Modular Multitask Reinforcement Learning with Policy Sketches

Jacob Andreas [1] **Dan Klein** [1] **Sergey Levine** [1]

Abstract

We describe a framework for multitask deep reinforcement learning guided by *policy sketches*. Sketches annotate tasks with sequences of named subtasks, providing information about high-level structural relationships among tasks but not how to implement them—specifically not providing the detailed guidance used by much previous work on learning policy abstractions for RL (e.g. intermediate rewards, subtask completion signals, or intrinsic motivations). To learn from sketches, we present a model that associates every subtask with a modular subpolicy, and jointly maximizes reward over full task-specific policies by tying parameters across shared subpolicies. Optimization is accomplished via a decoupled actor–critic training objective that facilitates learning common behaviors from multiple dissimilar reward functions. We evaluate the effectiveness of our approach in three environments featuring both discrete and continuous control, and with sparse rewards that can be obtained only after completing a number of high-level subgoals. Experiments show that using our approach to learn policies guided by sketches gives better performance than existing techniques for learning task-specific or shared policies, while naturally inducing a library of interpretable primitive behaviors that can be recombined to rapidly adapt to new tasks.

1. Introduction

This paper describes a framework for learning composable deep subpolicies in a multitask setting, guided only by abstract sketches of high-level behavior. General reinforcement learning algorithms allow agents to solve tasks in complex environments. But tasks featuring extremely

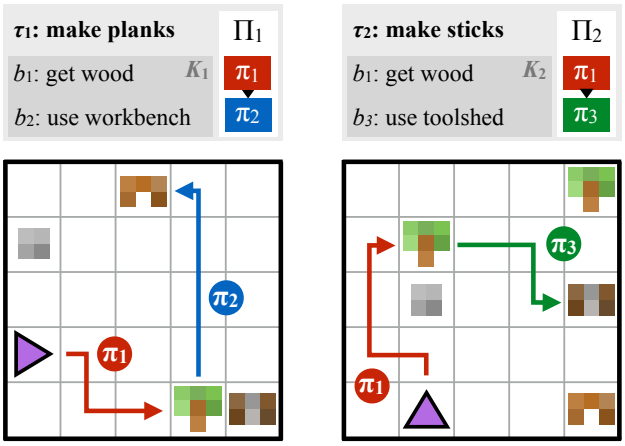


Figure 1: Learning from policy sketches. The figure shows simplified versions of two tasks (*make planks* and *make sticks*, each associated with its own policy (Π_1 and Π_2 respectively). These policies share an initial high-level action b_1: both require the agent to *get wood* before taking it to an appropriate crafting station. Even without prior information about how the associated behavior π_1 should be implemented, knowing that the agent should initially follow the same subpolicy in both tasks is enough to learn a reusable representation of their shared structure.

delayed rewards or other long-term structure are often difficult to solve with flat, monolithic policies, and a long line of prior work has studied methods for learning hierarchical policy representations (Sutton et al., 1999; Dietterich, 2000; Konidaris & Barto, 2007; Hauser et al., 2008). While unsupervised discovery of these hierarchies is possible (Daniel et al., 2012; Bacon & Precup, 2015), practical approaches often require detailed supervision in the form of explicitly specified high-level actions, subgoals, or behavioral primitives (Precup, 2000). These depend on state representations simple or structured enough that suitable reward signals can be effectively engineered by hand.

But is such fine-grained supervision actually necessary to achieve the full benefits of hierarchy? Specifically, is it necessary to explicitly ground high-level actions into the representation of the environment? Or is it sufficient to simply inform the learner about the abstract *structure* of policies, without ever specifying how high-level behaviors should make use of primitive percepts or actions?

To answer these questions, we explore a multitask reinforcement learning setting where the learner is pre-

[1]University of California, Berkeley. Correspondence to: Jacob Andreas <jda@cs.berkeley.edu>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

sented with *policy sketches*. Policy sketches are short, ungrounded, symbolic representations of a task that describe its component parts, as illustrated in Figure 1. While symbols might be shared across tasks (*get wood* appears in sketches for both the *make planks* and *make sticks* tasks), the learner is told nothing about what these symbols *mean*, in terms of either observations or intermediate rewards.

We present an agent architecture that learns from policy sketches by associating each high-level action with a parameterization of a low-level subpolicy, and jointly optimizes over concatenated task-specific policies by tying parameters across shared subpolicies. We find that this architecture can use the high-level guidance provided by sketches, without any grounding or concrete definition, to dramatically accelerate learning of complex multi-stage behaviors. Our experiments indicate that many of the benefits to learning that come from highly detailed low-level supervision (e.g. from subgoal rewards) can also be obtained from fairly coarse high-level supervision (i.e. from policy sketches). Crucially, sketches are much easier to produce: they require no modifications to the environment dynamics or reward function, and can be easily provided by non-experts. This makes it possible to extend the benefits of hierarchical RL to challenging environments where it may not be possible to specify by hand the details of relevant subtasks. We show that our approach substantially outperforms purely unsupervised methods that do not provide the learner with any task-specific guidance about how hierarchies should be deployed, and further that the specific use of sketches to parameterize modular subpolicies makes better use of sketches than conditioning on them directly.

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, 2016). Here we focus on learning in interactive environments. This extension presents a variety of technical challenges, requiring analogues of these methods that can be trained from sparse, non-differentiable reward signals without demonstrations of desired system behavior.

Our contributions are:

- A general paradigm for multitask, hierarchical, deep reinforcement learning guided by abstract sketches of task-specific policies.
- A concrete recipe for learning from these sketches, built on a general family of modular deep policy representations and a multitask actor–critic training objective.

The modular structure of our approach, which associates every high-level action symbol with a discrete subpolicy, naturally induces a library of interpretable policy fragments that are easily recombined. This makes it possible to evaluate our approach under a variety of different data conditions: (1) learning the full collection of tasks jointly via reinforcement, (2) in a zero-shot setting where a policy sketch is available for a held-out task, and (3) in a adaptation setting, where sketches are hidden and the agent must learn to adapt a pretrained policy to reuse high-level actions in a new task. In all cases, our approach substantially outperforms previous approaches based on explicit decomposition of the Q function along subtasks (Parr & Russell, 1998; Vogel & Jurafsky, 2010), unsupervised option discovery (Bacon & Precup, 2015), and several standard policy gradient baselines.

We consider three families of tasks: a 2-D Minecraft-inspired crafting game (Figure 3a), 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; a 2-D maze navigation task that requires the agent to collect keys and open doors, and a 3-D locomotion task (Figure 3b) in which a quadrupedal robot must actuate its joints to traverse a narrow winding cliff.

In all tasks, 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, so these tasks cannot be solved without considering their hierarchical structure. We have released code at `http://github.com/jacobandreas/psketch`.

2. Related Work

The agent representation we describe in this paper belongs to the broader family of hierarchical reinforcement learners. As detailed in Section 3, our approach may be viewed as an instantiation 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. Most techniques for learning options rely on intermediate supervisory signals, e.g. to encourage exploration (Kearns & Singh, 2002) or completion of pre-defined subtasks (Kulkarni et al., 2016). An alternative family of approaches employs post-hoc analysis of demonstrations or pretrained policies to extract reusable sub-components (Stolle & Precup, 2002; Konidaris et al., 2011; Niekum et al., 2015). 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). We will see that the minimal supervision provided by policy sketches re-

sults in (sometimes dramatic) improvements over fully unsupervised approaches, while being substantially less onerous for humans to provide compared to the grounded supervision (such as explicit subgoals or feature abstraction hierarchies) used in previous work.

Once a collection of high-level actions exists, agents are faced with the problem of learning meta-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 meta-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, abstract plans) from context. Our model can be combined with these approaches to support a "mixed" supervision condition where sketches are available for some tasks but not others (Section 4.5).

Another 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; Marthi et al., 2004) and use reinforcement learning to fill in low-level details. 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. They perform less effectively when applied to arbitrary state representations where these assumptions do not hold (Section 4.3). We are additionally unaware of past work showing that HAM automata can be automatically inferred for new tasks given a pre-trained model, while here we show that it is easy to solve the corresponding problem for sketch followers (Section 4.5).

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 the present 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, 2016), which generate discrete computational structures but require supervision in the form of output actions or full execution traces.

We view the problem of learning from policy sketches as complementary to the instruction following problem studied in the natural language processing literature. Existing work on instruction following focuses on mapping from natural language strings to symbolic action sequences that are then executed by a hard-coded interpreter (Branavan et al., 2009; Chen & Mooney, 2011; Artzi & Zettlemoyer, 2013; Tellex et al., 2011). Here, by contrast, we focus on learning to execute complex actions given symbolic representations as a starting point. Instruction following models may be viewed as joint policies over instructions and environment observations (so their behavior is not defined in the absence of instructions), while the model described in this paper naturally supports adaptation to tasks where no sketches are available. We expect that future work might combine the two lines of research, bootstrapping policy learning directly from natural language hints rather than the semi-structured sketches used here.

3. Learning Modular Policies from Sketches

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 $q_i := \sum_{j=i+1}^{\infty} \gamma^{j-i-1} R(s_j)$. 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 $\mathcal{B}$.

3.1. Model

We exploit the structural information provided by sketches by constructing for each symbol b a corresponding *subpolicy* π_b. 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.

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

$\mathcal{D} \leftarrow \emptyset$
while $|\mathcal{D}| < D$ **do**
  // sample task τ from curriculum (Section 3.3)
  $\tau \sim \text{curriculum}(\cdot)$
  // do rollout
  $d = \{(s_i, a_i, (b_i = K_{\tau,i}), q_i, \tau), \dots\} \sim \Pi_\tau$
  $\mathcal{D} \leftarrow \mathcal{D} \cup d$
// update parameters
for $b \in \mathcal{B}, \tau \in \mathcal{T}$ **do**
  $d = \{(s_i, a_i, b', q_i, \tau') \in \mathcal{D} : b' = b, \tau' = \tau\}$
  // update subpolicy
  $\theta_b \leftarrow \theta_b + \frac{\alpha}{D} \sum_d \big(\nabla \log \pi_b(a_i|s_i)\big)\big(q_i - c_\tau(s_i)\big)$
  // update critic
  $\eta_\tau \leftarrow \eta_\tau + \frac{\beta}{D} \sum_d \big(\nabla c_\tau(s_i)\big)\big(q_i - c_\tau(s_i)\big)$

Algorithm 2 TRAIN-LOOP()

// initialize subpolicies randomly
$\mathbf{\Pi} = \text{INIT}()$
$\ell_{\text{max}} \leftarrow 1$
loop
  $r_{\text{min}} \leftarrow -\infty$
  // initialize ℓ_{max}-step curriculum uniformly
  $\mathcal{T}' = \{\tau \in \mathcal{T} : |K_\tau| \le \ell_{\text{max}}\}$
  $\text{curriculum}(\cdot) = \text{Unif}(\mathcal{T}')$
  while $r_{\text{min}} < r_{\text{good}}$ **do**
    // update parameters (Algorithm 1)
    TRAIN-STEP($\mathbf{\Pi}$, curriculum)
    $\text{curriculum}(\tau) \propto \mathbb{1}[\tau \in \mathcal{T}'](1 - \hat{\mathbb{E}} r_\tau) \quad \forall \tau \in \mathcal{T}$
    $r_{\text{min}} \leftarrow \min_{\tau \in \mathcal{T}'} \hat{\mathbb{E}} r_\tau$
  $\ell_{\text{max}} \leftarrow \ell_{\text{max}} + 1$

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 $\mathcal{A}^+ := \mathcal{A} \cup \{\text{STOP}\}$. At a high level, this framework is agnostic to the implementation of subpolicies: any function that takes a representation of the current state onto a distribution over $\mathcal{A}^+$ will do.

In this paper, we focus on the case where each π_b is represented as a neural network.[1] 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 fixed sketch $(b_1, b_2, \dots)$, 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:

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

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 expected discounted reward

$$J(\mathbf{\Pi}) := \sum_\tau J(\Pi_\tau) := \sum_\tau \mathbb{E}_{s_i \sim \Pi_\tau}\Big[\sum_i \gamma^i R_\tau(s_i)\Big]$$

across all tasks $\tau \in \mathcal{T}$.

[1] For ease of presentation, this section assumes that these subpolicy networks are independently parameterized. As described in Section 4.2, it is also possible to share parameters between subpolicies, and introduce discrete subtask structure by way of an *embedding* of each symbol b.

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):

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

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., 2015a) with $\lambda = 1$. Here c achieves close to the optimal variance (Greensmith et al., 2004) when it is set

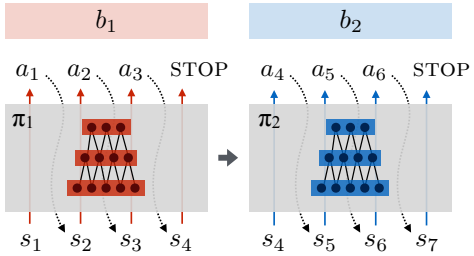


Figure 2: Model overview. Each subpolicy π is uniquely associated with a symbol b implemented as a neural network that maps from a state s_i to distributions over $\mathcal{A}^+$, and chooses an action a_i by sampling from this distribution. Whenever the STOP action is sampled, control advances to the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

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.

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 sum of gradients of expected rewards across all tasks in which π_b participates, we have:

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

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 with parameters η_τ. This is trained to minimize a squared error criterion, with gradients given by

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

Alternative forms of the advantage estimator (e.g. the TD residual $R_\tau(s_i) + \gamma V_\tau(s_{i+1}) - V_\tau(s_i)$ or any other member of the generalized advantage estimator family) can be easily substituted by simply maintaining one such estimator per task. Experiments (Section 4.4) 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.

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 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 τ. Then at each timestep, tasks are sampled in proportion to $1 - \hat{\mathbb{E}} r_\tau$, which by assumption must be positive.

Intuitively, the tasks that provide the strongest learning signal are those in which (1) the agent does not on average achieve reward close to the upper bound, but (2) many episodes result in high reward. The expected reward component of the curriculum addresses condition (1) by ensuring that time is not spent on nearly solved tasks, while the length bound component of the curriculum addresses condition (2) by ensuring that tasks are not attempted until high-reward episodes are likely to be encountered. Experiments show that both components of this curriculum learning scheme improve the rate at which the model converges to a good policy (Section 4.4).

The complete curriculum-based training procedure is specified in Algorithm 2. Initially, the maximum sketch length

ℓ_{max} is set to 1, 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 TRAIN-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_{good} is exceeded, at which point ℓ_{max} is incremented and the process repeated over a (now-expanded) collection of tasks.

4. Experiments

We evaluate the performance of our approach in three environments: a crafting environment, a maze navigation environment, and a cliff traversal environment. These environments involve various kinds of challenging low-level control: agents must learn to avoid obstacles, interact with various kinds of objects, and relate fine-grained joint activation to high-level locomotion goals. They also feature hierarchical structure: most rewards are provided only after the agent has completed two to five high-level actions in the appropriate sequence, without any intermediate goals to indicate progress towards completion.

4.1. Implementation

In all our experiments, we implement each subpolicy as a feedforward neural network 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 D in Algorithm 1 to be 2000, and set $\gamma = 0.9$ in both environments. For curriculum learning, the improvement threshold r_{good} is 0.8.

4.2. Environments

The crafting environment (Figure 3a) is inspired by the popular game Minecraft, but is implemented in a discrete 2-D world. The agent may interact with objects in the world by facing them and executing a special USE 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. Each task

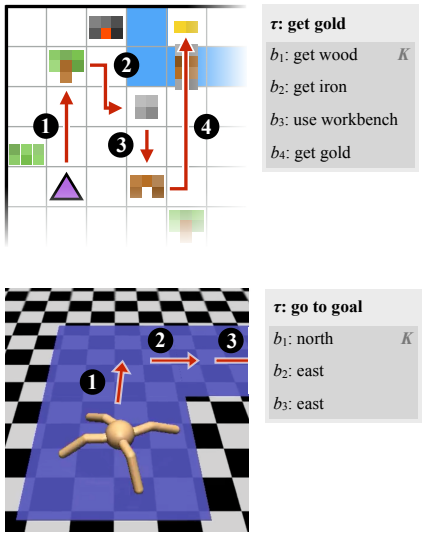
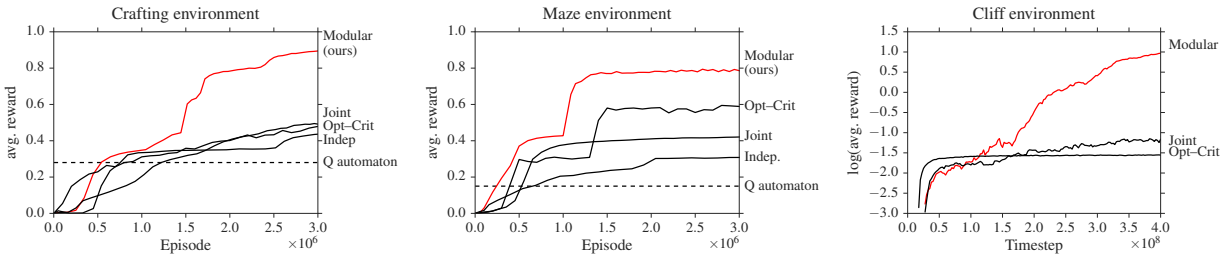


Figure 3: Examples from the crafting and cliff environments used in this paper. An additional maze environment is also investigated. (a) In the crafting environment, an agent seeking to pick up the gold nugget in the top corner must first collect wood (1) and iron (2), use a workbench to turn them into a bridge (3), and use the bridge to cross the water (4). (b) In the cliff environment, the agent must reach a goal position by traversing a winding sequence of tiles without falling off. Control takes place at the level of individual joint angles; high-level behaviors like "move north" must be learned.

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.

The maze environment (not pictured) 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. The sketch always corresponds to a viable traversal from the start to the goal position, but other (possibly shorter) traversals may also exist.

The cliff environment (Figure 3b) is intended to demonstrate the applicability of our approach to problems involving high-dimensional continuous control. In this environment, a quadrupedal robot (Schulman et al., 2015b) is placed on a variable-length winding path, and must navi-



(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, **Q automaton** learns a single network to map from states and action symbols to Q values, and **Opt–Crit** is an unsupervised option learner. Performance for the best iteration of the (off-policy) Q automaton is plotted. Performance is shown in (a) the crafting environment, (b) the maze environment, and (c) the cliff environment. The modular approach is eventually able to achieve high reward on all tasks, while the baseline models perform considerably worse on average.

gate to the end without falling off. This task is designed to provide a substantially more challenging RL problem, due to the fact that the walker must learn the low-level walking skill before it can make any progress, but has simpler hierarchical structure than the crafting environment. The agent receives a small reward for making progress toward the goal, and a large positive reward for reaching the goal square, with a negative reward for falling off the path.

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

4.3. 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. We aim to explore the differences between the approach described in Section 3 and relevant prior work that performs either unsupervised or weakly supervised multitask learning of hierarchical policy structure. Specifically, we compare our **modular** to approach to:

1. Structured hierarchical reinforcement learners:
   (a) the fully unsupervised **option–critic** algorithm of Bacon & Precup (2015)
   (b) a **Q automaton** that attempts to explicitly represent the Q function for each task / subtask combination (essentially a HAM (Andre & Russell, 2002) with a deep state abstraction function)
2. Alternative ways of incorporating sketch data into standard policy gradient methods:
   (c) learning an **independent** policy for each task
   (d) learning a **joint** policy across all tasks, conditioning directly on both environment features and a representation of the complete sketch

The joint and independent models performed best when trained with the same curriculum described in Section 3.3, while the option–critic model performed best with a length–weighted curriculum that has access to all tasks from the beginning of training.

Learning curves for baselines and the modular model are shown in Figure 4. It can be seen that in all environments, 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.

4.4. 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

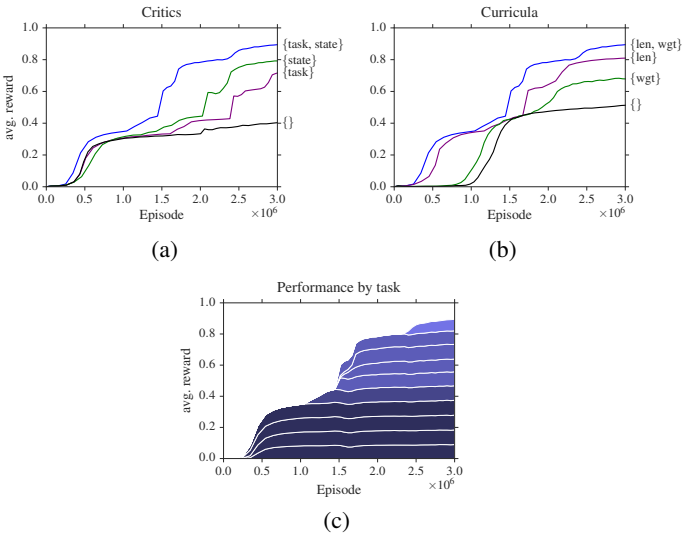


Figure 5: Training details in the crafting domain. (a) Critics: lines labeled "task" include a baseline that varies with task identity, while lines labeled "state" include a baseline that varies with state identity. Estimating a baseline that depends on both the representation of the current state and the identity of the current task is better than either alone or a constant baseline. (b) Curricula: lines labeled "len" use a curriculum with iteratively increasing sketch lengths, while lines labeled "wgt" sample tasks in inverse proportion to their current reward. Adjusting the sampling distribution based on both task length and performance return improves convergence. (c) Individual task performance. Colors correspond to task length. Sharp steps in the learning curve correspond to increases of $\ell_{\max}$ in the curriculum.

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. Both components help; prioritization by both length and weight gives the best results.

4.5. Zero-shot and Adaptation Learning

In our final experiments, we consider the model's ability to generalize beyond the standard training condition. We first consider two tests of generalization: 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 via interaction in the new task.

Model	Multitask	0-shot	Adaptation
Joint	.49	.01	–
Independent	.44	–	.01
Option–Critic	.47	–	.42
Modular (ours)	**.89**	**.77**	**.76**

Table 1: Accuracy and generalization of learned models in the crafting domain. The table shows the task completion rate for each approach after convergence under various training conditions. **Multitask** is the multitask training condition described in Section 4.3, while **0-Shot** and **Adaptation** are the generalization experiments described in Section 4.5. Our modular approach consistently achieves the best performance.

We hold out two length-four tasks from the full inventory used in Section 4.3, 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 RL over high-level actions $\mathcal{B}$ rather than low-level actions $\mathcal{A}$, implementing the high-level learner with the same agent architecture as described in Section 3.1. Note that the Independent and Option–Critic models cannot be applied to the zero-shot evaluation, while the Joint model 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: in particular, the joint model overfits to the training tasks and cannot generalize to new sketches, while the independent model cannot discover enough of a reward signal to learn in the adaptation setting. The modular model does comparatively well: individual subpolicies succeed in novel zero-shot configurations (suggesting that they have in fact discovered the behavior suggested by the semantics of the sketch) and provide a suitable basis for adaptive discovery of new high-level policies.

5. Conclusions

We have described an approach for multitask learning of deep multitask 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.

Acknowledgments

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

References

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

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

Andreas, Jacob, Rohrbach, Marcus, Darrell, Trevor, and Klein, Dan. 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.

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

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

Bakker, Bram and Schmidhuber, Jürgen. 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.

Bengio, Yoshua, Louradour, Jérôme, Collobert, Ronan, and Weston, Jason. Curriculum learning. In *International Conference on Machine Learning*, pp. 41–48. ACM, 2009.

Branavan, S.R.K., Chen, Harr, Zettlemoyer, Luke S., and Barzilay, Regina. 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.

Chen, David L. and Mooney, Raymond J. 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.

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

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

Dietterich, Thomas G. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res. (JAIR)*, 13:227–303, 2000.

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

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

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

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

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

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

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

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

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

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

Niekum, Scott, Osentoski, Sarah, Konidaris, George, Chitta, Sachin, Marthi, Bhaskara, and Barto, Andrew G. Learning grounded finite-state representations from unstructured demonstrations. *The International Journal of Robotics Research*, 34(2):131–157, 2015.

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

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

Reed, Scott and de Freitas, Nando. Neural programmer-interpreters. *Proceedings of the International Conference on Learning Representations*, 2016.

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

Schulman, John, Moritz, Philipp, Levine, Sergey, Jordan, Michael, and Abbeel, Pieter. Trust region policy optimization. In *International Conference on Machine Learning*, 2015b.

Socher, Richard, Huval, Brody, Manning, Christopher, and Ng, Andrew. 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.

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

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

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

Tieleman, Tijmen. RMSProp (unpublished), 2012.

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

Vogel, Adam and Jurafsky, Dan. 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.

Williams, Ronald J. 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 4.5, but included in the multitask training experiments in Sections 4.3 and 4.4.

Goal	Sketch				
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
Maze environment					
room 1	left	left			
room 2	left	down			
room 3	right	down			
room 4	up	left			
room 5	up	right			
room 6	up	right	up		
room 7	down	right	up		
room 8	left	left	down		
room 9	right	down	down		
room 10	left	up	right		
Cliff environment					
path 0	north				
path 1	east				
path 2	south				
path 3	west				
path 4	west	south			
path 5	west	north	north		
path 6	north	east	north		
path 7	west	north			
path 8	east	south			
path 9	north	west	west		
path 10	east	north	east		
path 11	south	east			
path 12	south	west			
path 13	south	south			
path 14	south	south	west		
path 15	east	south	south		
path 16	east	east			
path 17	east	north			
path 18	north	east			
path 19	west	west			
path 20	north	north			
path 21	north	west			
path 22	west	west	south		
path 23	south	east	south		