Hierarchy Through Composition with Multitask LMDPs

Andrew M. Saxe [1] **Adam C. Earle** [2] **Benjamin Rosman** [2 3]

Abstract

Hierarchical architectures are critical to the scalability of reinforcement learning methods. Most current hierarchical frameworks execute actions serially, with macro-actions comprising sequences of primitive actions. We propose a novel alternative to these control hierarchies based on concurrent execution of many actions in parallel. Our scheme exploits the guaranteed concurrent compositionality provided by the linearly solvable Markov decision process (LMDP) framework, which naturally enables a learning agent to draw on several macro-actions simultaneously to solve new tasks. We introduce the Multitask LMDP module, which maintains a parallel distributed representation of tasks and may be stacked to form deep hierarchies abstracted in space and time.

1. Introduction

Real world tasks unfold at a range of spatial and temporal scales, such that learning solely at the finest scale is likely to be slow. Hierarchical reinforcement learning (HRL) (Barto & Madadevan, 2003; Parr & Russell, 1998; Dietterich, 2000) attempts to remedy this by learning a nested sequence of ever more detailed plans. Hierarchical schemes have a number of desirable properties. Firstly, they are intuitive, as humans seldom plan at the level of raw actions, typically preferring to reason at a higher level of abstraction (Botvinick et al., 2009; Ribas-Fernandes et al., 2011; Solway et al., 2014). Secondly, they constitute one approach to tackling the curse of dimensionality (Bellman, 1957; Howard, 1960). In real world MDPs, the number of states typically grows dramatically in the size of a domain. A similar curse of dimensionality afflicts actions, such that a robot with multiple joints, for instance, must operate in the product space of actions for each joint individually (Mausam & Weld, 2008). Finally, the 'tasks' performed by an agent may come in great variety, and can also suffer from a curse of dimensionality: a robot that learns policies to navigate to each of ten rooms would need to learn 10 choose 2 policies to navigate to the closer of each pair of rooms. Transferring learning across tasks is therefore vital (Taylor & Stone, 2009; Foster & Dayan, 2002; Drummond, 1998; Fernández & Veloso, 2006; Bonarini et al., 2006; Barreto et al., 2016).

Many HRL schemes rely on a serial call/return procedure, in which temporally extended macro-actions or 'options' can call other (macro-)actions. In this sense, these schemes draw on a computer metaphor, in which a serial processor chooses sequential primitive actions, occasionally pushing or popping new macro-actions onto a stack. The influential options framework (Sutton et al., 1999), MAXQ method (Dietterich, 2000; Jonsson & Gómez, 2016) and the hierarchy of abstract machines (Parr & Russell, 1998; Burridge et al., 1999) all share this sequential-execution structure.

Concurrent MDPs (Mausam & Weld, 2008) relax the assumption of serial execution to allow multiple actions to be executed simultaneously at each time step, at the cost of a combinatorial increase in the size of the action space. In this paper, we develop a novel hierarchical scheme with a parallel and distributed execution structure that overcomes the combinatorial increase in action space by exploiting the guaranteed optimal compositionality afforded by the linearly solvable Markov decision process (LMDP) framework (Todorov, 2006; Kappen, 2005).

We present a Multitask LMDP module that executes concurrent blends of previously learned tasks. Here we use the term 'concurrent' or 'parallel' to refer to 'parallel distributed processing' or neural network-like methods that form weighted blends of many simple elements to achieve their aim. In standard schemes, if a robot arm has acquired policies to individually reach two points in some space, this knowledge typically does not aid it in optimally reaching to either point. However in our scheme, such behaviours can be expressed as different weighted task blends, such that an agent can simultaneously draw on several sub-policies to achieve goals not explicitly represented by any of the individual behaviours, including tasks the agent has never performed before.

[1]Center for Brain Science, Harvard University [2]School of Computer Science and Applied Mathematics, University of the Witwatersrand [3]Council for Scientific and Industrial Research, South Africa. Correspondence to: Andrew M. Saxe <asaxe@fas.harvard.edu>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

Next we show how this Multitask LMDP module can be stacked, resulting in a hierarchy of more abstract modules that communicate distributed representations of tasks between layers. We give a simple theoretical analysis showing that hierarchy can yield qualitative efficiency improvements in learning time and memory. Finally, we demonstrate the operation of the method on a navigation domain, and show that its multitasking ability can speed learning of new tasks compared to a traditional options-based agent.

Our scheme builds on a variety of prior work. Like the options framework (Sutton et al., 1999), we build a hierarchy in time. Similar to MAXQ Value Decomposition (Dietterich, 2000), we decompose a target MDP into a hierarchy of smaller SMDPs which progressively abstract over states. From Feudal RL, we draw the idea of a managerial hierarchy in which higher layers prescribe goals but not details for lower layers (Dayan & Hinton, 1993). Most closely related to our scheme, (Jonsson & Gómez, 2016) develop a MAXQ decomposition within the LMDP formalism (see Supplementary Material for extended discussion). Our method differs from all of the above approaches in permitting a graded, concurrent blend of tasks at each level, and developing a uniform, stackable module capable of performing a variety of tasks.

2. The Multitask LMDP: A compositional action module

Our goal in this paper is to describe a flexible action-selection module which can be stacked to form a hierarchy, such that the full action at any given point in time is composed of the concurrent composition of sub-actions within sub-actions. By analogy to perceptual deep networks, restricted Boltzmann machines (RBMs) form a component module from which a deep belief network can be constructed by layerwise stacking (Hinton et al., 2006; Hinton & Salakhutdinov, 2006). We seek a similar module in the context of action or control. This section describes the module, the Multitask LMDP (MLMDP), before turning to how it can be stacked. Our formulation relies on the linearly solvable Markov decision process (LMDP) framework introduced by Todorov (2006) (see also Kappen (2005)). The LMDP differs from the standard MDP formulation in fundamental ways, and enjoys a number of special properties. We first briefly describe the canonical MDP formulation, in order to explain what the switch to the LMDP accomplishes and why it is necessary.

2.1. Canonical MDPs

In its standard formulation, an MDP is a four-tuple $M = \langle S, A, P, R\rangle$, where S is a set of states, A is a set of discrete actions, P is a transition probability distribution $P : S \times A \times S \to [0, 1]$, and R is an expected instantaneous reward function $R : S \times A \to \mathbb{R}$. The goal is to determine an optimal policy $\pi : S \to A$ specifying which action to take in each state. This optimal policy can be computed from the optimal value function $V : S \to \mathbb{R}$, defined as the expected reward starting in a given state and acting optimally thereafter. The value function obeys the well-known Bellman optimality condition

$$V(s) = \max_{a\in A} \left\{ R(s,a) + \sum_{s'} P(s'|s,a)V(s') \right\}. \quad (1)$$

This formalism is the basis of most practical and theoretical studies of decision-making under uncertainty and reinforcement learning (Bellman, 1957; Howard, 1960; Sutton & Barto, 1998). See, for instance, (Mnih et al., 2015; Lillicrap et al., 2015; Levine et al., 2016) for recent successes in challenging domains.

For the purposes of a compositional hierarchy of actions, this formulation presents two key difficulties.

1. **Mutually exclusive sequential actions** First, the agent's actions are discrete and execute serially. Exactly one (macro-)action operates at any given time point. Hence there is no way to build up an action at a single time point out of several 'subactions' taken in parallel. For example, a control signal for a robotic arm cannot be composed of a control decision for the elbow joint, a control decision for the shoulder joint, and a control decision for the gripper, each taken in parallel and combined into a complete action for a specific time point.

2. **Non-composable optimal policies** The maximization in Eqn. (1) over a discrete set of actions is nonlinear. This means that optimal solutions, in general, do not compose in a simple way. Consider two standard MDPs $M_1 = \langle S, A, P, R_1\rangle$ and $M_2 = \langle S, A, P, R_2\rangle$ which have identical state spaces, action sets, and transition dynamics but differ in their instantaneous rewards R_1 and R_2. These may be solved independently to yield value functions V_1 and V_2. But the value function of the MDP $M_{1+2} = \langle S, A, P, R_1 + R_2\rangle$, whose instantaneous rewards are the sum of the first two, is not $V_{1+2} = V_1 + V_2$. In general, there is no simple procedure for deriving V_{1+2} from V_1 and V_2; it must be found by solving Eqn. (1) again.

2.2. Linearly Solvable MDPs

The LMDP (Todorov, 2009a; Dvijotham & Todorov, 2010; Todorov, 2009b; Dvijotham & Todorov, 2010) is defined by a three-tuple $L = \langle S, P, R\rangle$, where S is a set of states, P is a passive transition probability distribution $P : S \times S \to [0, 1]$, and R is an expected instantaneous reward function $R : S \to \mathbb{R}$. The LMDP framework replaces the traditional discrete set of actions A with a continuous probability distribution over next states $a : S \times S \to [0, 1]$. That is, the 'control' or 'action' chosen by the agent in state s is a transition probability distribution over next states, $a(\cdot|s)$. The controlled transition distribution may be interpreted either as directly constituting the agent's dynamics, or as a stochastic policy over deterministic actions which affect state transitions (Todorov, 2006; Jonsson & Gómez, 2016). Swapping a discrete action space for a continuous action space is a key change which will allow for concurrently selected 'subactions' and distributed representations.

The LMDP framework additionally posits a specific form for the cost function to be optimized. The instantaneous reward for taking action $a(\cdot|s)$ in state s is

$$\mathcal{R}(s, a) = R(s) - \lambda \mathrm{KL}\left(a(\cdot|s)||P(\cdot|s)\right), \tag{2}$$

where the KL term is the Kullback-Leibler divergence between the selected control transition probability and the passive dynamics. This term implements a control cost, encouraging actions to conform to the natural passive dynamics of a domain. In a cart-pole balancing task, for instance, the passive dynamics might encode the transition structure arising from physics in the absence of control input. Any deviation from these dynamics will require energy input. In more abstract settings, such as navigation in a 2D grid world, the passive dynamics might encode a random walk, expressing the fact that actions cannot transition directly to a far away goal but only move some limited distance in a specific direction. Examples of standard benchmark domains in the LMDP formalism are provided in the Supplementary Material. The parameter λ in Eqn. (2) acts to trade-off the relative value between the reward of being in a state and the control cost, and determines the stochasticity of the resulting policies.

We consider first-exit problems (see Dvijotham & Todorov (2011) for infinite horizon and other formulations), in which the state space is divided into a set of absorbing *boundary* states $\mathcal{B} \subset S$ and non-absorbing *interior* states $\mathcal{I} \subset S$, with $S = \mathcal{B} \cup \mathcal{I}$. In this formulation, an agent acts in a variable length episode that consists of a series of transitions through interior states before a final transition to a boundary state which terminates the episode. The goal is to find the policy a^* which maximizes the total expected reward across the episode,

$$a^* = \mathrm{argmax}_a \mathbb{E}_{\substack{s_{t+1}\sim a(\cdot|s_t) \\ \tau=\min\{t:s_t\in\mathcal{B}\}}} \left\{ \sum_{t=1}^{\tau-1} \mathcal{R}(s_t, a) + R(s_\tau) \right\}. \tag{3}$$

Because of the carefully chosen structure of the reward $\mathcal{R}(s, a)$ and the continuous action space, the Bellman equation simplifies greatly. In particular define the *desirability* function $z(s) = e^{V(s)/\lambda}$ as the exponentiated cost-to-go function, and define $q(s) = e^{R(s)/\lambda}$ to be the exponentiated instantaneous rewards. Let N be the number of states, and N_i and N_b be the number of internal and boundary states respectively. Represent $z(s)$ and $q(s)$ with N-dimensional column vectors z and q, and the transition dynamics $P(s'|s)$ with the N-by-N_i matrix P, where column index corresponds to s and row index corresponds to s'. Let z_i and z_b denote the partition of z into internal and boundary states, respectively, and similarly for q_i and q_b. Finally, let P_i denote the N_i-by-N_i submatrix of P containing transitions between internal states, and P_b denote the N_b-by-N_i submatrix of P containing transitions from internal states to boundary states.

As shown in Todorov (2009b), the Bellman equation in this setting reduces to

$$(I - M_i P_i^T) z_i = M_i P_b^T z_b \tag{4}$$

where $M_i = \mathrm{diag}(q_i)$ and, because boundary states are absorbing, $z_b = q_b$. The exponentiated Bellman equation is hence a linear system, the key advantage of the LMDP framework. A variety of special properties flow from the linearity of the Bellman equation, which we exploit in the following.

Solving for z_i may be done explicitly as $z_i = (I - M_i P_i^T)^{-1} M_i P_b^T z_b$ or via the z-iteration method (akin to value iteration),

$$z_i \leftarrow M_i P_i^T z_i + M_i P_b^T z_b. \tag{5}$$

Finally, the optimal policy may be computed in closed form as

$$a^*(s'|s) = \frac{P(s'|s) z(s')}{\mathcal{G}[z](s)}, \tag{6}$$

where the normalizing constant $\mathcal{G}[z](s) = \sum_{s'} P(s'|s) z(s')$. Detailed derivations of these results are given in (Todorov, 2009a;b; Dvijotham & Todorov, 2011). Intuitively, the hard maximization of Eqn. (1) has been replaced by a soft maximization $\log(\sum \exp(\cdot))$, and the continuous action space enables closed form computation of the optimal policy.

Compared to the standard MDP formulation, the LMDP has

1. **Continuous concurrent actions** Actions are expressed as transition probabilities over next states, such that these transition probabilities can be influenced by many subtasks operating in parallel, and in a graded fashion.

2. **Compositional optimal policies** In the LMDP, linearly blending desirability functions yields the correct composite desirability function (Todorov, 2009a;b). In particular, consider two LMDPs $L_1 = \langle S, P, q_i, q_b^1 \rangle$

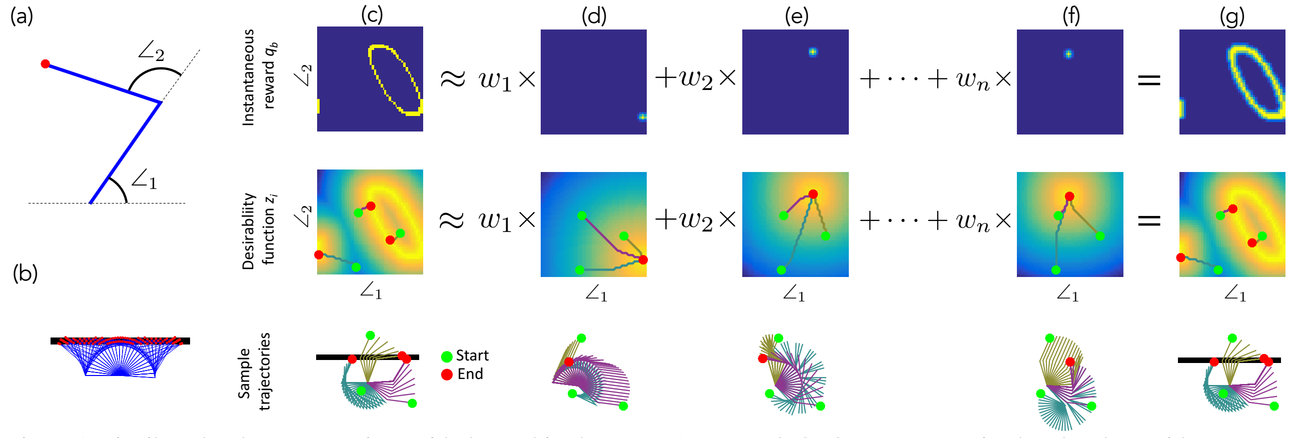

Figure 1. Distributed task representations with the Multitask LMDP. (a) Example 2DOF arm constrained to the plane with state space consisting of shoulder and elbow joint angles $\angle_1, \angle_2 \in [-\pi, \pi]$ respectively. (b) A novel task is specified as an instantaneous reward function over terminal states. In this example, the task "reach to the black rectangle" is encoded by rewarding any terminal state with the end effector in black rectangle (all successful configurations shown). (c-g) Solutions via the LMDP. Top row: Instantaneous rewards in the space of joint angles. Middle row: Optimal desirability function with sample trajectories starting at green circles, finishing at red circles. Bottom row: Strobe plot of sample trajectories in Cartesian space. Trajectories start at green circle, end at red circle. Column (c): The linear Bellman equation is solved for this particular instantaneous boundary reward structure to obtain the optimal value function. Columns (d-g): Solution via compositional Multitask LMDP. The instantaneous reward structure is expressed as a weighted combination of previously-learned subtasks (d-f), here chosen to be navigation to specific points in state space. (g) Because of the linearity of the Bellman equation, the resulting combined value function is optimal, and the system can act instantly despite no explicit training on the reach-to-rectangle task. The same fixed basis set can be used to express a wide variety of tasks (reach to a cross; reach to a circle; etc).

and $L_2 = \langle S, P, q_i, q_b^2 \rangle$ which have identical state spaces, transition dynamics, and internal reward structures, but differ in their exponentiated boundary rewards q_b^1 and q_b^2. These may be solved independently to yield desirability functions z^1 and z^2. The desirability function of the LMDP $L_{1+2} = \langle S, P, q_i, \alpha q_b^1 + \beta q_b^2 \rangle$, whose instantaneous rewards are a weighted sum of the first two, is simply $z^{1+2} = \alpha z^1 + \beta z^2$. This property follows from the linearity of Eqn. 4, and is the foundation of our hierarchical scheme.

2.3. The Multitask LMDP

To build a multitask action module, we exploit this compositionality to build a basis set of tasks (Foster & Dayan, 2002; Ruvolo & Eaton, 2013; Schaul et al., 2015; Pan et al., 2015; Borsa et al., 2016). Suppose that we learn a set of LMDPs $L_t = \langle S, P, q_i, q_b^t \rangle$, $t = 1, \cdots, N_t$ which all share the same state space, passive dynamics, and internal rewards, but differ in their instantaneous exponentiated boundary reward structure q_b^t, $t = 1, \cdots, N_t$. Each of these LMDPs corresponds to a different task, defined by its boundary reward structure, in the same overall state space (see Taylor & Stone (2009) for an overview of this and other notions of transfer and multitask learning). We denote the Multitask LMDP module M formed from these N_t LMDPs as $M = \langle S, P, q_i, Q_b \rangle$. Here the N_b-by-N_t task basis matrix $Q_b = \left[q_b^1 \; q_b^2 \; \cdots \; q_b^{N_t} \right]$ encodes a library of component tasks in the MLMDP. Solving each component LMDP yields the corresponding desirability functions z_i^t, $t = 1, \cdots, N_t$ for each task, which can also be formed into the N_i-by-N_t desirability basis matrix $Z_i = \left[z_i^1 \; z_i^2 \; \cdots \; z_i^{N_t} \right]$ for the multitask module.

When we encounter a new task defined by a novel instantaneous exponentiated reward structure q, if we can express it as a linear combination of previously learned tasks, $q = Q_b w$, where $w \in R^{N_t}$ is a vector of task blend weights, then we can instantaneously derive its optimal desirability function as $z_i = Z_i w$. This immediately yields the optimal action through Eqn. 6. Hence an MLMDP agent can act optimally even in never-before-seen tasks, provided that the new task lies in the subspace spanned by previously learned tasks. This approach is an off-policy variant on the successor representation method of Dayan (1993). It differs by yielding the optimal value function, rather than the value function under a particular policy π; and allows arbitrary boundary rewards for the component tasks (the SR implicitly takes these to be a positive reward on just one state). Also related is the successor feature approach of Barreto et al. (2016), which permits performance guarantees for transfer to new tasks. With successor features, new tasks must have rewards that are close in Euclidean distance to prior tasks, whereas here we require only that they lie in the span of prior tasks.

More generally, if the target task q is not an exact linear combination of previously learned tasks, an approximate task weighting w can be found as

$$\text{argmin}_w \|q - Q_b w\| \quad \text{subject to} \quad Q_b w \geq 0. \tag{7}$$

The technical requirement $Q_b w \geq 0$ is due to the relationship $q = \exp(r/\lambda)$, such that negative values of q are not possible. In practice this can be approximately solved as $w = Q_b^\dagger q$, where $\dagger$ denotes the pseudoinverse, and negative elements of w are projected back to zero.

Here the coefficients of the task blend w constitute a distributed representation of the current task to be performed. Although the set of basis tasks Q_b is fixed and finite, they permit an infinite space of tasks to be performed through their concurrent linear composition. Figure 2.2 demonstrates this ability of the Multitask LMDP module in the context of a 2D robot arm reaching task. From knowledge of how to reach individual points in space, the module can instantly act optimally to reach to a rectangular region.

3. Stacking the module: Concurrent Hierarchical LMDPs

To build a hierarchy out of this Multitask LMDP module, we construct a stack of MLMDPs in which higher levels select the instantaneous reward structure that defines the current task for lower levels. To take a navigation example, a high level module might specify that the lower level module should reach room A but not B by placing instantaneous rewards in room A but no rewards in room B. Crucially, the fine details of achieving this subgoal can be left to the low-level module. Critical to the success of this hierarchical scheme is the flexible, optimal composition afforded by the Multitask LMDP module: the specific reward structure commanded by the higher layer will often be novel for the lower level, but will still be performed optimally provided it lies in the basis of learned tasks.

3.1. Constructing a hierarchy of MLMDPs

We start with the MLMDP $M^1 = \langle S^1, P^1, q_i^1, Q_b^1 \rangle$ that we must solve, where here the superscript denotes the hierarchy level (Fig. 2). This serves as the base case for our recursive scheme for generating a hierarchy. For the inductive step, given an MLMDP $M^l = \langle S^l, P^l, q_i^l, Q_b^l \rangle$ at level l, we augment the state space $\tilde{S}^l = S^l \cup S_t^l$ with a set of N_t terminal boundary states S_t^l that we call *subtask* states. Semantically, entering one of these states will correspond to a decision by the layer l MLMDP to access the next level of the hierarchy. The transitions to subtask states are governed by a new N_t^l-by-N_i^l passive dynamics matrix P_t^l, which is chosen by the designer to encode the structure of the domain. Choosing fewer subtask states than interior states will yield a higher level which operates in a smaller state space, yielding state abstraction. In the augmented MLMDP, the passive dynamics become $\tilde{P}^l = \mathcal{N}([P_i^l;\ P_b^l;\ P_t^l])$ where the operation $\mathcal{N}(\cdot)$ renormalizes each column to sum to one.

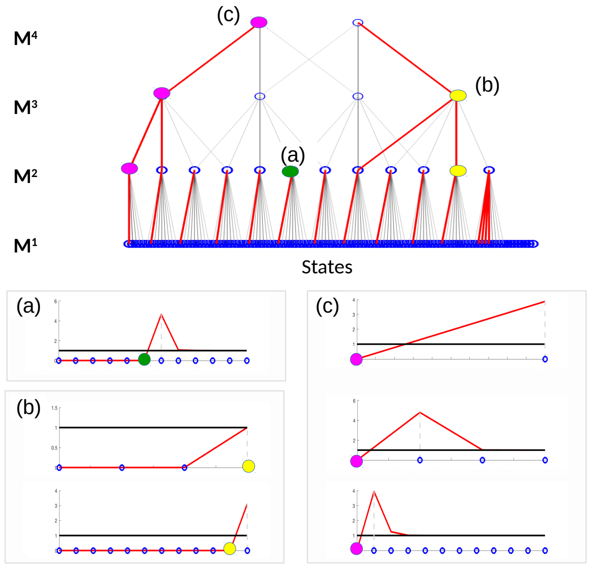

Figure 2. Stacking Multitask LMDPs. Top: Deep hierarchy for navigation through a 1D corridor. Lower-level MLMDPs are abstracted to form higher-level MLMDPs by choosing a set of 'subtask' states which can be accessed by the lower level (grey lines between levels depict passive subtask transitions P_t^l). Lower levels access these subtask states to indicate completion of a subgoal and to request more information from higher levels; higher levels communicate new subtask state instantaneous rewards, and hence the concurrent task blend, to the lower levels. Red lines indicate higher level access points for one sample trajectory starting from leftmost state and terminating at rightmost state. Bottom: Panels (a-c) depict distributed task blends arising from accessing the hierarchy at points denoted in left panel. The higher layer states accessed are indicated by filled circles. (a) Just the second layer of hierarchy is accessed, resulting in higher weight on the task to achieve the next subgoal and zero weights on already achieved subgoals. (b) The second and third levels are accessed, yielding new task blends for both. (c) All levels are accessed yielding task blends at a range of scales.

It remains to specify the matrix of subtask-state instantaneous rewards R_t^l for this augmented MLMDP. Often hierarchical schemes require designing a pseudoreward function to encourage successful completion of a subtask. Here we also pick a set of reward functions over subtask states; however, the performance of our scheme is only weakly dependent on this choice: we require only that our chosen reward functions form a good basis for the set of subtasks that the higher layer will command. Any set of tasks which can linearly express the required space of reward structures specified by the higher level is suitable. In our experiments, we define N_t^l tasks, one for each subtask state, and set each instantaneous reward to negative values on all but a single 'goal' subtask state. Then the augmented MLMDP is

$\tilde{M}^l = \left\langle \tilde{S}^l, \tilde{P}^l, q_i^l, \left[Q_b^l\, 0; 0\, Q_t^l\right] \right\rangle$.

The higher level is itself an MLMDP $M^{l+1} = \left\langle S_t^l, P^{l+1}, q_i^{l+1}, Q_b^{l+1} \right\rangle$, defined not over the entire state space but just over the N_t^l subtask states of the layer below. To construct this, we must compute an appropriate passive dynamics and reward structure. A natural definition for the passive dynamics is the probability of starting at one subtask state and terminating at another under the lower layer's passive dynamics,

$$P_i^{l+1} = \tilde{P}_t^l (I - \tilde{P}_i^l)^{-1} \tilde{P}_t^{l^T}, \tag{8}$$
$$P_b^{l+1} = \tilde{P}_b^l (I - \tilde{P}_i^l)^{-1} \tilde{P}_t^{l^T}. \tag{9}$$

In this way, the higher-level LMDP will incorporate the transition constraints from the layer below. The interior-state reward structure can be similarly defined, as the reward accrued under the passive dynamics from the layer below. However for simplicity in our implementation, we simply set small negative rewards on all internal states.

Hence, from a base MLMDP M^l and subtask transition matrix P_t^l, the above construction yields an augmented MLDMP $\tilde{M}^l$ at the same layer and unaugmented MLDMP M^{l+1} at the next higher layer. This procedure may be iterated to form a deep stack of MLMDPs $\left\{\tilde{M}^1, \tilde{M}^2, \cdots, \tilde{M}^{D-1}, M^D\right\}$, where all but the highest is augmented with subtask states. The key choice for the designer is P_t^l, the transition structure from internal states to subtask states for each layer. Through Eqns. (8)-(9), this matrix specifies the state abstraction used in the next higher layer. Fig. 2 illustrates this scheme for an example of navigation through a 1D corridor.

3.2. Instantaneous rewards and task blends: Communication between layers

Bidirectional communication between layers happens via subtask states and their instantaneous rewards. The higher layer sets the instantaneous boundary rewards over subtask states for the lower layer; and the lower layer signals that it has completed a subtask and needs new guidance from the higher layer by transitioning to a subtask state.

In particular, suppose we have solved a higher-level MLMDP using any method we like, yielding the optimal action a^{l+1}. This will make transitions to some states more likely than they would be under the passive dynamics, indicating that they are more attractive than usual for the current task. It will make other transitions less likely than the passive dynamics, indicating that transitions to these states should be avoided. We therefore define the instantaneous rewards for the subtask states at level l to be proportional to the difference between controlled and passive dynamics at the higher level $l+1$,

$$r_t^l \propto a_i^{l+1}(\cdot|s) - p_i^{l+1}(\cdot|s). \tag{10}$$

This effectively inpaints extra rewards for the lower layer, indicating which subtask states are desirable from the perspective of the higher layer.

The lower layer MLMDP then uses its basis of tasks to determine a task weighting w^l which will optimally achieve the reward structure r_t^l specified by the higher layer by solving (7). The reward structure specified by the higher layer may not correspond to any one task learned by the lower layer, but it will nonetheless be performed well by forming a concurrent blend of many different tasks and leveraging the compositionality afforded by the LMDP framework. This scheme may also be interpreted as implementing a parametrized option, but with performance guarantees for any parameter setting (Masson et al., 2016).

We now describe the execution model (see Supplementary Material for pseudocode listing). The true state of the agent is represented in the base level $\tilde{M}^1$, and next states are drawn from the controlled transition distribution. If the next state is an interior state, one unit of time passes and the state is updated as usual. If the next state is a subtask state, the next layer of the hierarchy is accessed at its corresponding state; no 'real' time passes during this transition. The higher level then draws its next state, and in so doing can access the next level of hierarchy by transitioning to one of its subtask states, and so on. At some point, a level will elect not to access a subtask state; it then transmits its desired rewards from Eqn. (10) to the layer below it. The lower layer then solves its multitask LMDP problem to compute its own optimal actions, and the process continues down to the lowest layer $\tilde{M}^1$ which, after updating its optimal actions, again draws a transition. Lastly, if the next state is a terminal boundary state, the layer terminates itself. This corresponds to a level of the hierarchy determining that it no longer has useful information to convey. Terminating a layer disallows future transitions from the lower layer to its subtask states, and corresponds to inpainting infinite negative rewards onto the lower level subtask states.

4. Computational complexity advantages of hierarchical decomposition

To concretely illustrate the value of hierarchy, consider navigation through a 1D ring of N states, where the agent must perform N different tasks corresponding to navigating to each particular state. We take the passive dynamics to be local (a nonzero probability of transitioning just to adjacent states in the ring, or remaining still). In one step of Z iteration (Eqn. 5), the optimal value function progresses at best $O(1)$ states per iteration because of the local passive dynamics (see Precup et al. (1998) for a similar argument). It therefore requires $O(N)$ iterations in a flat implementation for a useful value function signal to arrive at the furthest point in the ring for each task. As there are N tasks, the flat implementation requires $O(N^2)$ iterations to learn all of them.

Instead suppose that we construct a hierarchy by placing a subtask every $M = \log N$ states, and do this recursively to form D layers. The recursion terminates when $N/M^D \approx 1$, yielding $D \approx \log_M N$. With the correct higher level policy sequencing subtasks, each policy at a given layer only needs to learn to navigate between adjacent subtasks, which are no more than M states apart. Hence Z iteration can be terminated after $O(M)$ iterations. At level $l = 1, 2, \cdots, D$ of the hierarchy, there are N/M^l subtasks, and N/M^{l-1} boundary reward tasks to learn. Overall this yields

$$\sum_{l=1}^{D} M \left(\frac{N}{M^l} + \frac{N}{M^{l-1}} \right) \quad \approx \quad O(N \log N)$$

total iterations (see Supplementary Material for derivation and numerical verification). A similar analysis shows that this advantage holds for memory requirements as well. The flat scheme requires $O(N^2)$ nonzero elements of Z to encode all tasks, while the hierarchical scheme requires only $O(N \log N)$. Hence hierarchical decomposition can yield qualitatively more efficient solutions and resource requirements, reminiscent of theoretical results obtained for perceptual deep learning (Bengio, 2009; Bengio & LeCun, 2007). We note that this advantage only occurs in the multitask setting: the flat scheme can learn one specific task in time $O(N)$. Hence hierarchy is beneficial when performing an ensemble of tasks, due to the reuse of component policies across many tasks (see also Solway et al. (2014)).

5. Experiments

5.1. Conceptual Demonstration

To illustrate the operation of our scheme, we apply it to a 2D grid-world 'rooms' domain (Fig. 3(a)). The agent is required to navigate through an environment consisting of four rooms with obstacles to a goal location in one of the rooms. The agent can move in the four cardinal directions or remain still. To build a hierarchy, we place six higher layer subtask goal locations throughout the domain (Fig. 3(a), red dots). The inferred passive dynamics for the higher layer MLMDP is shown in Fig. 3(a) as weighted lines between these subtask states. The higher layer passive dynamics conform to the structure of the problem, with the probability of transition between higher layer states roughly proportional to the distance between those states at the lower layer.

As a basic demonstration that the hierarchy conveys useful information, Fig. 3(b) shows Z-learning curves for the base layer policy with and without an omniscient hierarchy (i.e., one preinitialized using Z-iteration). From the start of learning, the hierarchy is able to drive the agent to the vicinity of the goal. Fig. 3(c-d) illustrates the evolving distributed task representation commanded by the higher layer to the lower layer over the course of a trajectory. At each time point, several subtasks have nonzero weight, highlighting the concurrent execution in the system.

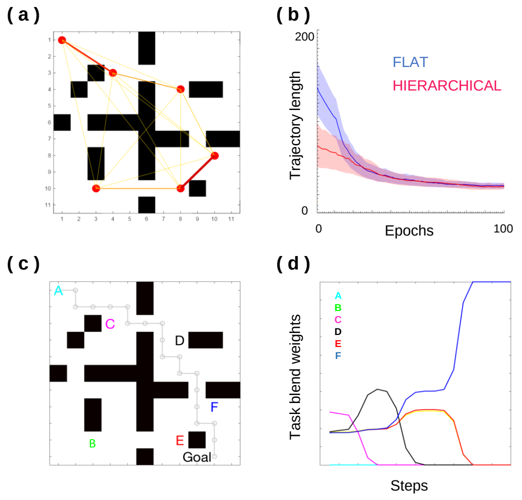

Figure 3. Rooms domain. (a) Four room domain with subtask locations marked as red dots, free space in white and obstacles in black, and derived higher-level passive dynamics shown as weighted links between subtasks. (b) Convergence as trajectory length over learning epochs with and without the help of the hierarchy. (c) Sample trajectory (gray line) from upper left to goal location in bottom right. (d) Evolution of distributed task weights on each subtask location over the course of the trajectory in panel (c).

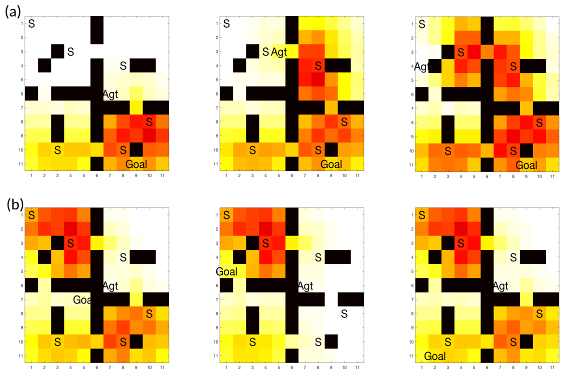

Figure 4. Desirability functions. (a) Desirability function over states as agent moves through environment, showing the effect of reward inpainting from the hierarchy. (b) Desirability function over states as goal location moves. Higher layers inpaint rewards into subtasks that will move the agent nearer the goal.

Fig. 4(a) shows the composite desirability function resulting from the concurrent task blend for different agent locations. Fig. 4(b) highlights the multitasking ability of the system, showing the composite desirability function as the goal location is moved. The leftmost panel, for instance, shows that rewards are painted concurrently into the upper left and bottom right rooms, as these are both equidistant to the goal.

5.2. Quantitative Comparison

In order to quantitatively demonstrate the performance improvements possible with our method we consider the problem of a mobile robot navigating through an office block in search of a charging station. This domain is shown in Fig. 5, and is taken from previous experiments in transfer learning (Fernández & Veloso, 2006).

Although the agent again moves in one of the four cardinal directions at each time step, the agent is additionally provided with a policy to navigate to each room in the office block, with goal locations marked by an 'S' in Fig. 5. The goal of the agent is to learn a policy to navigate to the nearest charging station from any location in the office block, given that there are a number of different charging stations available (one per room, but only in some of the rooms). This corresponds to an 'OR' navigation problem: navigating to location A OR B, while knowing separately a policy to get to A, and a policy to get to B. Concretely, the problem is to navigate to the nearest of two unknown subtask locations. The agent is randomly initialized at interior states, and the trajectory lengths are capped at 500 steps.

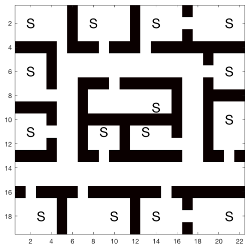

Figure 5. The office building domain. Each 'subtask' or 'option' policy navigates the agent to one of the rooms.

We compare a one-layer deep implementation of our method to a simple implementation of the options framework (Sutton et al., 1999). In the options framework the agent's action space is augmented with the full set of option policies. The initialization set for these options is the full state space, so that any option may be executed at any time. The termination condition is defined such that the option terminates only when it reaches its goal state. To minimize the action space for the options agent, we remove the primitive actions, reducing the learning problem to simply choosing the single correct option from each state. The options learning problem is solved using Q-learning with sigmoidal learning rate decrease and ϵ-greedy exploitation. These parameters were optimized on a coarse grid to yield the fastest learning curves. Results are averaged over 20 runs.

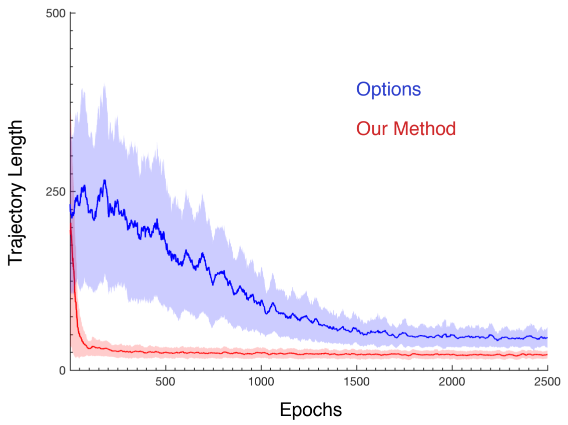

Figure 6. Learning rates. Our method jump-starts performance in the OR task, while the options agent must learn a state-action mapping.

Fig. 6 shows that our agent receives a significant jump-start in learning. By leveraging the distributed representation provided by our scheme, the agent is required only to learn when to request information from the higher layer. Although the task is novel, the higher layer can express it as a combination of prior tasks. Conversely, the options agent must learn a unique mapping between all states and actions. This issue is exacerbated as the number of available options grows (Rosman & Ramamoorthy, 2012).

6. Conclusion

The Multitask LMDP module provides a novel approach to control hierarchies, based on a distributed representation of tasks and parallel execution. Rather than learn to perform one task or a fixed library of tasks, it exploits the compositionality provided by linearly solvable Markov decision processes to perform an infinite space of task blends optimally. Stacking the module yields a deep hierarchy abstracted in state space and time, with the potential for qualitative efficiency improvements.

Experimentally, we have shown that the distributed representation provided by our framework can speed up learning in a simple navigation task, by representing a new task as a combination of prior tasks.

While a variety of sophisticated reinforcement learning methods have made use of deep networks as capable function approximators (Mnih et al., 2015; Lillicrap et al., 2015; Levine et al., 2016), in this work we have sought to transfer some of the underlying intuitions, such as parallel distributed representations and stackable modules, to the control setting. In the future this may allow other elements of the deep learning toolkit to be brought to bear in this setting, most notably gradient-based learning of the subtask structure itself.

Acknowledgements

We thank our reviewers for thoughtful comments. AMS acknowledges support from the Swartz Program in Theoretical Neuroscience at Harvard. AE and BR acknowledge support from the National Research Foundation of South Africa.

References

Barreto, A., Munos, R., Schaul, T., and Silver, D. Successor Features for Transfer in Reinforcement Learning. *arXiv*, 2016.

Barto, A.G. and Madadevan, S. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems: Theory and Applications*, (13):41–77, 2003.

Bellman, R.E. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

Bengio, Y. Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

Bengio, Y. and LeCun, Y. Scaling learning algorithms towards AI. In Bottou, L., Chapelle, O., DeCoste, D., and Weston, J. (eds.), *Large-Scale Kernel Machines*. MIT Press, 2007.

Bonarini, A., Lazaric, A., and Restelli, M. Incremental Skill Acquisition for Self-motivated Learning Animats. In Nolfi, S., Baldassare, G., Calabretta, R., Hallam, J., Marocco, D., Miglino, O., Meyer, J.-A., and Parisi, D. (eds.), *Proceedings of the Ninth International Conference on Simulation of Adaptive Behavior (SAB-06)*, volume 4095, pp. 357–368, Heidelberg, 2006. Springer Berlin.

Borsa, D., Graepel, T., and Shawe-Taylor, J. Learning Shared Representations in Multi-task Reinforcement Learning. *arXiv*, 2016.

Botvinick, M.M., Niv, Y., and Barto, A.C. Hierarchically organized behavior and its neural foundations: a reinforcement learning perspective. *Cognition*, 113(3):262–80, 12 2009.

Burridge, R. R., Rizzi, A. A., and Koditschek, D.E. Sequential Composition of Dynamically Dexterous Robot Behaviors. *The International Journal of Robotics Research*, 18(6):534–555, 6 1999.

Dayan, P. Improving Generalization for Temporal Difference Learning: The Successor Representation. *Neural Computation*, 5(4):613–624, 7 1993.

Dayan, P. and Hinton, G. Feudal Reinforcement Learning. In *NIPS*, 1993.

Dietterich, T.G. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Drummond, C. Composing functions to speed up reinforcement learning in a changing world. In Nédellec, C. and Rouveirol, C. (eds.), *Machine Learning: ECML-98.*, pp. 370–381, Heidelberg, 1998. Springer Berlin.

Dvijotham, K. and Todorov, E. Inverse Optimal Control with Linearly-Solvable MDPs. In *ICML*, 2010.

Dvijotham, K. and Todorov, E. A unified theory of linearly solvable optimal control. *Uncertainty in Artificial Intelligence*, 2011.

Fernández, F. and Veloso, M. Probabilistic policy reuse in a reinforcement learning agent. *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 720, 2006.

Foster, D. and Dayan, P. Structure in the Space of Value Functions. *Machine Learning*, (49):325–346, 2002.

Hinton, G.E. and Salakhutdinov, R.R. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–7, 7 2006.

Hinton, G.E., Osindero, S., and Teh, Y.-W. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18:1527–1554, 2006.

Howard, R.A. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

Jonsson, A. and Gómez, V. Hierarchical Linearly-Solvable Markov Decision Problems. In *ICAPS*, 2016.

Kappen, H.J. Linear Theory for Control of Nonlinear Stochastic Systems. *Physical Review Letters*, 95(20):200201, 11 2005.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-End Training of Deep Visuomotor Policies. *Journal of Machine Learning Research*, 17:1–40, 2016.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv*, 2015.

Masson, W., Ranchod, P., and Konidaris, G.D. Reinforcement Learning with Parameterized Actions. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 1934–1940, 9 2016.

Mausam and Weld, D.S. Planning with Durative Actions in Stochastic Domains. *Journal of Artificial Intelligence Research*, 31:33–82, 2008.

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

Pan, Y., Theodorou, E.A., and Kontitsis, M. Sample Efficient Path Integral Control under Uncertainty. In *NIPS*, 2015.

Parr, R. and Russell, S. Reinforcement learning with hierarchies of machines. In *NIPS*, 1998.

Precup, D., Sutton, R., and Singh, S. Theoretical results on reinforcement learning with temporally abstract options. In *ECML*, 1998.

Ribas-Fernandes, J.J.F., Solway, A., Diuk, C., McGuire, J.T., Barto, A.G., Niv, Y., and Botvinick, M.M. A neural signature of hierarchical reinforcement learning. *Neuron*, 71(2):370–9, 7 2011.

Rosman, B. and Ramamoorthy, S. What good are actions? Accelerating learning using learned action priors. In *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, number November. IEEE, 11 2012.

Ruvolo, P. and Eaton, E. ELLA: An efficient lifelong learning algorithm. *Proceedings of the 30th International Conference on Machine Learning*, 28(1):507–515, 2013.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal Value Function Approximators. *Proceedings of The 32nd International Conference on Machine Learning*, pp. 1312–1320, 2015.

Solway, A., Diuk, C., Córdova, N., Yee, D., Barto, A.G., Niv, Y., and Botvinick, M.M. Optimal Behavioral Hierarchy. *PLoS Computational Biology*, 10(8):e1003779, 8 2014.

Sutton, R.S. and Barto, A.G. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

Sutton, R.S., Precup, D., and Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 8 1999.

Taylor, M.E. and Stone, P. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

Todorov, E. Linearly-solvable Markov decision problems. In *NIPS*, 2006.

Todorov, E. Efficient computation of optimal actions. *Proceedings of the National Academy of Sciences*, 106(28):11478–11483, 7 2009a.

Todorov, E. Compositionality of optimal control laws. In *NIPS*, 2009b.