MODULAR MULTITASK REINFORCEMENT LEARNING WITH POLICY SKETCHES

Jacob Andreas, Dan Klein, and Sergey Levine
Computer Science Division
University of California, Berkeley
{jda,klein,svlevine}@eecs.berkeley.edu

ABSTRACT

We describe a framework for multitask deep reinforcement learning guided by *policy sketches*. Sketches annotate each task with a sequence of named subtasks, providing high-level structural relationships among tasks, but *not* providing the detailed guidance required by previous work on learning policy abstractions for RL (e.g. intermediate rewards, subtask completion signals, or intrinsic motivations). Our approach associates every subtask with its own modular subpolicy, and jointly optimizes over full task-specific policies by tying parameters across shared subpolicies. This optimization is accomplished via a simple decoupled actor–critic training objective that facilitates learning common behaviors from dissimilar reward functions. We evaluate the effectiveness of our approach on a maze navigation game and a 2-D Minecraft-inspired crafting game. Both games feature extremely sparse rewards that can be obtained only after completing a number of high-level subgoals (e.g. escaping from a sequence of locked rooms or collecting and combining various ingredients in the proper order). Experiments illustrate two main advantages of our approach. First, we outperform standard baselines that learn task-specific or shared monolithic policies. Second, our method naturally induces a library of primitive behaviors that can be recombined to rapidly acquire policies for new tasks.

1 INTRODUCTION

This paper describes a framework for learning composable deep subpolicies in a multitask setting, guided only by abstract *policy sketches*. We are interested in problems like the ones shown in Figure 1, with collections of tasks that involve sparse rewards and long-term planning, but which share structure in the form of common subgoals or reusable high-level actions. Our work aims to develop models that can learn efficiently from these sparse rewards and rapidly adapt to new tasks, by exploiting this shared structure and translating success on one task into progress on others. Our approach ultimately induces a library of high-level actions directly from symbolic annotations like the ones marked K_1 and K_2 in the figure.

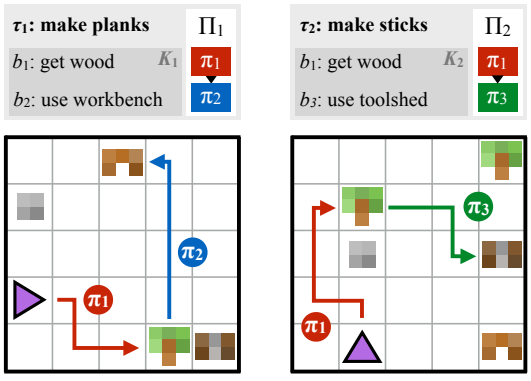


Figure 1: Composing policies from subpolicies. Here we have simplified versions of two tasks (*make planks* and *make sticks*, each associated with its own policy (Π_1 and Π_2 respectively). These policies share an initial high-level action b_1: both require the agent to *get wood* before taking it to an appropriate crafting station. By enforcing that the agent initially follows the same subpolicy π_1 in both tasks, we can learn a reusable representation of their shared structure.

This approach builds on a significant body of research in reinforcement learning that focuses on *hierarchical* representations of behavior. In these approaches, a high-level controller learns a policy over high-level actions—known variously as options (Sutton et al., 1999), skills

(Konidaris & Barto, 2007), or primitives (Hauser et al., 2008)—which are themselves implemented as policies over low-level actions in the environment. While one line of research (e.g. Daniel et al. (2012)) investigates learning hierarchical policies without any supervision, such hierarchies are empirically difficult to learn directly from unconstrained interaction (Hengst, 2002). The bulk of existing work instead relies on additional information (in the form of intermediate rewards, subtask completion signals, or intrinsic motivations) that guide the learner toward useful high-level actions. While effective, these approaches depend on state representations simple or structured enough that suitable reward signals can be effectively engineered by hand.

Here we focus on multitask learning of hierarchical policies from a weaker form of supervision: at training time, each task (τ_1 and τ_2 in Figure 1) is annotated with a sketch (K_1 and K_2) consisting of a sequence of high-level action symbols (b_1, b_2 and b_3)—with no information about how these actions should be implemented. Our approach associates each such high-level action with its own low-level subpolicy, and jointly optimizes over concatenated task-specific policies by tying parameters across shared subpolicies. Our thesis is that even the minimal information about high-level policy structure contained in a sketch provides enough of a learning signal to induce general, reusable subpolicies. Crucially, sketches are totally ungrounded in the representation of the world—they require no intervention in a simulator or environment model.

The present work may be viewed as an extension of recent approaches for learning compositional deep architectures from structured program descriptors (Andreas et al., 2016; Reed & de Freitas, 2015). Here we focus on learning in interactive environments with reinforcement training signals. This extension presents a variety of technical challenges. Concretely, our contributions are:

- A general paradigm for multitask, hierarchical, deep reinforcement learning guided by abstract sketches of task-specific policies.
- A concrete agent architecture for learning in this paradigm, featuring a modular model structure and multitask actor–critic training objective.

We evaluate our approach on two families of tasks: a maze navigation game (Figure 3a), in which the agent must navigate through a sequence of locked doors to reach a target room; and a 2-D Minecraft-inspired crafting game (Figure 3b), in which the agent must acquire particular resources by finding raw ingredients, combining them together in the proper order, and in some cases building intermediate tools that enable the agent to alter the environment itself. In both games, the agent receives a reward only after the final goal is accomplished. For the most challenging tasks, involving sequences of four or five high-level actions, a task-specific agent initially following a random policy essentially never discovers the reward signal.

We evaluate a modular agent architecture trained with guidance from policy sketches under several different data conditions: (1) when learning the full collection of tasks jointly via reinforcement, (2) in a zero-shot setting where a policy sketch is available for the held-out task, and (3) in a adaptation setting, where sketches are hidden and the agent must learn a policy over high-level actions. In all cases, our approach substantially outperforms standard policy optimization baselines.

2 Related Work

The agent representation we describe in this paper belongs to the broader family of hierarchical reinforcement learners described in the literature. As detailed in Section 3, our subpolicies may be viewed as a relaxation of the *options* framework first described by Sutton et al. (1999). A large body of work describes techniques for learning options and related abstract actions, in both single- and multitask settings. For learning the implementation of options, most techniques rely on intermediate supervisory signals, e.g. to encourage exploration (Kearns & Singh, 2002) or completion of predefined subtasks (Kulkarni et al., 2016). An alternative family of approaches employs either post-hoc analysis of already-learned policies to extract reusable sub-components (Stolle & Precup, 2002; Konidaris et al., 2011). Techniques for learning options with less guidance than the present work include Bacon & Precup (2015) and Vezhnevets et al. (2016), and other general hierarchical policy learners include Daniel et al. (2012), Bakker & Schmidhuber (2004) and Menache et al. (2002).

Once a library of high-level actions exists, agents are faced with the problem of learning high-level (typically semi-Markov) policies that invoke appropriate high-level actions in sequence (Precup,

2000). The learning problem we describe in this paper is in some sense the direct dual to the problem of learning these high-level policies. There, the agent begins with an inventory of complex primitives and must learn to model their behavior and select among them; here we begin knowing the names of appropriate high-level actions but nothing about how they are implemented, and must infer implementations (but not, initially, high-level plans) from context. We expect that our approach could be coupled with a generic learner of options policies to provide a general mechanism for hierarchical RL; we leave this for future work.

Our approach is also inspired by a number of recent efforts toward compositional reasoning and interaction with structured deep models. Such models have been previously used for tasks involving question answering (Iyyer et al., 2014; Andreas et al., 2016) and relational reasoning (Socher et al., 2012), and more recently for multi-task, multi-robot transfer problems (Devin et al., 2016). In this work—as in existing approaches employing dynamically assembled modular networks—task-specific training signals are propagated through a collection of composed discrete structures with tied weights. Here the composed structures specify time-varying policies rather than feedforward computations, and their parameters must be learned via interaction rather than direct supervision. Another closely related family of models includes neural programmers (Neelakantan et al., 2015) and programmer–interpreters (Reed & de Freitas, 2015), which generate discrete computational structures but require supervision in the form of output actions or full execution traces.

A closely related line of work is the Hierarchical Abstract Machines (HAM) framework introduced by Parr & Russell (1998). Like our approach, HAMs begin with a representation of a high-level policy as an automaton (or a more general computer program; Andre & Russell, 2001) and use reinforcement learning to fill in low-level details. Variations on this architecture have considered a number of control constructs beyond the scope of the current paper (e.g. concurrency and recursion; Marthi et al., 2004). However, because these approaches attempt to learn a single representation of the Q function for all subtasks and contexts, they require extremely strong formal assumptions about the form of the reward function and state representation (Andre & Russell, 2002) that the present work avoids by decoupling the policy representation from the value function.

Our approach also bears some resemblance to the instruction following literature in natural language processing. Existing work on instruction following falls into two broad categories: approaches that require a highly structured (typically logical) action and world representations (Chen & Mooney, 2011; Artzi & Zettlemoyer, 2013; Andreas & Klein, 2015; Tellex et al., 2011), and approaches that require detailed supervision of action sequences or dense reward signals essentially equivalent to full action traces (Branavan et al., 2009; Vogel & Jurafsky, 2010; Mei et al., 2016). By contrast, the framework we describe here involves no formal or logical language for describing plans, and no supervised action sequences. Additionally, the modular model described in this paper natrually supports adaptation to tasks where no sketches are available, while all existing instruction following models learn a joint policy over instructions and actions, and are unable to function in the absence of instructions.

3 Learning Modular Policies

We consider a multitask reinforcement learning problem arising from a family of infinite-horizon discounted Markov decision processes in a shared environment. This environment is specified by a tuple $(\mathcal{S}, \mathcal{A}, P, \gamma)$, with $\mathcal{S}$ a set of states, $\mathcal{A}$ a set of low-level actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ a transition probability distribution, and γ a discount factor. Each task $\tau \in \mathcal{T}$ is then specified by a pair (R_τ, ρ_τ), with $R_\tau : \mathcal{S} \to \mathbb{R}$ a task-specific reward function and $\rho_\tau : \mathcal{S} \to \mathbb{R}$ an initial distribution over states. For a fixed sequence $\{(s_i, a_i)\}$ of states and actions obtained from a rollout of a given policy, we will denote the empirical return starting in state s_i as $q_i := \sum_{j=i}^{\infty} \gamma^j R(s_j)$. In addition to the components of a standard multitask RL problem, we assume that tasks are annotated with *sketches* K_τ, each consisting of a sequence $(b_{\tau 1}, b_{\tau 2}, \ldots)$ of high-level symbolic labels drawn from a fixed vocabulary $\mathcal{B}$. Our model associates each of these symbols with a randomly initialized modular subpolicy. By sharing each subpolicy across all tasks annotated with the corresponding symbol, our approach naturally learns the shared abstraction for the corresponding subtask, without requiring any information about the grounding of that task to be explicitly specified by annotation.

3.1 MODEL

We exploit the structural information provided by sketches by constructing for each symbol b a corresponding *subpolicy* π_b. At each timestep, a subpolicy may select either a low-level action $a \in \mathcal{A}$ or a special STOP action. We denote the augmented state space $\mathcal{A}^+ := \mathcal{A} \cup \{\text{STOP}\}$. While this framework is agnostic to the implementation of subpolicies, we are especially interested in the case where subpolicies are specified by deep networks. As shown in Figure 2, the experiments in this paper represent each π_b as a neural network whose input is a representation of the current state, and whose output is a distribution over $\mathcal{A}^+$. While all action spaces in our experiments are discrete, it is straightforward to instead allow this last layer to parameterize a mixed distribution over an underlying continuous action space and the STOP action. These subpolicies may be viewed as options of the kind described by Sutton et al. (1999), with the key distinction that they have no initiation semantics, but are instead invokable everywhere, and have no explicit representation as a function from an initial state to a distribution over final states (instead implicitly using the STOP action to terminate).

Given a sketch, a task-specific policy Π_τ is formed by concatenating its associated subpolicies in sequence. In particular, the high-level policy maintains a subpolicy index i (initially 0), and executes actions from π_{b_i} until the STOP symbol is emitted, at which point control is passed to $\pi_{b_{i+1}}$. We may thus think of Π_τ as inducing a Markov chain over the state space $\mathcal{S} \times \mathcal{B}$, with transitions given by:

$$\begin{aligned} (s, b_i) &\to (s', b_i) && \text{with probability} && \textstyle\sum_{a \in \mathcal{A}} \pi_{b_i}(a|s) \cdot P(s'|s, a) \\ &\to (s, b_{i+1}) && \text{with probability} && \pi_{b_i}(\text{STOP}|s) \end{aligned}$$

Note that Π_τ is semi-Markov with respect to projection of the augmented state space $\mathcal{S} \times \mathcal{B}$ onto the underlying state space $\mathcal{S}$. We denote the complete family of task-specific policies $\mathbf{\Pi} := \bigcup_\tau \{\Pi_\tau\}$, and let each π_b be an arbitrary function of the current environment state parameterized by some weight vector θ_b. The learning problem is to optimize over all θ_b to maximize the sum of expected discounted rewards $J(\mathbf{\Pi}) := \sum_\tau J(\Pi_\tau) := \sum_\tau \mathbb{E}_{s_i \sim \Pi_\tau}\left[\sum_i \gamma^i R_\tau(s_i)\right]$ across all tasks $\tau \in \mathcal{T}$.

3.2 POLICY OPTIMIZATION

Here that optimization is accomplished via a simple decoupled actor–critic method. In a standard policy gradient approach, with a single policy π with parameters θ, we compute gradient steps of the form (Williams, 1992):

$$\nabla_\theta J(\pi) = \sum_i \big(\nabla_\theta \log \pi(a_i|s_i)\big)\big(q_i - c(s)\big), \qquad (1)$$

where the baseline or "critic" c can be chosen independently of the future without introducing bias into the gradient. Recalling our previous definition of q_i as the empirical return starting from s_i, this form of the gradient corresponds to a generalized advantage estimator (Schulman et al., 2015) with $\lambda = 1$. Here c achieves close to the optimal variance (Greensmith et al., 2004) when it is set exactly equal to the state-value function $V_\pi(s_i) = \mathbb{E}_\pi q_i$ for the target policy π starting in state s_i.

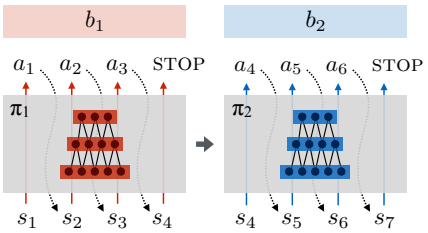


Figure 2: Model overview. Each subpolicy π is uniquely associated with a symbol b implemented as a neural network that maps from a state s_i to distributions over $\mathcal{A}^+$, and chooses an action a_i by sampling from this distribution. Whenever the STOP action is sampled, control advances to the next subpolicy in the sketch.

The situation becomes slightly more complicated when generalizing to modular policies built by sequencing subpolicies. In this case, we will have one subpolicy per symbol but one critic per *task*. This is because subpolicies π_b might participate in a number of composed policies Π_τ, each associated with its own reward function R_τ. Thus individual subpolicies are not uniquely identified with value functions, and the aforementioned subpolicy-specific state-value estimator is no longer well-defined. We extend the actor–critic method to incorporate the decoupling of policies from value functions by allowing the critic to vary per-sample (that is, per-task-and-timestep) depending on the reward function with which the sample is associated. Noting that $\nabla_{\theta_b} J(\mathbf{\Pi}) = \sum_{t:b \in K_\tau} \nabla_{\theta_b} J(\Pi_\tau)$, i.e. the expected reward across all tasks in which π_b participates, we have:

$$\nabla_\theta J(\mathbf{\Pi}) = \sum_\tau \nabla_\theta J(\Pi_\tau) = \sum_\tau \sum_i \big(\nabla_{\theta_b} \log \pi_b(a_{\tau i}|s_{\tau i})\big)\big(q_i - c_\tau(s_{\tau i})\big), \qquad (2)$$

where each state-action pair $(s_{\tau i}, a_{\tau i})$ was selected by the subpolicy π_b in the context of the task τ.

Now minimization of the gradient variance requires that each c_τ actually depend on the task identity. (This follows immediately by applying the corresponding argument in Greensmith et al. (2004) individually to each term in the sum over τ in Equation 2.) Because the value function is itself unknown, an approximation must be estimated from data. Here we allow these c_τ to be implemented with an arbitrary function approximator parameterized by a vector η_τ. This is trained to minimize a squared error criterion, with gradients given by

$$\nabla_{\eta_\tau}\left[\frac{1}{2}\sum_i (q_i - c_\tau(s_i))^2\right] = \sum_i \big(\nabla_{\eta_\tau} c_\tau(s_i)\big)\big(q_i - c_\tau(s_i)\big). \tag{3}$$

Alternative forms of the advantage estimator (e.g. the TD residual $R_\tau(s_i) + V_\tau(s_{i+1}) - \gamma V_\tau(s_i)$ or any other member of the GAE family) can be easily substituted by simply maintaining one such estimator per task. Experiments (Section 4.3) show that conditioning on both the state and the task identity results in noticeable performance improvements, suggesting that the variance reduction provided by this objective is important for efficient joint learning of modular policies.

Algorithm 1 DO-STEP($\mathbf{\Pi}$, curriculum)

$\mathcal{D} \leftarrow \emptyset$
while $|\mathcal{D}| < D$ **do**
  $\tau \sim$ curriculum$(\cdot)$ ▷ sample task τ from curriculum (Section 3.3)
  $d = \{(s_i, a_i, b_i = K_{\tau,i}, q_i, \tau), \ldots\} \sim \Pi_\tau$ ▷ do rollout
  $\mathcal{D} \leftarrow \mathcal{D} \cup d$

for $b \in \mathcal{B}, \tau \in \mathcal{T}$ **do**
  $d = \{(s_i, a_i, b', q_i, \tau') \in \mathcal{D} : b' = b, \tau' = \tau\}$
  $\theta_b \leftarrow \theta_b - \frac{\alpha}{D}\sum_d \big(\nabla \log \pi_b(a_i|s_i)\big)\big(q_i - c_\tau(s_i)\big)$ ▷ update policy
  $\eta_\tau \leftarrow \eta_\tau - \frac{\beta}{D}\sum_d \big(\nabla c_\tau(s_i)\big)\big(q_i - c_\tau(s_i)\big)$ ▷ update critic

The complete procedure for computing a *single* gradient step is given in Algorithm 1. (The outer training loop over these steps, which is driven by a curriculum learning procedure, is described in the following section and specified in Algorithm 2.) This is an on-policy algorithm. In each step, the agent samples tasks from a task distribution provided by a curriculum (described in the following subsection). The current family of policies $\mathbf{\Pi}$ is used to perform rollouts in each sampled task, accumulating the resulting tuples of (states, low-level actions, high-level symbols, rewards, and task identities) into a dataset $\mathcal{D}$. Once $\mathcal{D}$ reaches a maximum size D, it is used to compute gradients w.r.t. both policy and critic parameters, and the parameter vectors are updated accordingly. The step sizes α and β in Algorithm 1 can be chosen adaptively using any first-order method.

3.3 CURRICULUM LEARNING

For complex tasks, like the one depicted in Figure 3b, it is difficult for the agent to discover any states with positive reward until many subpolicy behaviors have already been learned. It is thus a better use of the learner's time to focus on "easy" tasks, where many rollouts will result in high reward from which appropriate subpolicy behavior can be inferred. But there is a fundamental tradeoff involved here: if the learner spends too much time on easy tasks before being made aware of the existence of harder ones, it may overfit and learn subpolicies that no longer generalize or exhibit the desired structural properties.

To avoid both of these problems, we use a curriculum learning scheme (Bengio et al., 2009) that allows the model to smoothly scale up from easy tasks to more difficult ones while avoiding overfitting. Initially the model is presented with tasks associated with short sketches. Once average reward on all these tasks reaches a certain threshold, the length limit is incremented. We assume that rewards across tasks are normalized with maximum achievable reward $0 < q_i < 1$. Let $\hat{\mathbb{E}} r_\tau$ denote the empirical estimate of the expected reward for the current policy on task t. Then at each timestep, tasks are sampled in proportion to $1 - \hat{\mathbb{E}} r_\tau$, which by assumption must be positive. Experiments show that both components of this curriculum learning scheme improve the rate at which the model converges to a good policy (Section 4.3).

The complete curriculum-based training procedure is specified in Algorithm 2. Initially, the maximum sketch length ℓ_{max} is set to one, and the curriculum initialized to sample length-1 tasks uniformly. (Neither of the environments we consider in this paper feature any length-1 tasks; in this case, observe that Algorithm 2 will simply advance to length-2 tasks without any parameter updates.) For each setting of ℓ_{max}, the algorithm uses the current collection of task policies $\mathbf{\Pi}$ to compute and apply the gradient step described in Algorithm 1. The rollouts obtained from the call to DO-STEP can also be used to compute reward estimates $\hat{\mathbb{E}} r_\tau$; these estimates determine a new task distribution for the curriculum. The inner loop is repeated until the reward threshold r_{min} is exceeded, at which point ℓ_{max} is incremented and the process repeated over a (now-expanded) collection of tasks.

4 EXPERIMENTS

As described in the introduction, we evaluate the performance of our approach in two environments: a maze navigation game and a crafting game. Both games involve nontrivial low-level control: agents must learn to avoid obstacles and interact with various kinds of objects. But the environments also feature hierarchical structure: rewards are accessible only after the agent has completed two to five high-level actions in the appropriate sequence.

In all our experiments, we implement each subpolicy as a multilayer perceptron with ReLU nonlinearities and a hidden layer with 128 hidden units, and each critic as a linear function of the current state. Each subpolicy network receives as input a set of features describing the current state of the environment, and outputs a distribution over actions. The agent acts at every timestep by sampling from this distribution. The gradient steps given in lines 8 and 9 of Algorithm 1 are implemented using RMSPROP (Tieleman, 2012) with a step size of 0.001 and gradient clipping to a unit norm. We take the batch size parameter D in Algorithm 1 to be 2000, and set $\gamma = 0.9$ in both environments. For curriculum learning, the improvement threshold r_{good} is set to 0.8.

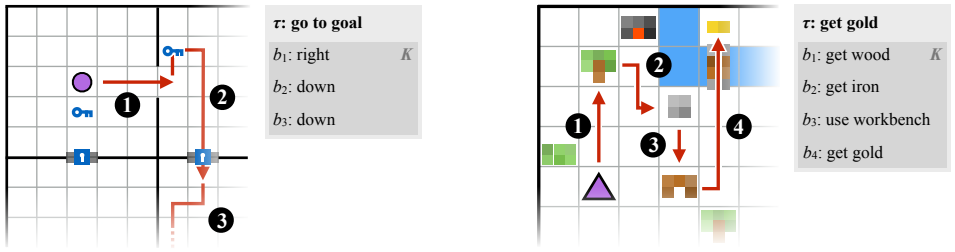
4.1 ENVIRONMENTS

The maze environment (Figure 3a) corresponds closely to the the "light world" described by Konidaris & Barto (2007). The agent is placed in a discrete world consisting of a series of rooms, some of which are connected by doors. Some doors require that the agent first pick up a key to open them. For our experiments, each task corresponds to a goal room (always at the same position relative to the agent's starting position) that the agent must reach by navigating through a sequence of intermediate rooms. The agent has one sensor on each side of its body, which reports the distance to keys, closed doors, and open doors in the corresponding direction. Sketches specify a particular sequence of directions for the agent to traverse between rooms to reach the goal. Mazes are sampled with random sizes and random decisions about whether to connect rooms with open doors, locked doors, or no doors. The sketch always corresponds to a viable traversal from the start to the goal position, but other (possibly shorter) traversals may also exist.

The crafting environment (Figure 3b) is inspired by the popular game Minecraft, but is implemented in a 2-D grid world. The agent may interact with some objects in the world by facing them

Algorithm 2 TRAIN-POLICIES()

1: $\mathbf{\Pi} = \text{INIT}()$ ▷ initialize subpolicies randomly
2: $\ell_{\text{max}} \leftarrow 1$
3: **loop**
4: $\quad r_{\text{min}} \leftarrow \infty$
5: $\quad \text{curriculum}(\cdot) = \text{Unif}(\mathcal{T}')$ ▷ initialize ℓ_{max}-step curriculum uniformly
6: $\quad \mathcal{T}' = \{\tau \in \mathcal{T} : |K_\tau| \leq \ell_{\text{max}}\}$
7: $\quad$ **while** $r_{\text{min}} < r_{\text{good}}$ **do**
8: $\quad\quad \text{DO-STEP}(\mathbf{\Pi}, \text{curriculum})$ ▷ update parameters (Algorithm 1)
9: $\quad\quad Z = \sum_{t \in \mathcal{T}'} [1 - \hat{\mathbb{E}} r_\tau]$
10: $\quad\quad \text{curriculum}(t) = \mathbb{1}[\tau \in \mathcal{T}'](1 - \hat{\mathbb{E}} r_\tau)/Z \quad \forall \tau \in \mathcal{T}$
11: $\quad\quad r_{\text{min}} \leftarrow \min_\tau \hat{\mathbb{E}} r_\tau$
12: $\quad \ell_{\text{max}} \leftarrow \ell_{\text{max}} + 1$



(a) (b)

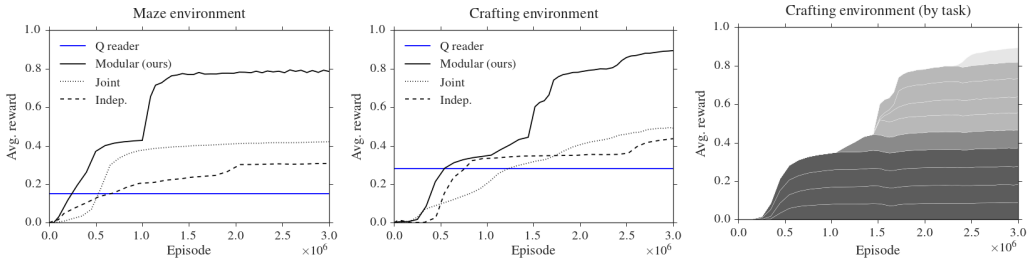
Figure 3: Example tasks from the environments used in this paper. (a) In the maze environment, the agent must reach a goal position by traversing right (1), down (2) and down again (3) through a sequence of rooms, some of which may have locked doors. (b) In the crafting environment, an agent seeking to pick up the gold nugget in the top corner must first collect wood (1) and iron (2), use a workbench to turn them into a bridge (3), and use the bridge to cross the water (4).

and executing a special INTERACT action. Interacting with raw materials initially scattered around the environment causes them to be added to an inventory. Interacting with different crafting stations causes objects in the agent's inventory to be combined or transformed into other objects. Each task in this game corresponds to some crafted object the agent must produce; the most complicated goals require the agent to also craft intermediate ingredients, and in some cases build tools (like a pickaxe and a bridge) to reach ingredients located in initially inaccessible regions of the environment.

A complete listing of tasks and sketches is given in Appendix A.

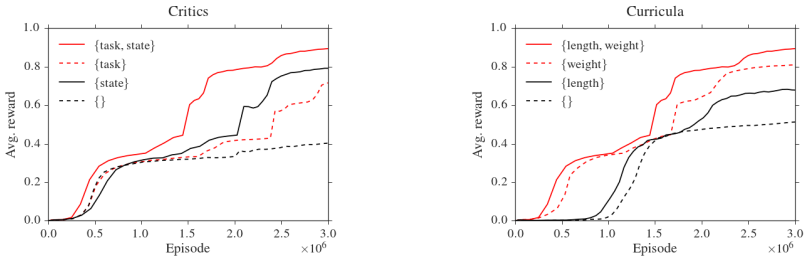
4.2 MULTITASK LEARNING

The primary experimental question in this paper is whether the extra structure provided by policy sketches alone is enough to enable fast learning of coupled policies across tasks. To evaluate this, we compare our **modular** approach to two policy gradient baselines—one that learns an **independent** policy for each task and one that learns a **joint** policy across all tasks—as well as a critic-only **Q reader** baseline. For the independent model, task-specific policies are represented by networks with the same structure as the modular subpolicies. The joint model conditions both on these environment features, as well as a feature vector encoding the complete sketch. The Q reader forms the same joint state and action space described in Section 3.1, and learns a single feedforward network to map from both environment states and representations of action symbols onto Q values. This baseline can be viewed either as a chain-structured hierarchical abstract machine with a learned state abstractor (Andre & Russell, 2002), or as a standard instruction following baseline from the natural language processing literature (Vogel & Jurafsky, 2010).



(a) (b) (c)

Figure 4: Comparing modular learning from sketches with standard RL baselines. **Modular** is the approach described in this paper, while **Independent** learns a separate policy for each task, **Joint** learns a shared policy that conditions on the task identity, and **Q reader** learns a single network to map from states and action symbols to Q values. Performance for the best iteration of the (off-policy) Q reader is plotted. (a) Performance of the three models in the maze environment. (b) Performance in the crafting environment. (c) Individual task performance for the modular model in the crafting domain. Colors correspond to task length. It can be seen that the sharp steps in the learning curve correspond to increases of ℓ_{max} in the curriculum. The modular approach is eventually able to achieve high reward on all tasks, while the baseline models perform considerably worse on average.



(a) (b)

Figure 5: Ablation experiments. (a) The critic: lines labeled "task" include a baseline that varies with the task identity, while lines labeled "state" include a baseline that varies with the state identity. Estimating a baseline that depends on both the representation of the current state and the identity of the current task is better than either alone or a constant baseline. (b) The curriculum: lines labeled "length" use a curriculum with iteratively increasing lengths, while lines labeled "weight" sample tasks in inverse proportion to their current reward. Adjusting the sampling distribution based on both task length and performance return improves convergence.

Learning curves for baselines and the modular model are shown in Figure 4. It can be seen that in both the maze domain and the crafting domain, our approach substantially outperforms the baselines: it induces policies with substantially higher average reward and converges more quickly than the policy gradient baselines. It can further be seen in Figure 4c that after policies have been learned on simple tasks, the model is able to rapidly adapt to more complex ones, even when the longer tasks involve high-level actions not required for any of the short tasks (Appendix A).

Having demonstrated the overall effectiveness of our approach, our remaining experiments explore (1) the importance of various components of the training procedure, and (2) the learned models' ability to generalize or adapt to held-out tasks. For compactness, we restrict our consideration on the crafting domain, which features a larger and more diverse range of tasks and high-level actions.

4.3 Ablations

In addition to the overall modular parameter-tying structure induced by our sketches, the key components of our training procedure are the decoupled critic and the curriculum. Our next experiments investigate the extent to which these are necessary for good performance.

To evaluate the the critic, we consider three ablations: (1) removing the dependence of the model on the environment state, in which case the baseline is a single scalar per task; (2) removing the dependence of the model on the task, in which case the baseline is a conventional generalized advantage estimator; and (3) removing both, in which case the baseline is a single scalar, as in a vanilla policy gradient approach. Results are shown in Figure 5a. Introducing both state and task dependence into the baseline leads to faster convergence of the model: the approach with a constant baseline achieves less than half the overall performance of the full critic after 3 million episodes. Introducing task and state dependence independently improve this performance; combining them gives the best result.

We also investigate two aspects of our curriculum learning scheme: starting with short examples and moving to long ones, and sampling tasks in inverse proportion to their accumulated reward. Experiments are shown in Figure 5b. We again see that both components are essential for good performance. Sampling uniformly across all tasks of the target length results in slow convergence.

4.4 Zero-shot and adaptation learning

In our final experiments, we consider the model's ability to generalize to new tasks unseen at training time. We consider two evaluation conditions: a **zero-shot** setting, in which the model is provided a sketch for the new task and must immediately achieve good performance, and a **adaptation** setting, in which no sketch is provided and the model must learn the form of a suitable sketch by interacting with the new task.

Model	MT	0-S	Ad.
Independent	.44	–	<.1
Joint	.49	<.1	–
Modular	**.89**	**.77**	**.76**

Table 1: Model performance under various evaluation conditions. **MT** is the multitask training condition described in Section 4.2, while **0-S** and **Ad.** are respectively the zero-shot and adaptation experiments described in Section 4.4.

We hold out two length-four tasks from the full inventory used in Section 4.2, and train on the remaining tasks. For zero-shot experiments, we simply form the concatenated policy described by the sketches of the held-out tasks, and repeatedly execute this policy (without learning) in order to obtain an estimate of its effectiveness. For adaptation experiments, we consider ordinary reinforcement learning over $\mathcal{B}$ rather than $\mathcal{A}$, implementing the high-level learner with the same agent architecture as described in Section 3.1. Note that the Independent baseline cannot be applied to the zero-shot evaluation, while the joint baseline cannot be applied to the adaptation baseline (because it depends on pre-specified sketch features). Results are shown in Table 1. The held-out tasks are sufficiently challenging that the baselines are unable to obtain more than negligible reward, while the modular model does comparatively well.

5 Conclusions

We have described an approach for multitask learning of neural network policies guided by symbolic policy sketches. By associating each symbol appearing in a sketch with a modular neural subpolicy, we have shown that it is possible to build agents that share behavior across tasks in order to achieve success in tasks with sparse and delayed rewards. This process induces an inventory of reusable and interpretable subpolicies which can be employed for zero-shot generalization when further sketches are available, and hierarchical reinforcement learning when they are not. Our work suggests that these sketches, which are easy to produce and require no grounding in the environment, provide an effective scaffold for learning hierarchical policies from minimal supervision. We have released our code at `http://github.com/jacobandreas/psketch`.

Acknowledgments

JA is supported by a Facebook Graduate Fellowship and a Huawei / Berkeley AI fellowship.

References

David Andre and Stuart Russell. Programmable reinforcement learning agents. In *Advances in Neural Information Processing Systems*, 2001.

David Andre and Stuart Russell. State abstraction for programmable reinforcement learning agents. In *Proceedings of the Meeting of the Association for the Advancement of Artificial Intelligence*, 2002.

Jacob Andreas and Dan Klein. Alignment-based compositional semantics for instruction following. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2015.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to compose neural networks for question answering. In *Proceedings of the Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, 2016.

Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1(1):49–62, 2013.

Pierre-Luc Bacon and Doina Precup. The option-critic architecture. In *NIPS Deep Reinforcement Learning Workshop*, 2015.

Bram Bakker and Jürgen Schmidhuber. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In *Proc. of the 8-th Conf. on Intelligent Autonomous Systems*, pp. 438–445, 2004.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. pp. 41–48. ACM, 2009.

S.R.K. Branavan, Harr Chen, Luke S. Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 82–90. Association for Computational Linguistics, 2009.

David L. Chen and Raymond J. Mooney. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the Meeting of the Association for the Advancement of Artificial Intelligence*, volume 2, pp. 1–2, 2011.

Christian Daniel, Gerhard Neumann, and Jan Peters. Hierarchical relative entropy policy search. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pp. 273–281, 2012.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. *arXiv preprint arXiv:1609.07088*, 2016.

Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530, 2004.

Kris Hauser, Timothy Bretl, Kensuke Harada, and Jean-Claude Latombe. Using motion primitives in probabilistic sample-based planning for humanoid robots. In *Algorithmic foundation of robotics*, pp. 507–522. Springer, 2008.

Bernhard Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *ICML*, volume 2, pp. 243–250, 2002.

Mohit Iyyer, Jordan Boyd-Graber, Leonardo Claudino, Richard Socher, and Hal Daumé III. A neural network for factoid question answering over paragraphs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002.

George Konidaris and Andrew G Barto. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, volume 7, pp. 895–900, 2007.

George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, pp. 0278364911428653, 2011.

Tejas D Kulkarni, Karthik R Narasimhan, Ardavan Saeedi, and Joshua B Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *arXiv preprint arXiv:1604.06057*, 2016.

Bhaskara Marthi, David Lantham, Carlos Guestrin, and Stuart Russell. Concurrent hierarchical reinforcement learning. In *Proceedings of the Meeting of the Association for the Advancement of Artificial Intelligence*, 2004.

Hongyuan Mei, Mohit Bansal, and Matthew Walter. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Proceedings of the Meeting of the Association for the Advancement of Artificial Intelligence*, 2016.

Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cutdynamic discovery of sub-goals in reinforcement learning. In *European Conference on Machine Learning*, pp. 295–306. Springer, 2002.

Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. *arXiv preprint arXiv:1511.04834*, 2015.

Ron Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, 1998.

Doina Precup. *Temporal abstraction in reinforcement learning*. PhD thesis, 2000.

Scott Reed and Nando de Freitas. Neural programmer-interpreters. *Proceedings of the International Conference on Learning Representations*, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

Richard Socher, Brody Huval, Christopher Manning, and Andrew Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 1201–1211, Jeju, Korea, 2012.

Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on Abstraction, Reformulation, and Approximation*, pp. 212–223. Springer, 2002.

Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *In Proceedings of the National Conference on Artificial Intelligence*, 2011.

Tijmen Tieleman. RMSProp (unpublished), 2012.

Alexander Vezhnevets, Volodymyr Mnih, John Agapiou, Simon Osindero, Alex Graves, Oriol Vinyals, and Koray Kavukcuoglu. Strategic attentive writer for learning macro-actions. *arXiv preprint arXiv:1606.04695*, 2016.

Adam Vogel and Dan Jurafsky. Learning to follow navigational directions. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 806–814. Association for Computational Linguistics, 2010.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

A TASKS AND SKETCHES

The complete list of tasks, sketches, and symbols is given below. Tasks marked with an asterisk* are held out for the generalization experiments described in Section 4.4, but included in the multitask training experiments in Sections 4.2 and 4.3.

Goal	Sketch				
Maze environment					
goal1	left	left			
goal2	left	down			
goal3	right	down			
goal4	up	left			
goal5	up	right			
goal6	up	right	up		
goal7	down	right	up		
goal8	left	left	down		
goal9	right	down	down		
goal10	left	up	right		
Crafting environment					
make plank	get wood	use toolshed			
make stick	get wood	use workbench			
make cloth	get grass	use factory			
make rope	get grass	use toolshed			
make bridge	get iron	get wood	use factory		
make bed*	get wood	use toolshed	get grass	use workbench	
make axe*	get wood	use workbench	get iron	use toolshed	
make shears	get wood	use workbench	get iron	use workbench	
get gold	get iron	get wood	use factory	use bridge	
get gem	get wood	use workbench	get iron	use toolshed	use axe