
[2] Microsoft Research
kdbrant@umd.edu, somehri@microsoft.com, ggordon@microsoft.com

Abstract

Successor-style representations have many advantages for reinforcement learning: for example, they can help an agent generalize from past experience to new goals, and they have been proposed as explanations of behavioral and neural data from human and animal learners. They also form a natural bridge between model-based and model-free RL methods: like the former they make predictions about future experiences, and like the latter they allow efficient prediction of total discounted rewards. However, successor-style representations are not optimized to generalize across policies: typically, we maintain a limited-length list of policies, and share information among them by representation learning or GPI. Successor-style representations also typically make no provision for gathering information or reasoning about latent variables. To address these limitations, we bring together ideas from predictive state representations, belief space value iteration, successor features, and convex analysis: we develop a new, general successor-style representation, together with a Bellman equation that connects multiple sources of information within this representation, including different latent states, policies, and reward functions. The new representation is highly expressive: for example, it lets us efficiently read off an optimal policy for a new reward function, or a policy that imitates a new demonstration. For this paper, we focus on exact computation of the new representation in small, known environments, since even this restricted setting offers plenty of interesting questions. Our implementation does not scale to large, unknown environments — nor would we expect it to, since it generalizes POMDP value iteration, which is difficult to scale. However, we believe that future work will allow us to extend our ideas to approximate reasoning in large, unknown environments. We conduct experiments to explore which of the potential barriers to scaling are most pressing.

Introduction

We describe a new representation for decision-theoretic planning, reinforcement learning, and imitation learning: the *successor feature set*. This representation generalizes a number of previous ideas in the literature, including successor features and POMDP/PSR value functions. Comparing to these previous representations: successor features assume a fixed policy or list of policies, while our goal is to reason efficiently about many policies at once; value functions assume a fixed reward function, while our goal is to reason efficiently about many reward functions at once.

Roughly, the successor feature set tells us how features of our future observations and actions depend on our current state and our choice of policy. More specifically, the successor feature set is a convex set of matrices; each matrix corresponds to a policy π, and describes how the features we will observe in the future depend on the current state under π.

The successor feature set provides a number of useful capabilities. These include reading off the optimal value function or policy for a new reward function, predicting the range of outcomes that we can achieve starting from a given state, and reading off a policy that imitates a desired state-action visitation distribution.

We describe a convergent dynamic programming algorithm for computing the successor feature set, generalizing the value iteration algorithm for POMDPs or PSRs. We also give algorithms for reading off the above-mentioned optimal policies and feature-matching policies from the successor feature set. Since the exact dynamic programming algorithm can be prohibitively expensive, we also experiment with randomized numerical approximations.

In this paper we focus on model-based reasoning about successor feature sets — that is, we assume access to an accurate world model. We also focus on algorithms that are exact in the limit of increasing computation. Successor-style representations are of course also extremely useful for approximate reasoning about large, unknown environments, and we believe that many of the ideas discussed here can inform that case as well, but we leave that direction for future work.

To summarize, our contributions are: a new successor-style representation that allows information to flow among different states, policies, and reward functions; algorithms for working with this new representation in small, known environments, including a convergent dynamic programming algorithm and ways to read off optimal policies and feature-matching policies; and computational experiments that evaluate the strengths and limitations of our new representation and algorithms.

Background and Notation

Our environment is a controlled dynamical system. We interact with it in a sequence of time steps; at each step, all relevant information is encoded in a state vector. Given this state vector, we choose an action. Based on the action and the current state, the environment changes to a new state, emits an observation, and moves to the next time step. We can describe such a system using one of a few related models: a Markov decision process (MDP), a partially-observable Markov decision process (POMDP), or a (transformed) predictive state representation (PSR). We describe these models below, and summarize our notation in Table 1.

MDPs

An MDP is the simplest model: there are k possible discrete states, numbered $1 \dots k$. The environment starts in one of these states, s_1. For each possible action $a \in \{1 \dots A\}$, the *transition matrix* $T_a \in \mathbb{R}^{k \times k}$ tells us how our state changes if we execute action a: $[T_a]_{ij}$ is the probability that the next state is $s_{t+1} = i$ if the current state is $s_t = j$.

More compactly, we can associate each state $1, 2, \dots, k$ with a corresponding standard basis vector $e_1, e_2, \dots, e_k$, and write q_t for the vector at time t. (So, if $s_t = i$ then $q_t = e_i$.) Then, $T_a q_t$ is the probability distribution over next states:

$$P(s_{t+1} \mid q_t, \mathrm{do}\ a) = \mathbb{E}(q_{t+1} \mid q_t, \mathrm{do}\ a) = T_a q_t$$

Here we have written $\mathrm{do}\ a$ to indicate that choosing an action is an *intervention*.

POMDPs

In an MDP, we get to know the exact state at each time step: q_t is always a standard basis vector. By contrast, in a POMDP, we only receive partial information about the underlying state: at each time step, after choosing our action a_t, we see an observation $o_t \in \{1 \dots O\}$ according to a distribution that depends on the next state s_{t+1}. The *observation matrix* $D \in \mathbb{R}^{O \times k}$ tells us the probabilities: D_{ij} is the probability of receiving observation $o_t = i$ if the next state is $s_{t+1} = j$.

To represent this partial information about state, we can let the state vector q_t range over the probability simplex instead of just the standard basis vectors: $[q_t]_i$ tells us the probability that the state is $s_t = i$, given all actions and observations so far, up to and including a_{t-1} and o_{t-1}. The vector q_t is called our *belief state*; we start in belief state q_1.

Just as in an MDP, we have $\mathbb{E}(q_{t+1} \mid q_t, \mathrm{do}\ a) = T_a q_t$. But now, instead of immediately resolving q_{t+1} to one of the corners of the simplex, we can only take into account partial state information: if $o_t = o$ then by Bayes rule

$$\begin{aligned} [q_{t+1}]_i &= P(s_{t+1} = i \mid q_t, \mathrm{do}\ a, o) \\ &= \frac{P(o \mid s_{t+1} = i) P(s_{t+1} = i \mid q_t, \mathrm{do}\ a)}{P(o \mid q_t, \mathrm{do}\ a)} \\ &= D_{oi}[T_a q_t]_i \,/\, \textstyle\sum_{o'} D_{o'i}[T_a q_t]_i \end{aligned}$$

More compactly, if $u \in \mathbb{R}^k$ is the vector of all 1s, and

$$T_{ao} = \mathrm{diag}(D_{o,\cdot}) T_a$$

Symbol	Type	Meaning
d	$\mathbb{N}$	dimension of feature vector
k	$\mathbb{N}$	dimension of state vector
A, O	$\mathbb{N}$	number of actions, observations
$f(q, a)$	$\mathbb{R}^d$	one-step feature function
F_a	$\mathbb{R}^{d \times k}$	implements f: $f(q, a) = F_a q$
T_{ao}	$\mathbb{R}^{k \times k}$	transition operator for action, observation
$\phi^\pi, \phi^\pi(q)$	$\mathbb{R}^d$	successor features for π (in state q)
A^π	$\mathbb{R}^{d \times k}$	implements ϕ^π: $\phi^\pi(q) = A^\pi q$
Φ	$\{\mathbb{R}^{d \times k}\}$	successor set
Φ_a, Φ_{ao}	$\{\mathbb{R}^{d \times k}\}$	backups of Φ for actions and observations

Table 1: Notation quick reference

where $\mathrm{diag}(\cdot)$ constructs a diagonal matrix from a vector, then our next belief state is

$$q_{t+1} = T_{ao} q_t \,/\, u^T T_{ao} q_t$$

A POMDP is strictly more general than an MDP: if our observation o_t tells us complete information about our next state s_{t+1}, then our belief state q_{t+1} will be a standard basis vector. This happens precisely when $[P(o_t = i \mid s_{t+1} = j) = 1] \Leftrightarrow [i = j]$.

PSRs

A PSR further generalizes a POMDP: we can think of a PSR as dropping the interpretation of q_t as a belief state, and keeping only the mathematical form of the state update. That is, we no longer require our model parameters to have any interpretation in terms of probabilities of partially observable states; we only require them to produce valid observation probability estimates. (It is possible to interpret PSR states and parameters in terms of experiments called *tests*; for completeness we describe this interpretation in the supplementary material, available online.)

In more detail, we are given a starting state vector q_1, matrices $T_{ao} \in \mathbb{R}^{k \times k}$, and a normalization vector $u \in \mathbb{R}^k$. We define our state vector by the recursion

$$q_{t+1} = T_{a_t o_t} q_t / u^T T_{a_t o_t} q_t$$

and our observation probabilities as

$$P(o_t = o \mid q_t, \mathrm{do}\ a) = u^T T_{ao} q_t$$

The only requirement on the parameters is that the observation probabilities $u^T T_{ao} q_t$ should always be nonnegative and sum to 1: under any sequence of actions and observations, if q_t is the resulting sequence of states,

$$(\forall a, o, t)\, u^T T_{ao} q_t \geq 0 \quad (\forall a, t)\, \textstyle\sum_o u^T T_{ao} q_t = 1$$

It is clear that a PSR generalizes a POMDP, and therefore also an MDP: we can always take u to be the vector of all 1s, and set T_{ao} according to the POMDP transition and observation probabilities, so that

$$[T_{ao}]_{ij} = P(o_t = o, s_{t+1} = i \mid s_t = j, a_t = a)$$

It turns out that PSRs are a *strict* generalization of POMDPs: there exist PSRs whose dynamical systems cannot be described by any finite POMDP. An example is the so-called *probability clock* (Jaeger 2000).

Policy Trees

We will need to work with policies for MDPs, POMDPs, and PSRs, handling different horizons as well as partial observability. For this reason, we will use a general policy representation: we will view a policy as a mixture of trees, with each tree representing a deterministic, nonstationary policy. A policy tree's nodes are labeled with actions, and its edges are labeled with observations (Fig. 1). To execute a policy tree π, we execute π's root action; then, based on the resulting observation o, we follow the edge labeled o from the root, leading to a subtree that we will call $\pi(o)$. To execute a mixture, we randomize over its elements. If desired we can randomize lazily, committing to each decision just before it affects our actions. We will work with finite, balanced trees, with depth equal to a horizon H; we can reason about infinite-horizon policies by taking a limit as $H \to \infty$.

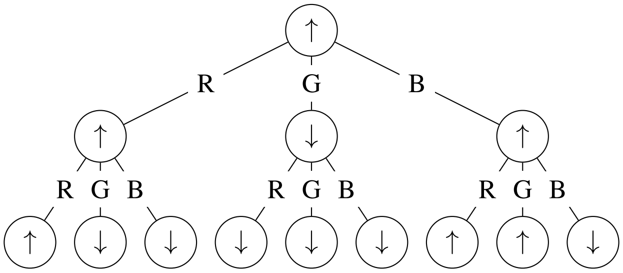


Figure 1: An example of a policy tree with actions ↑, ↓ and observations R, G, B.

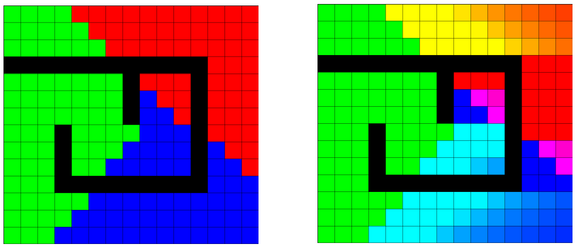
Imitation by Feature Matching

Successor feature sets have many uses, but we will start by motivating them with the goal of imitation. Often we are given demonstrations of some desired behavior in a dynamical system, and we would like to imitate that behavior. There are lots of ways to specify this problem, but one reasonable one is *apprenticeship learning* (Abbeel and Ng 2004) or *feature matching*. In this method, we define features of states and actions, and ask our learner to match some statistics of the observed features of our demonstrations.

In more detail, given an MDP, define a vector of features of the current state and action, $f(s, a) \in \mathbb{R}^d$; we call this the *one-step* or *immediate* feature vector. We can calculate the observed discounted features of a demonstration: if we visit states and actions $s_1, a_1, s_2, a_2, s_3, a_3, \ldots$, then the empirical discounted feature vector is

$$f(s_1, a_1) + \gamma f(s_2, a_2) + \gamma^2 f(s_3, a_3) + \ldots$$

where $\gamma \in [0, 1)$ is our *discount factor*. We can average the feature vectors for all of our demonstrations to get a *demonstration* or *target* feature vector ϕ^d.



(a) f of maze MDP.

(b) $\phi^{\text{go-left}}$ with $\gamma = 0.75$.

Figure 2: Maze environment example.

Analogously, for a policy π, we can define the *expected* discounted feature vector:

$$\phi^\pi = \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} f(s_t, a_t) \right]$$

We can use a finite horizon H by replacing $\sum_{t=1}^{\infty}$ with $\sum_{t=1}^{H}$ in the definitions of ϕ^d and ϕ^π; in this case we have the option of setting $\gamma = 1$.

Given a target feature vector in any of these models, we can ask our learner to design a policy that matches the target feature vector in expectation. That is, we ask the learner to find a policy π with

$$\phi^\pi = \phi^d$$

For example, suppose our world is a simple maze MDP like Fig. 2a. Suppose that our one-step feature vector $f(s, a) \in [0, 1]^3$ is the RGB color of the current state in this figure, and that our discount is $\gamma = 0.75$. If our demonstrations spend most of their time toward the left-hand side of the state space, then our target vector will be something like $\phi^d = [0.5, 3, 0.5]^T$: the green feature will have the highest expected discounted value. On the other hand, if our demonstrations spend most of their time toward the bottom-right corner, we might see something like $\phi^d = [2, 1, 1]^T$, with the blue feature highest.

Successor Features

To reason about feature matching, it will be important to predict how the features we see in the future depend on our current state. To this end, we define an analog of ϕ^π where we vary our start state, called the *successor feature representation* (Dayan 1993; Barreto et al. 2017):

$$\phi^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} f(s_t, a_t) \,\middle|\, \text{do } s_1 = s \right]$$

This function associates a vector of expected discounted features to each possible start state. We can think of $\phi^\pi(\cdot)$ as a generalization of a value function: instead of predicting total discounted rewards, it predicts total discounted feature vectors. In fact, the generalization is strict: $\phi^\pi(\cdot)$ contains enough information to compute the value function for any one-step reward function of the form $r^T f(s, a)$, via $V^\pi(s) = r^T \phi^\pi(s)$.

For example, in Fig. 2b, our policy is to always move left. The corresponding successor feature function looks similar to the immediate feature function, except that colors will be smeared rightward. The smearing will stop at walls, since an agent attempting to move through a wall will stop.

Extension to POMDPs and PSRs

We can generalize the above definitions to models with partial observability as well. This is not a typical use of successor features: reasoning about partial observability requires a model, while successor-style representations are often used in model-free RL. However, as Lehnert and Littman (2019) point out, the state of a PSR is already a prediction about the future, so incorporating successor features into these models makes sense.

In a POMDP, we have a belief state $q \in \mathbb{R}^k$ instead of a fully-observed state. We define the immediate features of q to be the expected features of the latent state:

$$f(q,a) = \sum_{s=1}^{k} q(s) f(s,a)$$

In a PSR, we similarly allow any feature function that is linear in the predictive state vector $q \in \mathbb{R}^k$:

$$f(q,a) = F_a q$$

with one matrix $F_a \in \mathbb{R}^{d \times k}$ for each action a. In either case, define the successor features to be

$$\phi^\pi(q) = \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} f(q_t, a_t) \,\middle|\, \text{do } q_1 = q \right]$$

Interestingly, the function ϕ^π is *linear* in q. That is, for each π, there exists a matrix $A^\pi \in \mathbb{R}^{d \times k}$ such that $\phi^\pi(q) = A^\pi q$. We call A^π the *successor feature matrix* for π; it is related to the parameters of the *Linear Successor Feature Model* of Lehnert and Littman (2019).

We can compute A^π recursively by working backward in time (upward from the leaves of a policy tree): for a tree with root action a, the recursion is

$$A^\pi = F_a + \gamma \sum_o A^{\pi(o)} T_{ao}$$

This recursion works by splitting A^π into contributions from the first step (F_a) and from steps $2 \ldots H$ (rest of RHS). We give a more detailed derivation, as well as a proof of linearity, in the supplementary material online. All the above works for MDPs as well by taking $q_t = e_{s_t}$, which lets us keep a uniform notation across MDPs, POMDPs, and PSRs.

It is worth noting the multiple feature representations that contribute to the function $\phi^\pi(q)$. First are the immediate features $f(q,a)$. Second is the PSR state, which can often be thought of as a feature representation for an underlying "uncompressed" model (Hefny, Downey, and Gordon 2015). Finally, both of the above feature representations help define the *exact* value of ϕ^π; we can also *approximate* ϕ^π using a third feature representation. Any of these feature representations could be related, or we could use separate features for all three purposes. We believe that an exploration of the roles of these different representations would be important and interesting, but we leave it for future work.

Successor Feature Sets

To reason about multiple policies, we can collect together multiple matrices: the *successor feature set* at horizon H is defined as the set of all possible successor feature matrices at horizon H,

$$\Phi^{(H)} = \{A^\pi \mid \pi \text{ a policy with horizon } H\}$$

As we will detail below, we can also define an infinite-horizon successor feature set Φ, which is the limit of $\Phi^{(H)}$ as $H \to \infty$.

The successor feature set tells us *how the future depends* on our state and our choice of policy. It tells us the range of outcomes that are possible: for a state q, each point in Φq tells us about one policy, and gives us moments of the distribution of future states under that policy. The extreme points of Φq therefore tell us the limits of what we can achieve. (Here we use the shorthand of *broadcasting*: set arguments mean that we perform an operation all possible ways, substituting one element from each set. E.g., if X, Y are sets, $X + Y$ means Minkowski sum $\{x + y \mid x \in X, y \in Y\}$.)

Note that $\Phi^{(H)}$ is a convex, compact set: by linearity of expectation, the feature matrix for a stochastic policy will be a convex combination of the matrices for its component deterministic policies. Therefore, $\Phi^{(H)}$ will be the convex hull of a finite set of matrices, one for each possible deterministic policy at horizon H.

Working with multiple policies at once provides a number of benefits: perhaps most importantly, it lets us define a Bellman backup that builds new policies combinatorially by combining existing policies at each iteration (Sec.). That way, we can reason about all possible policies instead of just a fixed list. Another benefit of Φ is that, as we will see below, it can help us compute optimal policies and feature-matching policies efficiently. On the other hand, because it contains so much information, the set Φ is a complicated object; it can easily become impractical to work with. We return to this problem in Sec. 12.

Special Cases

In some useful special cases, successor feature matrices and successor feature sets have a simpler structure that can make them easier to reason about and work with. E.g., in an MDP, we can split the successor feature matrix into its columns, resulting in one vector per state — this is the ordinary successor feature vector $\phi^\pi(s) = A^\pi e_s$. Similarly, we can split Φ into sets of successor feature vectors, one at each state, representing the range of achievable futures:

$$\phi(s) = \{\phi^\pi(s) \mid \pi \text{ a policy}\} = \Phi e_s$$

Fig. 3 visualizes these projections, along with the Bellman backups described below. Each projection tells us the discounted total feature vectors that are achievable from the corresponding state. For example, the top-left plot shows a set with five corners, each corresponding to a policy that is optimal in this state under a different reward function; the bottom-left corner corresponds to "always go down," which is optimal under reward $R(s,a) = (-1,-1) f(s,a)$.

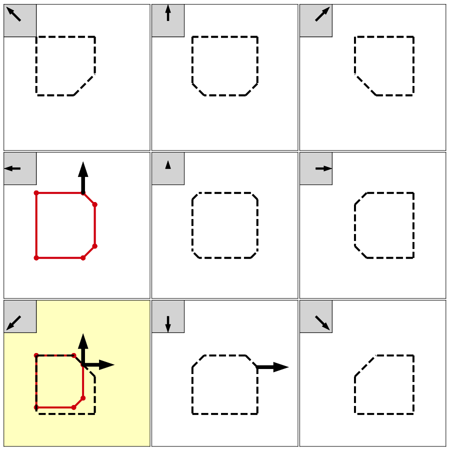


Figure 3: Visualization of the successor feature set Φ for a 3×3 gridworld MDP with 2d features. Start state is in yellow. Gray insets show one-step feature vectors, which depend only on the state, not the action. Each subplot shows one projection Φe_j (scale is arbitrary, so no axes are necessary). The red sets illustrate a Bellman backup at the bottom-left state, and the black arrows illustrate the feature-matching policy there. See text for details.

On the other hand, if we only have a single one-step feature ($f(q, a) \in \mathbb{R}$), then we can only represent a 1d family of reward functions. All positive multiples of f are equivalent to one another, as are all negative multiples. In this case, our recursion effectively reduces to classic POMDP or PSR value iteration: each element of Φ is now a vector $\alpha^\pi \in \mathbb{R}^k$ instead of a matrix $A^\pi \in \mathbb{R}^{d\times k}$. This α-vector represents the (linear) value function of policy π; the pointwise maximum of all these functions is the (piecewise linear and convex) optimal value function of the POMDP or PSR.

Bellman Equations

Each element of the successor feature set is a successor feature matrix for some policy, and as such, it satisfies the recursion given above. For efficiency, though, we would like to avoid running Bellman backups separately for too many possible policies. To this end, we can write a backup operator and Bellman equations that apply to all policies at once, and hence describe the entire successor feature set.

The joint backup works by relating horizon-H policies to horizon-$(H-1)$ policies. Every horizon-H policy tree can be constructed recursively, by choosing an action to perform at the root node and a horizon-$(H-1)$ tree to execute after each possible observation. So, we can break down any horizon-H policy (including stochastic ones) into a distribution over the initial action, followed by conditional distributions over horizon-$(H-1)$ policy trees for each possible initial observation.

Therefore, if we have the successor feature set $\Phi^{(H-1)}$ at horizon $H-1$, we can construct the successor feature set at horizon H in two steps: first, for each possible initial action a, we construct

$$\Phi_a^{(H)} = F_a + \gamma \sum_o \Phi^{(H-1)} T_{ao}$$

This set tells us the successor feature matrices for all horizon-H policies that begin with action a. Note that only the first action is deterministic: $\Phi^{(H-1)}$ lets us assign any conditional distribution over horizon-$(H-1)$ policy trees after each possible observation.

Second, since a general horizon-H policy is a distribution over horizon-H policies that start with different actions, each element of $\Phi^{(H)}$ is a convex combination of elements of $\Phi_a^{(H)}$ for different values of a. That is,

$$\Phi^{(H)} = \operatorname{conv} \bigcup_a \Phi_a^{(H)}$$

The recursion bottoms out at horizon 0, where we have

$$\Phi^{(0)} = \{0\}$$

since the discounted sum of a length-0 trajectory is always the zero vector.

Fig. 3 shows a simple example of the Bellman backup. Since this is an MDP, Φ is determined by its projections Φe_j onto the individual states. The action "up" takes us from the bottom-left state to the middle-left state. So, we construct $\Phi_{\text{up}} e_{\text{bottom-left}}$ by shifting and scaling $\Phi e_{\text{middle-left}}$ (red sets). The full set $\Phi e_{\text{bottom-left}}$ is the convex hull of four sets $\Phi_a e_{\text{bottom-left}}$; the other three are not shown, but for example, taking $a = $ right gives us a shifted and scaled copy of the set from the bottom-center plot.

The update from $\Phi^{(H-1)}$ to $\Phi^{(H)}$ is a contraction: see the supplementary material online for a proof. So, as $H \to \infty$, $\Phi^{(H)}$ will approach a limit Φ; this set represents the achievable successor feature matrices in the infinite-horizon discounted setting. Φ is a fixed point of the Bellman backup, and therefore satisfies the *stationary* Bellman equations

$$\Phi = \operatorname{conv} \bigcup_a \left[F_a + \gamma \sum_o \Phi T_{ao} \right]$$

Feature Matching and Optimal Planning

Once we have computed the successor feature set, we can return to the feature matching task described in Section . Knowing Φ makes feature matching easier: for any target vector of discounted feature expectations ϕ^d, we can efficiently either compute a policy that matches ϕ^d or verify that matching ϕ^d is impossible. We detail an algorithm for doing so in Alg. 1; more detail is in the supplementary material.

Fig. 3 shows the first steps of our feature-matching policy in a simple MDP. At the bottom-left state, the two arrows show the initial target feature vector (root of the arrows) and the computed policy (randomize between "up" and "right" according to the size of the arrows). The target feature vector at the next step depends on the outcome of randomization: each destination state shows the corresponding target and the second step of the computed policy.

We can also use the successor feature set to make optimal planning easier. In particular, if we are given a new reward

Algorithm 1: Feature Matching Policy

```
$t \leftarrow 1$
Initialize $\phi^d_t$ to the target vector of expected
  discounted features.
Initialize $q_t$ to the initial state of the environment.
repeat
    Choose actions $a_{it}$, vectors $\phi_{it} \in \Phi_{a_{it}} q_t$, and
      convex combination weights $p_{it}$ s.t.
      $\phi^d_t = \sum_{i=1}^{\ell} p_{it} \phi_{it}$.
    Choose an index $i$ according to probabilities $p_{it}$,
      and execute the corresponding action: $a_t \leftarrow a_{it}$.
    Write the corresponding $\phi_{it}$ as
      $\phi_{it} = F_{a_t} q_t + \gamma \sum_o \phi_{ot}$ by choosing
      $\phi_{ot} \in \Phi T_{a_t o} q_t$ for each $o$.
    Receive observation $o_t$, and calculate
      $p_t = P(o_t \mid q_t, a_t) = u^T T_{a_t o_t} q_t$.
    $q_{t+1} \leftarrow T_{a_t o_t} q_t / p_t$
    $\phi^d_{t+1} \leftarrow \phi_{o_t t} / p_t$
    $t \leftarrow t + 1$
until done
```

function expressed in terms of our features, say $R(q, a) = r^T f(q, a)$ for some coefficient vector r, then we can efficiently compute the optimal value function under R:

$$V^*(q) = \max_{\pi} r^T \phi^{\pi} q = \max \{ r^T \psi q \mid \psi \in \Phi \}$$

As a by-product we get an optimal policy: there will always be a matrix ψ that achieves the max above and satisfies $\psi \in \Phi_a$ for some a. Any such a is an optimal action.

Implementation

An exact representation of Φ can grow faster than exponentially with the horizon. So, in our experiments below, we work with a straightforward approximate representation. We use two tools: first, we store $\Phi_{ao} = \Phi T_{ao}$ for all a, o instead of storing Φ, since the former sets tend to be effectively lower-dimensional due to sparsity. Second, analogous to PBVI (Pineau, Gordon, and Thrun 2003a; Shani, Pineau, and Kaplow 2013), we fix a set of directions $m_i \in \mathbb{R}^{d \times k}$, and retain only the most extreme point of Φ_{ao} in each direction. Our approximate backed-up set is then the convex hull of these retained points. Just as in PBVI, we can efficiently compute backups by passing the max through the Minkowski sum in the Bellman equation. That is, for each i and each a, o, we solve

$$\arg\max \langle m_i, \phi \rangle \text{ for } \phi \in \bigcup_{a'} \left[F_{a'} + \gamma \sum_{o'} \Phi_{a'o'} \right] T_{ao}$$

by solving, for each i, a, o, a', o'

$$\arg\max \langle m_i, \phi \rangle \text{ for } \phi \in \Phi_{a'o'} T_{ao}$$

and combining the solutions.

There are a couple of useful variants of this implementation that we can use in *stoppable* problems (i.e., problems where we have an emergency-stop or a safety policy; see the supplemental material for more detail). First, we can update *monotonically*, i.e., keep the better of the horizon-H or horizon-$(H+1)$ successor feature matrices in each direction. Second, we can update *incrementally*: we can update any subset of our directions while leaving the others fixed.

More on Special Cases

With the above pruning strategy, our dynamic programming iteration generalizes PBVI (Pineau, Gordon, and Thrun 2003b). PBVI was defined originally for POMDPs, but it extends readily to PSRs as well: we just sample predictive states instead of belief states. To relate PBVI to our method, we look at a single task, with reward coefficient vector r. We sample a set of belief states or predictive states q_i; these are the directions that PBVI will use to decide which value functions (α-vectors) to retain. Based on these, we set the successor feature matrix directions to be $m_i = r q_i^T$ for all i.

Now, when we search within our backed up set $\Phi^{(H)}$ for the maximal element in direction m_i, we get some successor feature matrix ϕ. Because $\mathrm{tr}(\phi^T r q_i^T)$ is maximal, we know that $\mathrm{tr}(q_i^T \phi^T r) = q_i^T (\phi^T r)$ is also maximal: that is, $\phi^T r$ is as far as possible in the direction q_i. But $\phi^T r$ is a backed-up value function under the reward r; so, $\phi^T r$ is exactly the value function that PBVI would retain when maximizing in the direction q_i.

Experiments: Dynamic Programming

We tried our dynamic programming method on several small domains: the classic mountain-car domain and a random 18×18 gridworld with full and partial observability. We evaluated both planning and feature matching; results for the former are discussed in this section, and an example of the latter is in Fig. 3. We give further details of our experimental setup in the supplementary material online. At a high level, our experiments show that the algorithms behave as expected, and that they are practical for small domains. They also tell us about limits on scaling: the tightest of these limits is our ability to represent Φ accurately, governed by the number of boundary points that we retain for each Φ_{ao}.

In mountain-car, the agent has two actions: accelerate left and accelerate right. The state is (position, velocity), in $[-1.2, 0.6] \times [-0.07, 0.07]$. We discretize to a 12×12 mesh with piecewise-constant approximation. Our one-step features are radial basis functions of the state, with values in $[0, 1]$. We use 9 RBF centers evenly spaced on a 3×3 grid. In the MDP gridworld, the agent has four deterministic actions: up, down, left, and right. The one-step features are (x, y) coordinates scaled to $[-1, 1]$, similar to Fig. 3. In the POMDP gridworld, the actions are stochastic, and the agent only sees a noisy indicator of state. In all domains, the discount is $\gamma = 0.9$.

Fig. 4 shows how Bellman error evolves across iterations of dynamic programming. Since Φ is a set, we evaluate error by looking at random projections: how far do Φ and the backup of Φ extend in a given direction? We evaluate directions m_i that we optimized for during backups, as well as new random directions.

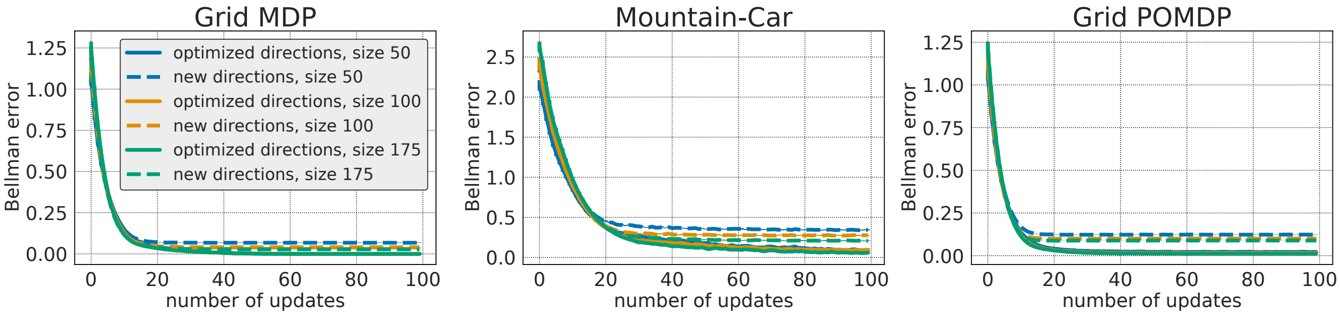


Figure 4: Bellman error v. iteration for three simple test domains, varying the amount of computation per iteration. We show error separately in directions we have optimized over and in new random directions. Average of 25 random seeds of the direction m_i with the highest bellman error per seed; all error bars are smaller than the line widths. The center panel shows the effect on Bellman error when we have higher-dimensional feature vectors. The rightmost panel shows the effect on Bellman error when the agent has less information about the exact state. In both cases the convergence rate stays similar, but we need more directions m_i to adequately sample the boundary of Φ (i.e., to lower the asymptotic error on new directions).

Note that the asymptotes for the new-directions lines are above zero; this persistent error is due to our limited-size representation of Φ. The error decreases as we increase the number of boundary points that we store. It is larger in the domains with more features and more uncertainty (center and right panels), due to the higher-dimensional A^π matrices and the need to sample mixed (uncertain) belief states.

Related Work

Successor features, a version of which were first introduced by Dayan (1993), provide a middle ground between model-free and model-based RL (Russek et al. 2017). They have been proposed as neurally plausible explanations of learning (Gershman et al. 2012; Gershman 2018; Momennejad et al. 2017; Stachenfeld, Botvinick, and Gershman 2017; Gardner, Schoenbaum, and Gershman 2018; Vértes and Sahani 2019).

Recently, numerous extensions have been proposed. Most similar to the current work are methods that generalize to a set of policies or tasks. Barreto et al. (2017) achieve transfer learning by generalizing across tasks with successor features; Barreto et al. (2018) use generalized policy improvement (GPI) over a set of policies. A few methods (Borsa et al. 2018; Ma, Wen, and Bengio 2018) recently combined universal value function approximators (Schaul et al. 2015) with GPI to perform multi-task learning, generalizing to a set of goals by conditioning on a goal representation. Barreto et al. (2020) extend policy improvement and policy evaluation from single tasks and policies to a list of them, but do not attempt to back up across policies.

Many authors have trained nonlinear models such as neural networks to predict successor-style representations, e.g., Kulkarni et al. (2016); Zhu et al. (2017); Zhang et al. (2017); Machado et al. (2017); Hansen et al. (2019). These works are complementary to our goal here, which is to design and analyze new, more general successor-style representations. We hope our generalizations eventually inform training methods for large-scale nonlinear models.

At the intersection of successor features and imitation learning, Zhu et al. (2017) address visual semantic planning; Lee, Srinivasan, and Doshi-Velez (2019) address off-policy model-free RL in a batch setting; and Hsu (2019) addresses active imitation learning.

As mentioned above, the individual elements of Φ are related to the work of Lehnert and Littman (2019). And, we rely on point-based methods (Pineau, Gordon, and Thrun 2003a; Shani, Pineau, and Kaplow 2013) to compute Φ.

Conclusion

This work introduces *successor feature sets*, a new representation that generalizes successor features. Successor feature sets represent and reason about successor feature predictions for all policies at once, and respect the compositional structure of policies, in contrast to other approaches that treat each policy separately. The set represents the boundaries of what is achievable in the future, and how these boundaries depend on our initial state. This information lets us read off optimal policies or imitate a demonstrated behavior.

We give algorithms for working with successor feature sets, including a dynamic programming algorithm to compute them, as well as algorithms to read off policies from them. The dynamic programming update is a contraction mapping, and therefore convergent. We give both exact and approximate versions of the update. The exact version can be intractable, due to the so-called “curse of dimensionality” and “curse of history.” The approximate version mitigates these curses using point-based sampling.

Finally, we present computational experiments. These are limited to relatively small, known environments; but in these environments, we demonstrate that we can compute successor feature sets accurately, and that they aid generalization. We also explore how our approximations scale with environment complexity.

Overall we believe that our new representation can provide insight on how to reason about policies in a dynamical system. We know, though, that we have only scratched the surface of possible strategies for working with this representation, and we hope that our analysis can inform future work on larger-scale environments.

References

Abbeel, P.; and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, 1.

Barreto, A.; Borsa, D.; Quan, J.; Schaul, T.; Silver, D.; Hessel, M.; Mankowitz, D.; Zidek, A.; and Munos, R. 2018. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *International Conference on Machine Learning*, 501–510. PMLR.

Barreto, A.; Dabney, W.; Munos, R.; Hunt, J. J.; Schaul, T.; van Hasselt, H. P.; and Silver, D. 2017. Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, 4055–4065.

Barreto, A.; Hou, S.; Borsa, D.; Silver, D.; and Precup, D. 2020. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences* .

Borsa, D.; Barreto, A.; Quan, J.; Mankowitz, D.; Munos, R.; van Hasselt, H.; Silver, D.; and Schaul, T. 2018. Universal successor features approximators. *arXiv preprint arXiv:1812.07626* .

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

Gardner, M. P.; Schoenbaum, G.; and Gershman, S. J. 2018. Rethinking dopamine as generalized prediction error. *Proceedings of the Royal Society B* 285(1891): 20181645.

Gershman, S. J. 2018. The successor representation: its computational logic and neural substrates. *Journal of Neuroscience* 38(33): 7193–7200.

Gershman, S. J.; Moore, C. D.; Todd, M. T.; Norman, K. A.; and Sederberg, P. B. 2012. The successor representation and temporal context. *Neural Computation* 24(6): 1553–1568.

Hansen, S.; Dabney, W.; Barreto, A.; Van de Wiele, T.; Warde-Farley, D.; and Mnih, V. 2019. Fast task inference with variational intrinsic successor features. *arXiv preprint arXiv:1906.05030* .

Hefny, A.; Downey, C.; and Gordon, G. 2015. Supervised Learning for Dynamical System Learning. In *Advances in neural information processing systems*.

Hsu, D. 2019. A New Framework for Query Efficient Active Imitation Learning. *arXiv preprint arXiv:1912.13037* .

Jaeger, H. 2000. Observable operator models for discrete stochastic time series. *Neural Computation* 12(6): 1371–1398.

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

Lee, D.; Srinivasan, S.; and Doshi-Velez, F. 2019. Truly batch apprenticeship learning with deep successor features. *arXiv preprint arXiv:1903.10077* .

Lehnert, L.; and Littman, M. L. 2019. Successor Features Combine Elements of Model-Free and Model-based Reinforcement Learning. Technical Report arXiv:1901.11437.

Ma, C.; Wen, J.; and Bengio, Y. 2018. Universal successor representations for transfer reinforcement learning. *arXiv preprint arXiv:1804.03758* .

Machado, M. C.; Rosenbaum, C.; Guo, X.; Liu, M.; Tesauro, G.; and Campbell, M. 2017. Eigenoption discovery through the deep successor representation. *arXiv preprint arXiv:1710.11089* .

Momennejad, I.; Russek, E. M.; Cheong, J. H.; Botvinick, M. M.; Daw, N. D.; and Gershman, S. J. 2017. The successor representation in human reinforcement learning. *Nature Human Behaviour* 1(9): 680–692.

Pineau, J.; Gordon, G.; and Thrun, S. 2003a. Point-based Value Iteration: An Anytime Algorithm for POMDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*.

Pineau, J.; Gordon, G. J.; and Thrun, S. B. 2003b. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, 1025–1030.

Russek, E. M.; Momennejad, I.; Botvinick, M. M.; Gershman, S. J.; and Daw, N. D. 2017. Predictive representations can link model-based reinforcement learning to model-free mechanisms. *PLoS computational biology* 13(9): e1005768.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal value function approximators. In *International conference on machine learning*, 1312–1320.

Shani, G.; Pineau, J.; and Kaplow, R. 2013. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems* 27: 1–51.

Stachenfeld, K. L.; Botvinick, M. M.; and Gershman, S. J. 2017. The hippocampus as a predictive map. *Nature neuroscience* 20(11): 1643.

Vértes, E.; and Sahani, M. 2019. A neurally plausible model learns successor representations in partially observable environments. In *Advances in Neural Information Processing Systems*, 13714–13724.

Zhang, J.; Springenberg, J. T.; Boedecker, J.; and Burgard, W. 2017. Deep reinforcement learning with successor features for navigation across similar environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2371–2378. IEEE.

Zhu, Y.; Gordon, D.; Kolve, E.; Fox, D.; Fei-Fei, L.; Gupta, A.; Mottaghi, R.; and Farhadi, A. 2017. Visual semantic planning using deep successor representations. In *Proceedings of the IEEE international conference on computer vision*, 483–492.