COLLOQUIUM PAPER

Fast reinforcement learning with generalized policy updates

André Barreto[a,1], Shaobo Hou[a], Diana Borsa[a], David Silver[a], and Doina Precup[a,b]

[a]DeepMind, London EC4A 3TW, United Kingdom; and [b]School of Computer Science, McGill University, Montreal, QC H3A 0E9, Canada

Edited by David L. Donoho, Stanford University, Stanford, CA, and approved July 9, 2020 (received for review July 20, 2019)

The combination of reinforcement learning with deep learning is a promising approach to tackle important sequential decision-making problems that are currently intractable. One obstacle to overcome is the amount of data needed by learning systems of this type. In this article, we propose to address this issue through a divide-and-conquer approach. We argue that complex decision problems can be naturally decomposed into multiple tasks that unfold in sequence or in parallel. By associating each task with a reward function, this problem decomposition can be seamlessly accommodated within the standard reinforcement-learning formalism. The specific way we do so is through a generalization of two fundamental operations in reinforcement learning: policy improvement and policy evaluation. The generalized version of these operations allow one to leverage the solution of some tasks to speed up the solution of others. If the reward function of a task can be well approximated as a linear combination of the reward functions of tasks previously solved, we can reduce a reinforcement-learning problem to a simpler linear regression. When this is not the case, the agent can still exploit the task solutions by using them to interact with and learn about the environment. Both strategies considerably reduce the amount of data needed to solve a reinforcement-learning problem.

artificial intelligence | reinforcement learning | generalized policy improvement | generalized policy evaluation | successor features

COMPUTER SCIENCES

Reinforcement learning (RL) provides a conceptual framework to address a fundamental problem in artificial intelligence: the development of situated agents that learn how to behave while interacting with the environment (1). In RL, this problem is formulated as an agent-centric optimization in which the goal is to select actions to get as much reward as possible in the long run. Many problems of interest are covered by such an inclusive definition; not surprisingly, RL has been used to model robots (2) and animals (3), to simulate real (4) and artificial (5) limbs, to play board (6) and card (7) games, and to drive simulated bicycles (8) and radio-controlled helicopters (9), to cite just a few applications.

Recently, the combination of RL with deep learning added several impressive achievements to the above list (10–13). It does not seem too far-fetched, nowadays, to expect that these techniques will be part of the solution of important open problems. One obstacle to overcome on the track to make this possibility a reality is the enormous amount of data needed for an RL agent to learn to perform a task. To give an idea, in order to learn how to play one game of Atari, a simple video game console from the 1980s, an agent typically consumes an amount of data corresponding to several weeks of uninterrupted playing; it has been shown that, in some cases, humans are able to reach the same performance level in around 15 min (14).

One reason for such a discrepancy in the use of data is that, unlike humans, RL agents usually learn to perform a task essentially from scratch. This suggests that the range of problems our agents can tackle can be significantly extended if they are endowed with the appropriate mechanisms to leverage prior knowledge. In this article we describe one such mechanism. The framework we discuss is based on the premise that an RL problem can usually be decomposed into a multitude of "tasks." The intuition is that, in complex problems, such tasks will naturally emerge as recurrent patterns in the agent–environment interaction. Tasks then lead to a useful form of prior knowledge: once the agent has solved a few, it should be able to reuse their solution to solve other tasks faster. Grasping an object, moving in a certain direction, and seeking some sensory stimulus are examples of naturally occurring behavioral building blocks that can be learned and reused.

Ideally, information should be exchanged across tasks whenever useful, preferably through a mechanism that integrates seamlessly into the standard RL formalism. The way this is achieved here is to have each task be defined by a reward function. As most animal trainers would probably attest, rewards are a natural mechanism to define tasks because they can induce very distinct behaviors under otherwise identical conditions (15, 16). They also provide a unifying language that allows each task to be cast as an RL problem, blurring the (somewhat arbitrary) distinction between the two (17). Under this view, a conventional RL problem is conceptually indistinguishable from a task self-imposed by the agent by "imagining" recompenses associated with arbitrary events. The problem then becomes a stream of tasks, unfolding in sequence or in parallel, whose solutions inform each other in some way.

What allows the solution of one task to speed up the solution of other tasks is a generalization of two fundamental operations underlying much of RL: policy evaluation and policy improvement. The generalized version of these procedures, jointly referred to as "generalized policy updates," extend their standard counterparts from point-based to set-based operations. This makes it possible to reuse the solution of tasks in two distinct ways. When the reward function of a task can be reasonably approximated as a linear combination of other tasks' reward functions, the RL problem can be reduced to a simpler linear regression that is solvable with only a fraction of the data. When the linearity constraint is not satisfied, the agent can still leverage the solution of tasks—in this case, by using them to interact with and learn about the environment. This can also considerably

This paper results from the Arthur M. Sackler Colloquium of the National Academy of Sciences, "The Science of Deep Learning," held March 13–14, 2019, at the National Academy of Sciences in Washington, DC. NAS colloquia began in 1991 and have been published in PNAS since 1995. From February 2001 through May 2019 colloquia were supported by a generous gift from The Dame Jillian and Dr. Arthur M. Sackler Foundation for the Arts, Sciences, & Humanities, in memory of Dame Sackler's husband, Arthur M. Sackler. The complete program and video recordings of most presentations are available on the NAS website at http://www.nasonline.org/science-of-deep-learning.

Author contributions: A.B., D.B., D.S., and D.P. designed research; A.B., S.H., and D.B. performed research; A.B., S.H., and D.B. analyzed data; and A.B. wrote the paper.

The authors declare no competing interest.

This article is a PNAS Direct Submission.

[1]To whom correspondence may be addressed. Email: andrebarreto@google.com.

This article contains supporting information online at https://www.pnas.org/lookup/suppl/doi:10.1073/pnas.1907370117/-/DCSupplemental.

First published August 17, 2020.

reduce the amount of data needed to solve the problem. Together, these two strategies give rise to a divide-and-conquer approach to RL that can potentially help scale our agents to problems that are currently intractable.

RL

We consider the RL framework outlined in the Introduction: an agent interacts with an environment by selecting actions to get as much reward as possible in the long run (1). This interaction happens at discrete time steps, and, as usual, we assume it can be modeled as a Markov decision process (MDP) (18).

An MDP is a tuple $M \equiv (\mathcal{S}, \mathcal{A}, p, r, \gamma)$ whose components are defined as follows. The sets $\mathcal{S}$ and $\mathcal{A}$ are the state space and action space, respectively (we will consider that $\mathcal{A}$ is finite to simplify the exposition, but most of the ideas extend to infinite action spaces). At every time step t, the agent finds itself in a state $s \in \mathcal{S}$ and selects an action $a \in \mathcal{A}$. The agent then transitions to a next state s', where a new action is selected, and so on. The transitions between states can be stochastic: the dynamics of the MDP, $p(\cdot|s, a)$, give the next-state distribution upon taking action a in state s. In RL, we assume that the agent does not know p, and thus it must learn based on transitions sampled from the environment.

A sample transition is a tuple (s, a, r', s') where r' is the reward given by the function $r(s, a, s')$, also unknown to the agent. As discussed, here we adopt the view that different reward functions give rise to distinct tasks. Given a task r, the agent's goal is to find a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$, that is, a mapping from states to actions, that maximizes the value of every state–action pair, defined as

$$Q_r^\pi(s, a) \equiv \mathbb{E}^\pi \left[\sum_{i=0}^{\infty} \gamma^i r(S_{t+i}, A_{t+i}, S_{t+i+1}) \mid S_t = s, A_t = a \right], \quad [1]$$

where S_t and A_t are random variables indicating the state occupied and the action selected by the agent at time step t, $\mathbb{E}^\pi[\cdot]$ denotes expectation over the trajectories induced by π, and $\gamma \in [0, 1)$ is the discount factor, which gives less weight to rewards received further into the future. The function $Q_r^\pi(s, a)$ is usually referred to as the "action-value function" of policy π on task r; sometimes, it will be convenient to also talk about the "state-value function" of π, defined as $V_r^\pi(s) \equiv Q_r^\pi(s, \pi(s))$.

Given an MDP representing a task r, there exists at least one optimal policy π_r^* that attains the maximum possible value at all states; the associated optimal value function V_r^* is shared by all optimal policies (18). Solving a task r can thus be seen as the search for an optimal policy π_r^* or an approximation thereof. Since the number of possible policies grows exponentially with the size of $\mathcal{S}$ and $\mathcal{A}$, a direct search in the space of policies is usually infeasible. One way to circumvent this difficulty is to resort to methods based on dynamic programming, which exploit the properties of MDPs to reduce the cost of searching for a policy (19).

Policy Updates. RL algorithms based on dynamic programming build on two fundamental operations (1).

Definition 1. "Policy evaluation" is the computation of Q_r^π, the value function of policy π on task r.

Definition 2. Given a policy π and a task r, "policy improvement" is the definition of a policy π' such that

$$Q_r^{\pi'}(s, a) \geq Q_r^\pi(s, a) \textit{ for all } (s, a) \in \mathcal{S} \times \mathcal{A}. \quad [2]$$

We call one application of policy evaluation followed by one application of policy improvement a "policy update." Given an arbitrary initial policy π, successive policy updates give rise to a sequence of improving policies that will eventually reach an optimal policy π_r^* (18). Even when policy evaluation and policy improvement are not performed exactly, it is possible to derive guarantees on the performance of the resulting policy based on the approximation errors introduced in these steps (20, 21). Fig. 1 illustrates the basic intuition behind policy updates.

What makes policy evaluation tractable is a recursive relation between state-action values known as the Bellman equation:

$$Q_r^\pi(s, a) = \mathbb{E}_{S' \sim p(\cdot|s,a)} \left[r(s, a, S') + \gamma Q_r^\pi(S', \pi(S')) \right]. \quad [3]$$

Expression (Exp.) **3** induces a system of linear equations whose solution is Q_r^π. This immediately suggests ways of performing policy evaluation when the MDP is known (18). Importantly, the Bellman equation also facilitates the computation of Q_r^π without knowledge of the dynamics of the MDP. In this case, one estimates the expectation on the right-hand side of Exp. **3** based on samples from $p(\cdot|s, a)$, leading to the well-known method of temporal differences (22, 23). It is also often the case that in problems of interest the state space $\mathcal{S}$ is too big to allow for a tabular representation of the value function, and hence Q_r^π is replaced by an approximation $\tilde{Q}_r^\pi$.

As for policy improvement, it is in fact simple to define a policy π' that performs at least as well as, and generally better than, a given policy π. Once the value function of π on task r is known, one can compute an improved policy π' as

$$\pi'(s) \in \operatorname*{arg\,max}_{a \in \mathcal{A}} Q_r^\pi(s, a). \quad [4]$$

In words, the action selected by policy π' on state s is the one that maximizes the action-value function of policy π on that state. The fact that policy π' satisfies Definition 2 is one of the fundamental results in dynamic programming and the driving force behind many algorithms used in practice (18).

The specific way policy updates are carried out gives rise to different dynamic programming algorithms. For example, value iteration and policy iteration can be seen as the extremes of a spectrum of algorithms defined by the extent of the policy evaluation step (19, 24). RL algorithms based on dynamic programming can be understood as stochastic approximations of these methods or other instantiations of policy updates (25).

Generalized Policy Updates

From the discussion above, one can see that an important branch of the field of RL depends fundamentally on the notions of policy evaluation and policy improvement. We now discuss generalizations of these operations.

Definition 3. "Generalized policy evaluation" (GPE) is the computation of the value function of a policy π on a set of tasks $\mathcal{R}$.

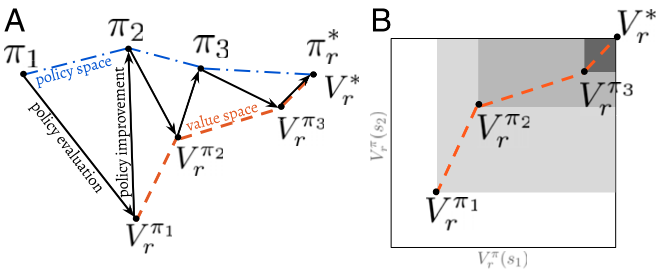

Fig. 1. (*A*) Sequence of policy updates as a trajectory that alternates between the policy and value spaces and eventually converges to an optimal solution (1). (*B*) Detailed view of the trajectory across the value space for a state space with two states only. The shadowed rectangles associated with each value function represent the region of the value space containing the value function that will result from one application of policy improvement followed by policy evaluation (cf. Exp. **2**).

Definition 4. Given a set of policies Π and a task r, "generalized policy improvement" (GPI) is the definition of a policy π' such that

$$Q_r^{\pi'}(s,a) \geq \sup_{\pi\in\Pi} Q_r^{\pi}(s,a) \textit{ for all } (s,a)\in\mathcal{S}\times\mathcal{A}. \qquad \textbf{[5]}$$

GPE and GPI are strict generalizations of their standard counterparts, which are recovered when $\mathcal{R}$ has a single task and Π has a single policy. However, it is when $\mathcal{R}$ and Π are not singletons that GPE and GPI reach their full potential. In this case, they become a mechanism to quickly construct a solution for a task, as we now explain. Suppose we are interested in one of the tasks $r\in\mathcal{R}$, and we have a set of policies Π available. The origin of these policies is not important: they may have come up as solutions for specific tasks or have been defined in any other arbitrary way. If the policies $\pi\in\Pi$ are submitted to GPE, we have their value functions on the task $r\in\mathcal{R}$. We can then apply GPI over these value functions to obtain a policy π' that represents an improvement over all policies in Π. Clearly, this reasoning applies without modification to any task in $\mathcal{R}$. Therefore, by applying GPE and GPI to a set of policies Π and a set of tasks $\mathcal{R}$, one can compute a policy for any task in $\mathcal{R}$ that will in general outperform every policy in Π. Fig. 2 shows a graphical depiction of GPE and GPI.

Obviously, in order for GPE and GPI to be useful in practice, we must have efficient ways of performing these operations. Consider GPE, for example. If we were to individually evaluate the policies $\pi\in\Pi$ over the set of tasks $r\in\mathcal{R}$, it is unlikely that the scheme above would result in any gains in terms of computation or consumption of data. To see why this is so, suppose again that we are interested in a particular task r. Computing the value functions of policies $\pi\in\Pi$ on task r would require $|\Pi|$ policy evaluations with a naive form of GPE (here, $|\cdot|$ denotes the cardinality of a set). Although the resulting GPI policy π' would compare favorably to all policies in Π, this guarantee would be vacuous if these policies are not competent at task r. Therefore, a better allocation of resources might be to use the policy evaluations for standard policy updates, which would generate a sequence of $|\Pi|$ policies with increasing performance on task r (compare Figs. 1 and 2). This difficulty in using generalized policy updates in practice is further aggravated if we do not have a fast way to carry out GPI. Next, we discuss efficient instantiations of GPE and GPI.

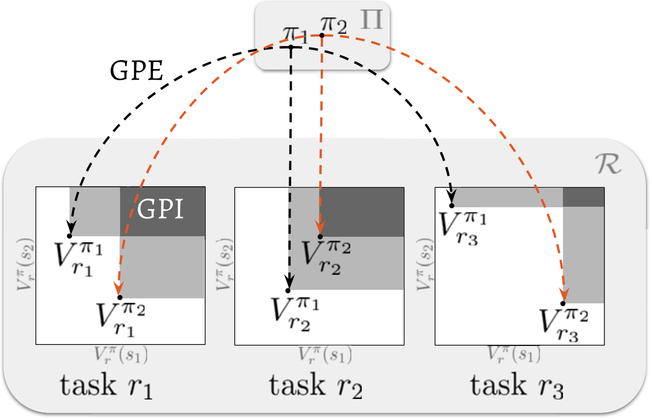

Fig. 2. Depiction of generalized policy updates on a state space with two states only. With GPE each policy $\pi\in\Pi$ is evaluated on all tasks $r\in\mathcal{R}$. The state-value function of policy π on task r, V_r^{π}, delimits a region in the value space where the next value function resulting from policy improvement will be (cf. Fig. 1). The analogous space induced by GPI corresponds to the intersection of the regions associated with the individual value functions (represented as dark gray rectangles in the figure). The smaller the space of value functions associated with GPI, the stronger the guarantees regarding the performance of the resulting policy.

Fast GPE with Successor Features. Conceptually, we can think of GPE as a function associated with a policy π that takes a task r as input and outputs a value function Q_r^{π} (26). Hence, a practical way of implementing GPE would be to define a suitable representation for tasks and then learn a mapping from r to value functions Q_r^{π} (27). This is feasible when such a mapping can be reasonably approximated by the choice of function approximator and enough examples (r, Q_r^{π}) are available to characterize the relation underlying these pairs. Here, we will focus on a form of GPE that is based on a similar premise but leads to a form of generalization over tasks that is correct by definition.

Let $\boldsymbol{\phi}:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\mapsto\mathbb{R}^d$ be an arbitrary function whose output we will see as "features." Then, for any $\mathbf{w}\in\mathbb{R}^d$, we have a task defined as

$$r_{\mathbf{w}}(s,a,s') = \boldsymbol{\phi}(s,a,s')^{\top}\mathbf{w}, \qquad \textbf{[6]}$$

where $^{\top}$ denotes the transpose of a vector. Let

$$\mathcal{R}_{\boldsymbol{\phi}} \equiv \{r_{\mathbf{w}} = \boldsymbol{\phi}^{\top}\mathbf{w} \,|\, \mathbf{w}\in\mathbb{R}^d\}$$

be the set of tasks induced by all possible instantiations of $\mathbf{w}\in\mathbb{R}^d$. We now show how to carry out an efficient form of GPE over $\mathcal{R}_{\boldsymbol{\phi}}$.

Following Barreto et al. (28), we define the "successor features" (SFs) of policy π as

$$\boldsymbol{\psi}^{\pi}(s,a) \equiv \mathbb{E}^{\pi}\left[\sum_{i=0}^{\infty}\gamma^i\boldsymbol{\phi}(S_{t+i},A_{t+i},S_{t+i+1}) \,|\, S_t=s, A_t=a\right].$$

The ith component of $\boldsymbol{\psi}^{\pi}(s,a)$ gives the expected discounted sum of ϕ_i when following policy π starting from (s,a). Thus, $\boldsymbol{\psi}^{\pi}$ can be seen as a d-dimensional value function in which the features $\phi_i(s,a,s')$ play the role of reward functions (cf. Exp. **1**). As a consequence, SFs satisfy a Bellman equation analogous to Exp. **3**, which means that they can be computed using standard RL methods like temporal differences (22).

Given the SFs of a policy π, $\boldsymbol{\psi}^{\pi}$, we can quickly evaluate π on task $r_{\mathbf{w}}\in\mathcal{R}_{\boldsymbol{\phi}}$ by computing

$$\begin{aligned}\boldsymbol{\psi}^{\pi}(s,a)^{\top}\mathbf{w} &= \mathbb{E}^{\pi}\left[\sum_{i=0}^{\infty}\gamma^i\boldsymbol{\phi}(S_{t+i},A_{t+i},S_{t+i+1})^{\top}\mathbf{w}|S_t=s,A_t=a\right]\\ &= \mathbb{E}^{\pi}\left[\sum_{i=0}^{\infty}\gamma^i r_{\mathbf{w}}(S_{t+i},A_{t+i},S_{t+i+1}) \,|\, S_t=s, A_t=a\right]\\ &= Q_{r_{\mathbf{w}}}^{\pi}(s,a)\equiv Q_{\mathbf{w}}^{\pi}(s,a). \qquad \textbf{[7]}\end{aligned}$$

That is, the computation of the value function of policy π on task $r_{\mathbf{w}}$ is reduced to the inner product $\boldsymbol{\psi}^{\pi}(s,a)^{\top}\mathbf{w}$. Since this is true for any task $r_{\mathbf{w}}$, SFs provide a mechanism to implement a very efficient form of GPE over the set $\mathcal{R}_{\boldsymbol{\phi}}$ (cf. Definition 3).

The question then arises as to how inclusive the set $\mathcal{R}_{\boldsymbol{\phi}}$ is. Since $\mathcal{R}_{\boldsymbol{\phi}}$ is fully determined by $\boldsymbol{\phi}$, the answer to this question lies in the definition of these features. Mathematically speaking, $\mathcal{R}_{\boldsymbol{\phi}}$ is the linear space spanned by the d features ϕ_i. This view suggests ways of defining $\boldsymbol{\phi}$ that result in a $\mathcal{R}_{\phi}$ containing all possible tasks. A simple example can be given for when both the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are finite. In this case, we can recover any possible reward function by making $d=|S|^2\times|A|$ and having each ϕ_i be an indicator function associated with the occurrence of a specific transition (s,a,s'). This is essentially Dayan's (29) concept of "successor representation" extended from $\mathcal{S}$ to $\mathcal{S}\times\mathcal{A}\times\mathcal{S}$. The notion of covering the space of tasks can be generalized to continuous $\mathcal{S}$ and $\mathcal{A}$ if we think of ϕ_i as basis functions over the function space $\mathcal{S}\times\mathcal{A}\times\mathcal{S}\mapsto\mathbb{R}$, for we know that, under some technical conditions, one can approximate any function in this space with arbitrary accuracy by having an appropriate basis.

Although these limiting cases are reassuring, more generally we want to have a number of features $d \ll |S|$. This is possible if we consider that we are only interested in a subset of all possible tasks—which should in fact be a reasonable assumption in most realistic scenarios. In this case, the features $\boldsymbol{\phi}$ should capture the underlying structure of the set of tasks $\mathcal{R}$ in which we are interested. For example, many tasks we care about could be recovered by features $\boldsymbol{\phi}_i$ representing entities (from concrete objects to abstract concepts) or salient events (such as the interaction between entities). In *Fast RL with GPE and GPI*, we give concrete examples of what the features $\boldsymbol{\phi}$ may look like and also discuss possible ways of computing them directly from data. For now, it is worth mentioning that, even when $\boldsymbol{\phi}$ does not span the space $\mathcal{R}$, one can bound the error between the two sides of Exp. **7** based on how well the tasks of interest can be approximated using $\boldsymbol{\phi}$ (proposition 1 in ref. 30).

Fast GPI with Value-Based Action Selection. Thanks to the concept of value function, standard policy improvement can be carried out efficiently through Exp. **4**. This raises the question whether the same is possible with GPI. We now answer this question in the affirmative for the case in which GPI is applied over a finite set Π.

Following Barreto et al. (28), we propose to compute an improved policy π' as

$$\pi'(s) \in \underset{a\in\mathcal{A}}{\operatorname{argmax}} \max_{\pi\in\Pi} Q_r^{\pi}(s,a). \qquad \textbf{[8]}$$

Barreto et al. have shown that Exp. **8** qualifies as a legitimate form of GPI as per Definition 4 (theorem 1 in ref. 28). Note that the action selected by a GPI policy π' on a state s may not coincide with any of the actions selected by the policies $\pi\in\Pi$ on that state. This highlights an important difference between Exp. **8** and methods that define a higher-level policy that alternates between the policies $\pi\in\Pi$ (32). In *SI Appendix*, we show that the policy π' resulting from Exp. **8** will in general outperform its analog defined over Π.

As discussed, GPI reduces to standard policy improvement when Π contains a single policy. Interestingly, this also happens with Exp. **8** when one of the policies in Π strictly outperforms the others on task r. This means that, after one application of GPI, adding the resulting policy π' to the set Π will reduce the next application of Exp. **8** to Exp. **4**. Therefore, if GPI is used in a sequence of policy updates, it will collapse back to its standard counterpart after the first update. In contrast with policy improvement, which is inherently iterative, GPI is an one-off operation that yields a quick solution for a task when multiple policies have been evaluated on it.

The guarantees regarding the performance of the GPI policy π' can be extended to the scenario where Exp. **8** is used with approximate value functions. In this case, the lower bound in Exp. **5** decreases in proportion to the maximum approximation error in the value function approximations (theorem 1 in ref. 28). We refer the reader to *SI Appendix* for an extended discussion on GPI.

The Generalized Policy. Strictly speaking, in order to leverage the instantiations of GPE and GPI discussed in this article, we only need two elements: features $\boldsymbol{\phi}:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\mapsto\mathbb{R}^d$ and a set of policies $\Pi=\{\pi_i\}_{i=1}^n$. Together, these components yield a set of SFs $\Psi=\{\boldsymbol{\psi}^{\pi_i}\}_{i=1}^n$, which provide a quick form of GPE over the set of tasks $\mathcal{R}_\phi$ induced by the features $\boldsymbol{\phi}$. Specifically, for any $\mathbf{w}\in\mathbb{R}^d$, we can quickly evaluate policy π_i on task $r_\mathbf{w}(s,a,s')=\boldsymbol{\phi}(s,a,s')^\top\mathbf{w}$ by computing $Q_\mathbf{w}^{\pi_i}(s,a)=\boldsymbol{\psi}^{\pi_i}(s,a)^\top\mathbf{w}$. Once we have the value functions of policies π_i on task $r_\mathbf{w}$, $\{Q_\mathbf{w}^{\pi_i}\}_{i=1}^n$, Exp. **8** can be used to obtain a GPI policy π' that will in general outperform the policies π_i on $r_\mathbf{w}$. Observe that the same scheme can be used if we have approximate SFs $\tilde{\boldsymbol{\psi}}^{\pi_i}\approx\boldsymbol{\psi}^{\pi_i}$, which will be the case in most realistic scenarios. Since the resulting value functions will also be approximations, $\tilde{Q}_\mathbf{w}^{\pi_i}\approx Q_\mathbf{w}^{\pi_i}$, we are back to the approximate version of GPI discussed in *Fast GPI with Value-Based Action Selection*.

Given a set of SFs Ψ, approximate or otherwise, any $\mathbf{w}\in\mathbb{R}^d$ gives rise to a policy through GPE (Exp. **7**) and GPI (Exp. **8**). We can thus think of generalized updates as implementing a policy $\pi_\Psi(s;\mathbf{w})$ whose behavior is modulated by $\mathbf{w}$. We will call π_Ψ a "generalized policy." Fig. 3 provides a schematic view of the computation of π_Ψ through GPE and GPI.

Since each component of $\mathbf{w}$ weighs one of the features $\phi_i(s,a,s')$, changing them can intuitively be seen as setting the agent's current "preferences." For example, the vector $\mathbf{w}=[0,1,-2]^\top$ indicates that the agent is indifferent to feature ϕ_1 and wants to seek feature ϕ_2 while avoiding feature ϕ_3 with twice the impetus. Specific instantiations of π_Ψ can behave in ways that are very different from its constituent policies $\pi\in\Pi$. We can draw a parallel with nature if we think of features as concepts like water or food and note how much the desire for these items can affect an animal's behavior. Analogies aside, this sort

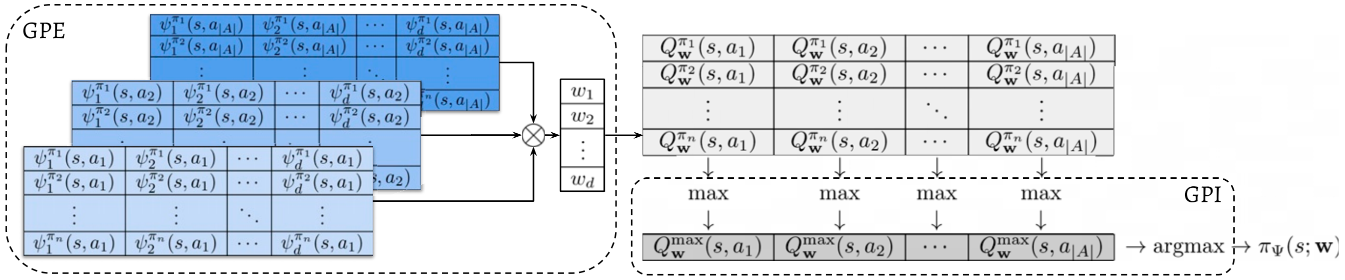
GPE
$\psi_1^{\pi_1}(s,a_{|A|})$ $\psi_2^{\pi_1}(s,a_{|A|})$ $\cdots$ $\psi_d^{\pi_1}(s,a_{|A|})$
$\psi_1^{\pi_2}(s,a_{|A|})$ $\psi_2^{\pi_2}(s,a_{|A|})$ $\cdots$ $\psi_d^{\pi_2}(s,a_{|A|})$
$\psi_1^{\pi_1}(s,a_2)$ $\psi_2^{\pi_1}(s,a_2)$ $\cdots$ $\psi_d^{\pi_1}(s,a_2)$
$\psi_1^{\pi_2}(s,a_2)$ $\psi_2^{\pi_2}(s,a_2)$ $\cdots$ $\psi_d^{\pi_2}(s,a_2)$
$\psi_1^{\pi_1}(s,a_1)$ $\psi_2^{\pi_1}(s,a_1)$ $\cdots$ $\psi_d^{\pi_1}(s,a_1)$
$\psi_1^{\pi_2}(s,a_1)$ $\psi_2^{\pi_2}(s,a_1)$ $\cdots$ $\psi_d^{\pi_2}(s,a_1)$
$\psi_1^{\pi_n}(s,a_1)$ $\psi_2^{\pi_n}(s,a_1)$ $\cdots$ $\psi_d^{\pi_n}(s,a_1)$
w_1 w_2 $\vdots$ w_d
$Q_\mathbf{w}^{\pi_1}(s,a_1)$ $Q_\mathbf{w}^{\pi_1}(s,a_2)$ $\cdots$ $Q_\mathbf{w}^{\pi_1}(s,a_{|A|})$
$Q_\mathbf{w}^{\pi_2}(s,a_1)$ $Q_\mathbf{w}^{\pi_2}(s,a_2)$ $\cdots$ $Q_\mathbf{w}^{\pi_2}(s,a_{|A|})$
$Q_\mathbf{w}^{\pi_n}(s,a_1)$ $Q_\mathbf{w}^{\pi_n}(s,a_2)$ $\cdots$ $Q_\mathbf{w}^{\pi_n}(s,a_{|A|})$
max max max max
GPI
$Q_\mathbf{w}^{\max}(s,a_1)$ $Q_\mathbf{w}^{\max}(s,a_2)$ $\cdots$ $Q_\mathbf{w}^{\max}(s,a_{|A|})$ → argmax ↦ $\pi_\Psi(s;\mathbf{w})$

Fig. 3. Schematic representation of how the generalized policy π_Ψ is implemented through GPE and GPI. Given a state $s\in\mathcal{S}$, the first step is to compute, for each $a\in\mathcal{A}$, the values of the n SFs: $\{\boldsymbol{\psi}^{\pi_i}(s,a)\}_{i=1}^n$. Note that $\boldsymbol{\psi}^{\pi_i}$ can be arbitrary nonlinear functions of their inputs, possibly represented by complex approximators like deep neural networks (30, 31). The SFs can be laid out as an $n\times|A|\times d$ tensor in which the entry (i,a,j) is $\psi_j^{\pi_i}(s,a)$. Given this layout, GPE reduces to the multiplication of the SF tensor with a vector $\mathbf{w}\in\mathbb{R}^d$, the result of which is an $n\times|A|$ matrix whose (i,a) entry is $Q^{\pi_i}(s,a)$. GPI then consists in finding the maximum element of this matrix and returning the associated action a. This whole process can be seen as the implementation of a policy $\pi_\Psi(s;\mathbf{w})$ whose behavior is parameterized by a vector of preferences $\mathbf{w}$.

of arithmetic over features provides a rich interface for the agent to interact with the environment at a higher level of abstraction in which decisions correspond to preferences encoded as a vector $\mathbf{w}$. Next, we discuss how this can be leveraged to speed up the solution of an RL task.

Fast RL with GPE and GPI

We now describe how to build and use the adaptable policy π_Ψ implemented by GPE and GPI. To make the discussion more concrete, we consider a simple RL environment depicted in Fig. 4. The environment consists of a 10×10 grid with four actions available: $\mathcal{A} = \{\texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}\}$. The agent occupies one of the grid cells, and there are also 10 objects spread across the grid. Each object belongs to one of two types. At each time step t, the agent receives an image showing its position and the position and type of each object. Based on this information, the agent selects an action $a \in \mathcal{A}$, which moves it one cell along the desired direction. The agent can pick up an object by moving to the cell occupied by it; in this case, it gets a reward defined by the type of the object. A new object then pops up in the grid, with both its type and location sampled uniformly at random (more details are in *SI Appendix*).

This simple environment can be seen as a prototypical member of the class of problems in which GPE and GPI could be useful. This becomes clear if we think of objects as instances of (potentially abstract) concepts, here symbolized by their types, and note that the navigation dynamics are a proxy for any sort of dynamics that mediate the interaction of the agent with the world. In addition, despite its small size, the number of possible configurations of the grid is actually quite large, of the order of 10^{15}. This precludes an exact representation of value functions and illustrates the need for approximations that inevitably arises in many realistic scenarios.

By changing the rewards associated with each object type, one can create different tasks. We will consider that the agent wants to build a set of SFs Ψ that give rise to a generalized policy $\pi_\Psi(s; \mathbf{w})$ that can adapt to different tasks through the vector of preferences $\mathbf{w}$. This can be either because the agent does not know in advance the task it will face or because it will face more than one task.

Defining a Basis for Behavior. In order to build the SFs Ψ, the agent must define two things: features $\boldsymbol{\phi}$ and a set of policies Π. Since $\boldsymbol{\phi}$ should be associated with rewarding events, we define each feature ϕ_i as an indicator function signaling whether an object of type i has been picked up by the agent (i.e., $\boldsymbol{\phi} \in \mathbb{R}^2$). To be precise, we have that $\phi_i(s, a, s') = 1$ if the transition from state s to state s' is associated with the agent picking up an object of type i, and $\phi_i(s, a, s') = 0$ otherwise. These features induce a set $\mathcal{R}_\phi$ where task $r_\mathbf{w} \in \mathcal{R}_\phi$ is characterized by how desirable or undesirable each type of object is.

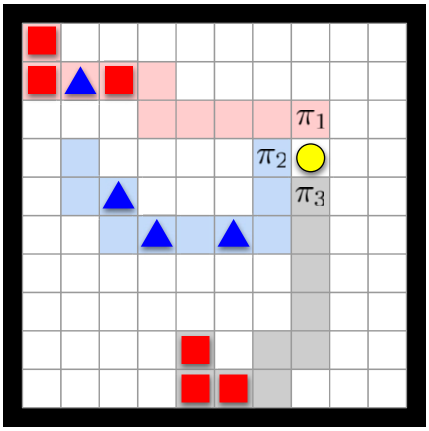

Fig. 4. Depiction of the environment used in the experiments. The shape of the objects (square or triangle) represents their type; the agent is depicted as a circle. We also show the first 10 steps taken by 3 policies, π_1, π_2, and π_3, that would perform optimally on tasks $\mathbf{w}_1 = [1, 0]$, $\mathbf{w}_2 = [0, 1]$, and $\mathbf{w}_3 = [1, -1]$ for any discount factor $\gamma \geq 0.5$.

Now that we have defined $\boldsymbol{\phi}$, we turn to the question of how to determine an appropriate set of policies Π. We will restrict the policies in Π to be solutions to tasks $r_\mathbf{w} \in \mathcal{R}_\phi$. We start with what is perhaps the simplest choice in this case: a set $\Pi_{12} = \{\pi_1, \pi_2\}$ whose two elements are solutions to the tasks $\mathbf{w}_1 = [1, 0]^\top$ and $\mathbf{w}_2 = [0, 1]^\top$ (henceforth, we will drop the transpose superscript to avoid clutter). Note that the goal in tasks $\mathbf{w}_1$ and $\mathbf{w}_2$ is to pick up objects of one type while ignoring objects of the other type.

We are now ready to compute the SFs Ψ induced by our choices of $\boldsymbol{\phi}$ and Π. In our experiments, we used an algorithm analogous to Q-learning to compute approximate SFs $\tilde{\boldsymbol{\psi}}^{\pi_1}$ and $\tilde{\boldsymbol{\psi}}^{\pi_2}$ (pseudocode in *SI Appendix*). We represented the SFs using multilayer perceptrons with two hidden layers (33).

The set of SFs $\tilde{\Psi}$ yields a generalized policy $\pi_{\tilde{\Psi}}(s; \mathbf{w})$ parameterized by $\mathbf{w}$. We now evaluate $\pi_{\tilde{\Psi}}$ on the task whose goal is to pick up objects of the first type while avoiding objects of the second type. Using $\boldsymbol{\phi}$ defined above, this task can be represented as $r_{\mathbf{w}_3}(s, a, s') = \boldsymbol{\phi}(s, a, s')^\top \mathbf{w}_3$, with $\mathbf{w}_3 = [1, -1]$. We thus evaluate the generalized policy instantiated as $\pi_{\tilde{\Psi}}(s; \mathbf{w}_3)$.

Results are shown in Fig. 5*A*. As a reference, we also show the learning curve of Q-learning (23) using the same architecture to directly approximate $Q^\pi_{\mathbf{w}_3}$. GPE and GPI allow one to compute an instantaneous solution for a new task, without any learning on the task itself, that is competitive with the policies found by Q-learning when using around 6×10^4 sample transitions. The performance of the policy $\pi_{\tilde{\Psi}}$ synthesized by GPE and GPI corresponds to more than 70% of the performance eventually achieved by Q-learning after processing 10^6 transitions. This is quite an impressive result when we note that $\pi_{\tilde{\Psi}}$ managed to avoid objects of the second type even though its constituents policies π_1 and π_2 were never trained to actively avoid objects.

We used a total of 10^6 sample transitions to learn both SFs $\tilde{\boldsymbol{\psi}}^{\pi_1}$ and $\tilde{\boldsymbol{\psi}}^{\pi_2}$, which is the same amount of data used by Q-learning to achieve its final performance. The advantage of doing the former is that, once we have the SFs, we can use GPE and GPI to instantaneously compute a solution for any task in $\mathcal{R}_\phi$. However, how well do GPE and GPI actually perform on $\mathcal{R}_\phi$? To answer this question, we ran a second round of experiments to assess the generalization of $\pi_{\tilde{\Psi}}$ over the entire set $\mathcal{R}_\phi$. Since this evaluation clearly depends on the set of policies used, we consider two other sets in addition to $\Pi_{12} = \{\pi_1, \pi_2\}$. The new sets are $\Pi_{34} = \{\pi_3, \pi_4\}$ and $\Pi_5 = \{\pi_5\}$, where the policies π_i are solutions to the tasks $\mathbf{w}_3 = [1, -1]$, $\mathbf{w}_4 = [-1, 1]$, and $\mathbf{w}_5 = [1, 1]$. We repeated the previous experiment with each policy set and evaluated the resulting policies $\pi_{\tilde{\Psi}}$ over 19 tasks $\mathbf{w}$ evenly spread over the nonnegative quadrants of the unit circle (tasks in the negative quadrant are uninteresting because all of the agent must do is to avoid hitting objects). Results are shown in Fig. 6*A*. As expected, the generalization ability of GPE and GPI depends on the set of policies used. Perhaps more surprising is how well the generalized policy $\pi_{\tilde{\Psi}}$ induced by some of these sets perform across the entire space of tasks $\mathcal{R}_\phi$, sometimes matching the best performance of Q-learning when solving each task individually.

These experiments show that a proper choice of base policies Π can lead to good generalization over the entire set of tasks $\mathcal{R}_\phi$. In general, though, it is unclear how to define an appropriate Π. Fortunately, we can refer to our theoretical understanding of GPE and GPI to have some guidance. First, we know from

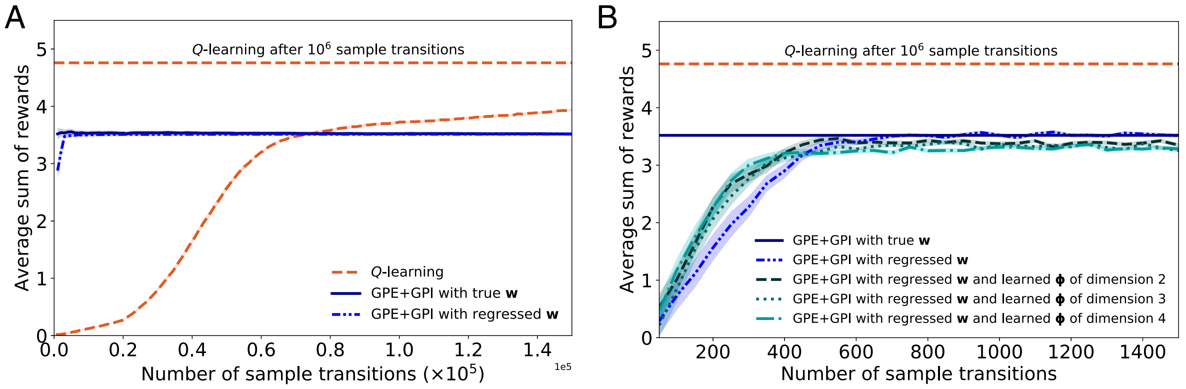

Fig. 5. Average sum of rewards on task $\mathbf{w}_3 = [1, -1]$. GPE and GPI used $\Pi_{12} = \{\pi_1, \pi_2\}$ as the base policies and the corresponding SFs consumed 5×10^5 sample transitions to be trained each. *B* is a zoomed-in version of *A* showing the early performance of GPE and GPI under different setups. The results reflect the best performance of each algorithm over multiple parameter configurations (*SI Appendix*). Shadowed regions are one standard error over 100 runs.

the discussion above that the larger the number of policies in Π the stronger the guarantees regarding the performance of the resulting policy $\pi_{\tilde{\Psi}}$ (Exp. **5**). In addition to that, Barreto et al. have shown that it is possible to guarantee a minimum performance level for $\pi_{\tilde{\Psi}}$ on task $\mathbf{w}$ based on $\min_i \|\mathbf{w} - \mathbf{w}_i\|$, where $\|\cdot\|$ is some norm and $\mathbf{w}_i$ are the tasks associated with the policies $\pi_i \in \Pi$ used for GPE and GPI (theorem 2 in ref. 28). Together, these two insights suggest that, as we increase the size of Π, the performance of the resulting policy $\pi_{\tilde{\Psi}}$ should improve across $\mathcal{R}_\phi$, especially on tasks that are close to the tasks $\mathbf{w}_i$. To test this hypothesis empirically, we repeated the previous experiment, but now, instead of comparing disjoint policy sets, we compared a sequence of sets formed by adding one by one the policies π_2, π_5, π_1, and π_3, in this order, to the initial set $\{\pi_4\}$. The results, in Fig. 6*B*, confirm the trend implied by the theory.

Task Inference. So far, we have assumed that the agent knows the vector $\mathbf{w}$ that describes the task of interest. Although this can be the case in some scenarios, ideally we would be able to apply GPE and GPI even when $\mathbf{w}$ is not provided. In this section and in *Preferences as Actions*, we describe two possible ways for the agent to learn $\mathbf{w}$.

Given a task r, we are looking for a $\mathbf{w} \in \mathbb{R}^d$ that leads to good performance of the generalized policy $\pi_\Psi(s; \mathbf{w})$. We could in principle approach this problem as an optimization over $\mathbf{w} \in \mathbb{R}^d$ whose objective is to maximize the value of $\pi_\Psi(s; \mathbf{w})$ across (a subset of) the state space. It turns out that we can exploit the structure underlying SFs to efficiently determine $\mathbf{w}$ without ever looking at the value of π_Ψ. Suppose we have a set of m sample transitions from a given task, $\{(s_i, a_i, r'_i, s'_i)\}_{i=1}^m$. Then, based on Exp. **6**, we can infer $\mathbf{w}$ by solving the following minimization:

$$\min_{\tilde{\mathbf{w}}} \sum_{i=1}^{m} |\boldsymbol{\phi}(s_i, a_i, s'_i)^\top \tilde{\mathbf{w}} - r'_i|^p, \qquad \textbf{[9]}$$

where $p \geq 1$ (one may also want to consider the inclusion of a regularization term, see ref. 33). Observe that, once we have a solution $\tilde{\mathbf{w}}$ for the problem above, we can plug it in Exp. **7** and use GPE and GPI as we did before—that is, we have just turned an RL task into an easier linear regression problem.

To illustrate the potential of this approach, we revisited the task $\mathbf{w}_3 = [1, -1]$ tackled above, but now, instead of assuming we knew $\mathbf{w}_3$, we solved the problem in Exp. **9** using $p = 2$. We collected sample transitions using a policy $\hat{\pi}$ that picks actions

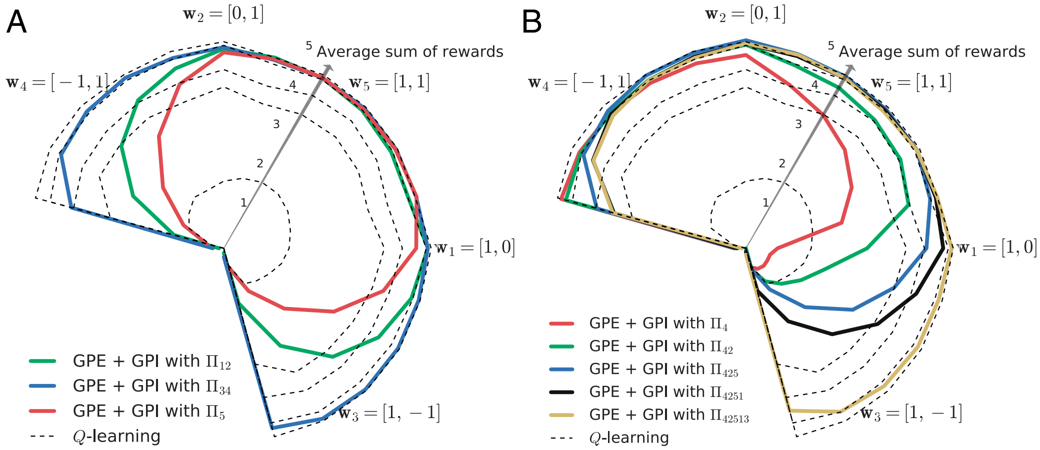

Fig. 6. Results on the space of tasks $\mathcal{R}_\phi$ induced by a two-dimensional ϕ. The sets of policies Π used in *A* are disjoint; in *B* these sets overlap. The evaluation of an algorithm on a task is represented as a vector whose direction indicates the task $\mathbf{w}$ and whose magnitude gives the average sum of rewards over 10 runs with 250 trials each. *Q*-learning learned each task individually, from scratch; the dotted curves correspond to its performance after having processed 5×10^4, 1×10^5, 2×10^5, 5×10^5, and 1×10^6 sample transitions. Our method only learned the policies in the sets Π and then generalized across all others tasks through GPE and GPI. The SFs used for GPE and GPI consumed 5×10^5 sample transitions to be trained each.

uniformly at random and used the gradient descent method to refine an approximation $\tilde{\mathbf{w}} \approx \mathbf{w}$ online (*SI Appendix*). Results are shown in Fig. 5 *A* and *B*. The performance of the generalized policy $\pi_{\tilde{\Psi}}(s; \mathbf{w}_3)$ using the correct $\mathbf{w}_3$ was recovered after around 800 sample transitions only. To put this number into perspective, recall that Q-learning needs around 60,000 transitions, or 75 times as much data, to achieve the same performance level. The number of transitions needed to learn $\tilde{\mathbf{w}}$ can be reduced further if we use the closed-form solution for the least-squares problem.

The problem described in Exp. **9** can be solved even if the rewards are the only information available about a given task. Other interesting possibilities arise when the observations provided by the environment can be used to infer $\tilde{\mathbf{w}}$, such as, for example, visual or auditory cues indicating the current task. In this case, it might be possible for the agent to determine $\tilde{\mathbf{w}}$ even before having observed a single nonzero reward (30).

So far, we have used handcrafted features $\boldsymbol{\phi}$. It turns out that Exp. **9** can be generalized to also allow $\boldsymbol{\phi}$ to be inferred from data. Given sample transitions from k tasks, $\{(s_{ij}, a_{ij}, r'_{ij}, s'_{ij})\}_{i=1}^{m_j}$, with $j = 1, 2, \ldots, k$, we can formulate the problem as the search for a function $\tilde{\boldsymbol{\phi}} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}^c$ and k vectors $\tilde{\mathbf{w}}_i \in \mathbb{R}^c$ satisfying

$$\min_{\tilde{\boldsymbol{\phi}}} \sum_{j=1}^{k} \min_{\tilde{\mathbf{w}}_j} \sum_{i=1}^{m_j} |\tilde{\boldsymbol{\phi}}(s_{ij}, a_{ij}, s'_{ij})^\top \tilde{\mathbf{w}}_j - r'_{ij}|^p, \qquad \textbf{[10]}$$

with $p \geq 1$. Note that the features $\tilde{\boldsymbol{\phi}}$ can be arbitrary nonlinear functions of their inputs. As discussed by Barreto et al. (30), if we make $c = k$ we can solve a simplified version of the problem above in which $\tilde{\mathbf{w}}_j$ do not show up. More generally, the problem in Exp. **10** can be solved as a multitask regression (34).

In Fig. 5*B*, we show the performance of $\pi_{\tilde{\Psi}}(s; \tilde{\mathbf{w}})$ when using learned features $\tilde{\boldsymbol{\phi}}$. These results were obtained as follows. First, we used the random policy $\hat{\pi}$ to collect data from tasks $\mathbf{w}_1$, $\mathbf{w}_2$, $\mathbf{w}_4$, and $\mathbf{w}_5$ and applied the stochastic gradient descent algorithm to solve Exp. **10** with $p = 2$. We then discarded the resulting approximations $\tilde{\mathbf{w}}_i$ appearing in Exp. **10** and solved Exp. **9** exactly as before but now using $\tilde{\boldsymbol{\phi}}$ instead of $\boldsymbol{\phi}$. As with SFs, in order to represent the features, we used multilayer perceptrons with two hidden layers (*SI Appendix*). The results in Fig. 5*B* also show the effect of varying the dimension of the learned features on the performance of $\pi_\Psi(s; \tilde{\mathbf{w}})$ (other illustrations are in refs. 28 and 30).

Preferences as Actions. In this section, we consider the scenario where, given features $\boldsymbol{\phi}$ and a set of policies Π, there is no vector $\mathbf{w}$ that leads to acceptable performance of $\pi_\Psi(s; \mathbf{w})$. To illustrate this situation, we use our environment with a slight twist: now, on picking up an object of a certain type, the agent gets a positive reward only if the number of objects of this type is greater or equal to the number of objects of the other type. Otherwise, the agent gets a negative reward.

One can represent this problem using the previously defined features $\boldsymbol{\phi}$ by switching between two instantiations of $\mathbf{w}$: $[1, -1]$ when the first type of object is more abundant and $[-1, 1]$ otherwise. This means that the appropriate value for $\mathbf{w}$ is now state-dependent. To handle this situation, we introduce a function mapping states to preferences, $\omega : \mathcal{S} \mapsto \mathcal{W}$, with $\mathcal{W} \subseteq \mathbb{R}^d$, and redefine the generalized policy as $\pi_\Psi(s; \omega(s))$. The RL problem then becomes to find a ω that correctly modulates π_Ψ (32, 35).

We can look at the computation of $\pi_\Psi(s; \omega(s))$ as a two-stage process: first, the agent computes a vector of preferences $\mathbf{w} = \omega(s)$; then, using GPE and GPI, it determines the action to be executed in state s as $a = \pi_\Psi(s; \mathbf{w})$. Under this view, $\omega(s)$ becomes an indirect way of selecting an action in s, which allows us to think of it as a policy defined in the space $\mathcal{W}$. We have thus replaced the original problem of finding a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ with the problem of finding a policy $\omega : \mathcal{S} \mapsto \mathcal{W}$.

To illustrate the benefits of applying this transformation to the problem, we will use the task described above in which the agent must pick up the type of object that is more abundant. We will tackle this problem by reusing the SFs $\tilde{\Psi}$ induced by the two policies in Π_{12}. Given $\tilde{\Psi}$, the problem comes down to finding a mapping $\omega : \mathcal{S} \mapsto \mathcal{W}$ that leads to good performance of $\pi_{\tilde{\Psi}}(s; \omega(s))$. We define $\mathcal{W} = \{-1, 0, 1\}^2$ and use Q-learning to learn ω.

We compare the performance of $\pi_{\tilde{\Psi}}(s; \omega(s))$ with two baselines: GPE and GPI using a fixed $\mathbf{w}$, $\pi_{\tilde{\Psi}}(s; \mathbf{w})$, and the standard version of Q-learning that learns a policy $\pi(s)$ over primitive actions. The results, shown in Fig. 7, illustrate how $\pi_{\tilde{\Psi}}(s; \omega(s))$ can significantly outperform the baselines. It is easy to understand why this is the case when we compare $\pi_{\tilde{\Psi}}(s; \omega(s))$ with $\pi_{\tilde{\Psi}}(s; \mathbf{w})$, for the experiment was designed to render the latter unsuitable. However, why is it that replacing the four-dimensional space $\mathcal{A}$ with the nine-dimensional space $\mathcal{W}$ leads to better performance?

The reason is that $\omega(s)$ can be easier to learn than $\pi(s)$. As discussed, the vectors $\mathbf{w}$ represent the agent's current preferences for types of objects. Since, in general, such preferences should only change when an object is picked up, after selecting a specific $\mathbf{w}$, the agent can hold on to it until it bumps into an object. That is, it is possible to define a good policy ω that maps multiple consecutive states s to the same $\mathbf{w}$. In contrast, it will normally not be possible to execute a single primitive action $a \in \mathcal{A}$ between the collection of two consecutive objects. This means that the same policy can have a simpler representation as a mapping $\mathcal{S} \mapsto \mathcal{W}$ than as a mapping $\mathcal{S} \mapsto \mathcal{A}$, making ω easier to learn with certain types of function approximators like neural networks (33). Note that other choices of $\boldsymbol{\phi}$ may induce other forms of regularities in ω—smoothness being an obvious example. Interestingly, the fact that the agent may be able to adhere to the same $\mathbf{w}$ for many time steps allows for a temporally extended policy $\pi_\Psi(s; \omega(s))$ that is only permitted to change preferences at certain points in time, reducing even further the amount of data needed to learn the task (31).

As a final remark, we note that the ability to chain a sequence of preference vectors $\mathbf{w}$ changes the nature of the problem of computing an appropriate set of features $\tilde{\boldsymbol{\phi}}$. When the agent has to adhere to a single $\mathbf{w}$, the features $\tilde{\phi}_i$ must span the space of rewards defining the tasks of interest. When the agent is allowed

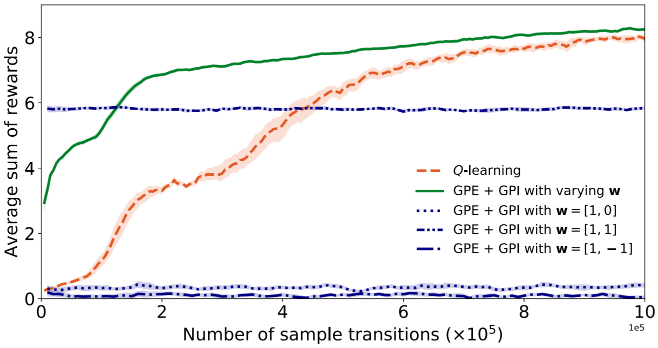

Fig. 7. Average sum of rewards on the task whose goal is to pick up objects of the type that is more abundant. GPE and GPI used Π_{12} as the base policies and the corresponding SFs consumed 5×10^5 transitions to be trained each. The results reflect the best performance of each algorithm over multiple parameter configurations (*SI Appendix*). Shadowed regions are one standard error over 100 runs.

to learn a policy ω over preferences $\mathbf{w}$, this linearity assumption is no longer necessary. Similarly, the use of a fixed $\mathbf{w}$ requires that the set of available SFs leads to a competent policy for all tasks $\mathbf{w}$ we may care about. Here, this requirement is alleviated, since, by changing $\mathbf{w}$, the agent may be able to compensate for eventual imperfections on the induced GPI policies (31).

Lifelong Learning. We have looked at the different subproblems involved in the use of GPE and GPI. For didactic purposes, we have discussed these problems in isolation, but it is instructive to consider how they can jointly emerge in a real application. One scenario where the problems above would naturally arise is "lifelong learning," in which the interaction of the agent with the environment unfolds over a long period (36). Since we expect some patterns to repeat during such an interaction, it is natural to break it in tasks that share some common structure. In terms of the concepts discussed in this article, this common structure would be $\boldsymbol{\phi}$, which can be handcrafted based on prior knowledge about the problem or learned through Exp. **10** at the beginning of the agent's lifetime. Based on the features $\tilde{\boldsymbol{\phi}}$, the agent can compute and store the SFs $\tilde{\boldsymbol{\psi}}^{\pi}$ associated with the solutions π of the tasks encountered along the way. When facing a new task r, the agent can use the vector of preferences $\mathbf{w}$ to quickly adapt the generalized policy $\pi_{\tilde{\Psi}}$. We have discussed two ways to do so. In the first one, the agent computes an approximation $\tilde{\mathbf{w}}$ through Exp. **9**, which immediately yields a solution $\pi_{\tilde{\Psi}}(s;\tilde{\mathbf{w}})$. The second possibility is to learn a mapping $\omega(s): \mathcal{S} \mapsto \mathcal{W}$ and use it to have a per-state modulation of the generalized policy, $\pi_{\tilde{\Psi}}(s;\omega(s))$. Regardless of how the agent computes $\mathbf{w}$, it can then choose to hold on to $\pi_{\tilde{\Psi}}$ or use it as a initial solution for the task to be refined by a standard RL algorithm. Either way, the agent can also compute the SFs of the resulting policy, which can in turn be added to the library of SFs $\tilde{\Psi}$, improving even further its ability to tackle future tasks (28, 30).

Conclusion

We have generalized two fundamental operations in RL, policy improvement and policy evaluation, from single to multiple operands (tasks and policies, respectively). The resulting operations, GPE and GPI, can be used to speed up the solution of an RL problem. We showed possible ways to efficiently implement GPE and GPI and discussed how their combination leads to a generalized policy $\pi_{\Psi}(s;\mathbf{w})$ whose behavior is modulated by a vector of preferences $\mathbf{w}$. Two ways of learning $\mathbf{w}$ were considered. In the simplest case, $\mathbf{w}$ is the solution of a linear regression problem. This reduces an RL task to a much simpler problem that can be solved using only a fraction of the data. This strategy depends on two conditions: 1) the reward function of the tasks of interest are approximately linear in the features $\boldsymbol{\phi}$ used for GPE and 2) the set of policies Π used for GPI lead to good performance of π_{Ψ} on the tasks of interest. When these assumptions do not hold, one can still resort to GPE and GPI by looking at the preferences $\mathbf{w}$ as actions. This strategy can also improve sample efficiency if the mapping from states to preferences is simpler to learn than the corresponding policy.

Many extensions of the basic framework presented in this article are possible. As mentioned, Barreto et al. (31) have proposed a way of implementing a generalized policy π_{Ψ} that explicitly leverages temporal abstraction by treating preferences $\mathbf{w}$ as abstract actions that persist for multiple time steps. Borsa et al. (37) introduced a generalized form of SFs that has a representation $\mathbf{z} \in \mathcal{Z}$ of a policy $\pi_{\mathbf{z}}$ as one of its inputs: $\boldsymbol{\psi}(s,a,\mathbf{z}) = \boldsymbol{\psi}^{\pi_{\mathbf{z}}}(s,a)$. In principle, this allows one to apply GPI over arbitrary, potentially infinite, sets Π (for example, by replacing the maximization in Exp. **8** with $\sup_{\mathbf{z}\in\mathcal{Z}} \boldsymbol{\psi}(s,a,\mathbf{z})^{\top}\mathbf{w}$). Hunt et al. (38) extended SFs to entropy-regularized RL, and, in doing so, they proposed solutions for many challenges that come up when applying GPE and GPI in continuous action spaces. More recently, Hansen et al. (39) proposed an approach that allows the features $\tilde{\boldsymbol{\phi}}$ to be learned from data in the absence of a reward signal. Other extensions are possible, including different instantiations of GPE and GPI.

All of these extensions expand the framework built upon GPE and GPI. Together, they provide a conceptual toolbox that allows one to decompose an RL problem into tasks whose solutions inform each other. This results in a divide-and-conquer approach to RL, which, combined with deep learning, has the potential to scale up our agents to problems currently out of reach.

Data Availability. The source code used to generate all of the data associated with this article has been deposited in GitHub (https://github.com/deepmind/deepmind-research/tree/master/option_keyboard/gpe_gpi_experiments).

ACKNOWLEDGMENTS. We thank Joseph Modayil, Pablo Sprechmann, and the anonymous reviewer for their invaluable comments regarding this article.

1. R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, 2018).
2. J. Kober, J. A. Bagnell, J. Peters, Reinforcement learning in robotics: A survey. *Int. J. Robot. Res.* **32**, 1238–1274 (2013).
3. W. Schultz, P. Dayan, P. R. Montague, A neural substrate of prediction and reward. *Science* **275**, 1593–1599 (1997).
4. E. Todorov, Optimality principles in sensorimotor control. *Nat. Neurosci.* **7**, 907–915 (2004).
5. P. M. Pilarski *et al.*, "Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning" in *IEEE International Conference on Rehabilitation Robotics* (IEEE, 2011), pp. 1–7.
6. G. J. Tesauro, Temporal difference learning and TD-gammon. *Commun. ACM* **38**, 58–68 (1995).
7. M. Bowling, N. Burch, M. Johanson, O. Tammelin, Heads-up limit hold'em poker is solved. *Science* **347**, 145–149 (2015).
8. J. Randløv, P. Alstrøm, "Learning to drive a bicycle using reinforcement learning and shaping" in *Proceedings of the International Conference on Machine Learning (ICML)* (Morgan Kaufmann Publishers, Inc., 1998), pp. 463–471.
9. A. Y. Ng, H. J. Kim, M. I. Jordan, S. Sastry, "Autonomous helicopter flight via reinforcement learning" in *Advances in Neural Information Processing Systems (NIPS)* (MIT Press, 2003), pp. 799–806.
10. V. Mnih *et al.*, Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).
11. D. Silver *et al.*, Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–503 (2016).
12. D. Silver *et al.*, Mastering the game of Go without human knowledge. *Nature* **550**, 354–359 (2017).
13. D. Silver *et al.*, A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* **362**, 1140–1144 (2018).
14. P. Tsividis, T. Pouncy, J. Xu, J. Tenenbaum, S. Gershman, "Human learning in Atari" in *AAAI Spring Symposium Series* (AAAI Press, 2017).
15. B. F. Skinner, *The Behavior of Organisms: An Experimental Analysis* (Appleton-Century, 1938).
16. C. L. Hull, *Principles of Behavior* (Appleton-Century, New York, NY, 1943).
17. A. Ng, S. Russell, "Algorithms for inverse reinforcement learning" in *Proceedings of the International Conference on Machine Learning (ICML)* (Morgan Kaufmann Publishers, Inc., 2000), pp. 663–670.
18. M. L. Puterman, *Markov Decision Processes—Discrete Stochastic Dynamic Programming* (John Wiley & Sons, Inc., 1994).
19. R. E. Bellman, *Dynamic Programming* (Princeton University Press, 1957).
20. D. P. Bertsekas, J. N. Tsitsiklis, *Neuro-Dynamic Programming* (Athena Scientific, 1996).
21. R. Munos, "Error bounds for approximate policy iteration" in *Proceedings of the International Conference on Machine Learning (ICML)* (AAAI Press, 2003), pp. 560–567.
22. R. S. Sutton, Learning to predict by the methods of temporal differences. *Mach. Learn.* **3**, 9–44 (1988).
23. C. Watkins, P. Dayan, Q-learning. *Mach. Learn.* **8**, 279–292 (1992).
24. R. Howard, *Dynamic Programming and Markov Processes* (MIT Press, 1960).
25. T. Jaakkola, M. I. Jordan, S. P. Singh, On the convergence of stochastic iterative dynamic programming algorithms. *Neural Comput.* **6**, 1185–1201 (1994).
26. R. S. Sutton *et al.*, "Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction" in *International Joint Conference on Autonomous Agents and Multiagent Systems* (International Foundation for Autonomous Agents and Multiagent Systems, 2011), pp. 761–768.

27. T. Schaul, D. Horgan, K. Gregor, D. Silver, "Universal value function approximators" in *International Conference on Machine Learning (ICML)* (PMLR, 2015), vol. 37, pp. 1312–1320.
28. A. Barreto *et al.*, "Successor features for transfer in reinforcement learning" in *Advances in Neural Information Processing Systems (NIPS)* (Curran Associates, Inc., 2017), pp. 4055–4065.
29. P. Dayan, Improving generalization for temporal difference learning: The successor representation. *Neural Comput.* **5**, 613–624 (1993).
30. A. Barreto *et al.*, "Transfer in deep reinforcement learning using successor features and generalised policy improvement" in *Proceedings of the International Conference on Machine* (PMLR, 2018), vol. 80, pp. 501–510.
31. A. Barreto *et al.*, "The option keyboard: Combining skills in reinforcement learning" in *Advances in Neural Information Processing Systems (NeurIPS)* (Curran Associates, Inc., 2019), pp. 13052–13062.
32. A. G. Barto, S. Mahadevan, Recent advances in hierarchical reinforcement learning. *Discrete Event Dyn. Syst.* **13**, 341–379 (2003).
33. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (MIT Press, 2016).
34. R. Caruana, Multitask learning. *Mach. Learn.* **28**, 41–75 (1997).
35. R. S. Sutton, D. Precup, S. Singh, Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.* **112**, 181–211 (1999).
36. S. Thrun, "Is learning the N-th thing any easier than learning the first?" in *Advances in Neural Information Processing Systems (NIPS)* (MIT Press, 1996), pp. 640–646.
37. D. Borsa *et al.*, "Universal successor features approximators" in *International Conference on Learning Representations (ICLR)* (2019). https://openreview.net/forum?id=S1VWjiRcKX. Accessed 5 August 2020.
38. J. Hunt, A. Barreto, T. Lillicrap, N. Heess, "Composing entropic policies using divergence correction" in *Proceedings of the International Conference on Machine Learning (ICML)* (PMLR, 2019), vol. 97, pp. 2911–2920.
39. S. Hansen *et al.*, "Fast task inference with variational intrinsic successor features" in *International Conference on Learning Representations (ICLR)* (2020). https://openreview.net/forum?id=BJeAHkrYDS. Accessed 5 August 2020.