Augmenting Policy Learning with Routines Discovered from a Single Demonstration

Augmenting Policy Learning with Routines Discovered from a Single Demonstration

Zelin Zhao * [1] **, Chuang Gan** [2]**, Jiajun Wu** [3]**, Xiaoxiao Guo** [2]**, Joshua Tenenbaum** [4]

[1] Shanghai Jiao Tong University
[2] MIT-IBM Watson AI Lab
[3] Stanford University
[4] Massachusetts Institute of Technology

Abstract

Humans can abstract prior knowledge from very little data and use it to boost skill learning. In this paper, we propose routine-augmented policy learning (RAPL), which discovers routines composed of primitive actions from a single demonstration and uses discovered routines to augment policy learning. To discover routines from the demonstration, we first abstract routine candidates by identifying grammar over the demonstrated action trajectory. Then, the best routines measured by length and frequency are selected to form a routine library. We propose to learn policy simultaneously at primitive-level and routine-level with discovered routines, leveraging the temporal structure of routines. Our approach enables imitating expert behavior at multiple temporal scales for imitation learning and promotes reinforcement learning exploration. Extensive experiments on Atari games demonstrate that RAPL improves the state-of-the-art imitation learning method SQIL and reinforcement learning method A2C. Further, we show that discovered routines can generalize to unseen levels and difficulties on the CoinRun benchmark.

Introduction

Extensive evidence from cognitive psychology and neuroscience suggests that humans are remarkably capable of abstracting knowledge from very few observations to boost practice in new scenarios. For instance, behavioral experiments on the Atari games (Tsividis et al. 2017) have demonstrated that human game players could learn from a video of one episode and earn more than double scores than those who do not watch the video. On the contrary, previous Learning from Demonstrations (LfD) approaches either require a large amount of pre-collected data (Esmaili, Sammut, and Shirazi 1995), an active oracle (Ross, Gordon, and Bagnell 2010), or a family of similar tasks (Kipf et al. 2019). In this paper, we focus on the following question: how can a single demonstration promote policy learning?

Two challenges exist when learning from a single demonstration. First, the agent would often drift away from the few seen expert observations and not return to demonstrated states. Second, high-dimension value function approximators such as deep neural networks (Mnih et al. 2015) may over-fit the few demonstrated state-action pairs and cannot overcome unseen environment dynamics. We propose to abstract routines from the demonstration via a non-parametric algorithm and use the routines to help policy learning to address these problems. This idea can alleviate the out-of-distribution problem because routines force the agent to follow segments of the demonstration. Besides, the process of decomposing the demonstration is non-parametric, making the learned policy generalizable to unseen states.

The overview of the proposed approach is shown in Figure 1. A library of routines that represent useful skills is abstracted from the demonstration. The routines can be used in two settings. First, the agent could imitate expert behaviors at multiple temporal scales without access to the reward signal. Second, in reinforcement learning, the abstracted routines can promote deeper exploration and long-range value learning. However, previous option learning approaches must rely on reward signals (Bacon, Harb, and Precup 2016; Stolle and Precup 2002; Sutton, Precup, and Singh 1999).

We propose a two-phase model for routine discovery. During the first phase, we adopt a non-parametric algorithm, Sequitur (Nevill-Manning and Witten 1997), to discover the structure of the demonstration. Each element in the structure is treated as one routine proposal. In the second phase, we select the best proposals by the frequency and lengths of routine candidates to form a routine library. Too similar routine candidates are pruned to keep the parsimony of the routine library. This model can effectively discover routines without a time-consuming training procedure.

The discovered routines are then used as higher-level actions to boost exploration and policy learning. A naïve approach is to run an off-the-shelf policy learning algorithm based on the augmented action space composed by routines and action primitives (Durugkar et al. 2016; Chang et al. 2019). The problem of such an approach is that it ignores the inner structure of routines, so experiences from routine execution are exclusively used to update values at the routine level, which would slow down value learning at the primitive level. Such conflict turns to be a bigger issue as the number of routines grows. To address this problem, since the routines are temporally decomposable, we reuse routine execution experiences to update the value function at the primitive level. Our approach harmonizes the relationship between routines and primitives and has stronger performance when utilizing

*Work was done when Zelin was a visiting student at MIT.

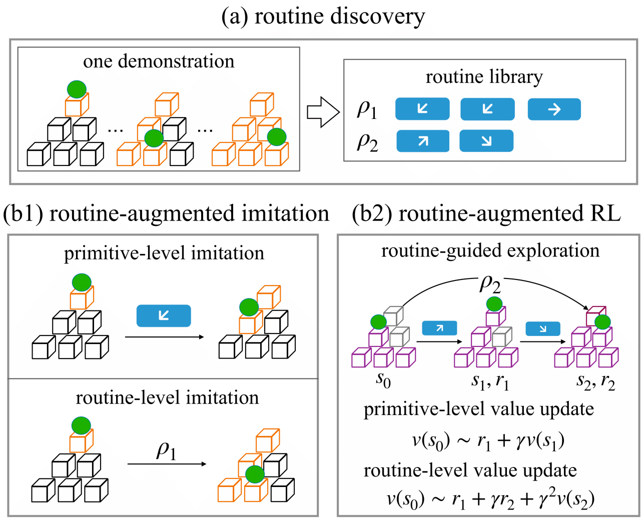
(a) routine discovery
one demonstration
routine library
ρ_1
ρ_2
(b1) routine-augmented imitation
(b2) routine-augmented RL
primitive-level imitation
routine-level imitation
ρ_1
routine-guided exploration
ρ_2
s_0
s_1, r_1
s_2, r_2
primitive-level value update
$v(s_0) \sim r_1 + \gamma v(s_1)$
routine-level value update
$v(s_0) \sim r_1 + \gamma r_2 + \gamma^2 v(s_2)$

Figure 1: Schematic of routine-augmented policy learning (RAPL). In the examples, the green ball represents an agent, which needs to step on every square to change its color (a mini version of Qbert from Bellemare et al. (2012)). a: We propose to discover a library of routines from a single demonstration. The abstracted routines can be applied to augment both imitation learning and reinforcement learning. b1: For imitation learning (no reward signal), the discovered routines can help the agent imitate the expert's behavior at multiple temporal scales. b2: For reinforcement learning (with reward signal), routines can help exploration as policy shortcuts. The experiences from routine execution are fully exploited to conduct value approximation at both the routine level and the primitive level.

more and longer routines.

This paper's main contribution is *routine-augmented policy learning* (RAPL): an approach to discover routines from a single demonstration and use them to augment policy learning. Through extensive experiments on the Atari benchmark (Bellemare et al. 2012), we find that our approach can improve both A2C (Mnih et al. 2016) and SQIL (Reddy, Dragan, and Levine 2019) on most of the games. Moreover, we conduct generalization experiments on CoinRun (Cobbe et al. 2018) and observe that the abstracted routines can successfully generalize to unseen levels and harder cases. Our code is now available at https://github.com/sjtuytc/AAAI21-RoutineAugmentedPolicyLearning.

Related Work

Imitation Learning. The goal of imitation learning is to learn a policy from the demonstration (Argall et al. 2009). Behavior Cloning (Esmaili, Sammut, and Shirazi 1995) only succeeds with a large amount of data. To efficiently leverage the demonstrations, GAIL (Ho and Ermon 2016) utilizes adversarial training to prioritizes the demonstration over others. Our approach is different from those approaches because they do not consider discovering higher-level actions from the demonstrations. Besides, we only assume access to one demonstration and need neither a large number of demonstrations nor a family of similar tasks (Duan et al. 2017).

Demonstrations Guided RL. Reinforcement Learning (RL) requires huge time costs and extensive sampling to learn a good strategy (Thrun 1992; Pathak et al. 2017). Since humans may have prior knowledge of the given task (Wingate et al. 2011), much recent work (Hester et al. 2018; Vecerik et al. 2017; Kang, Jie, and Feng 2018; Nair et al. 2018) proposes to leverage demonstrations to help RL. These methods add extra costs in policy learning to penalize the deviation between the learned and the expert policy. Another approach (Salimans and Chen 2018) utilizes one demonstration to play Montezuma's Revenge, a hard exploration game, by resetting the agent to states in the demonstration. These methods have not considered discovering routines from the demonstration. Moreover, DQfD-based (Hester et al. 2018; Vecerik et al. 2017; Kang, Jie, and Feng 2018) approaches assume access to reward signals, while our proposed algorithm can also improve imitation learning from one demonstration.

Macro-Actions. Macro-actions are temporally extended actions built on primitive actions. In robotics, the classical STRIPS system (Fikes and Nilsson 1971; Minton 1985; Dawson and Siklossy 1977; Fikes, Hart, and Nilsson 1972; McGovern and Sutton 1998) uses predefined *routines* to accelerate making plans. Notably, a few concurrent works consider the discovery of macro actions from the agent's good experiences (Chang et al. 2019; Christodoulou et al. 2019; Garcia, da Silva, and Thomas 2019). Our work is different from them in several folds. First, they adopt an off-the-shelf RL algorithm to train over an action space of macro-actions and primitives. But we propose an efficient and sound manner to train a routine policy. Besides, we propose using routines to augment imitation learning, but they only study adopting macro-actions under reinforcement learning. Third, we do not require the knowledge or the approximation of the environment dynamics, different from Garcia, da Silva, and Thomas (2019). We compare to Durugkar et al. (2016) in experiments.

The Option Frameworks. Our work is also related to literature under the *option framework* which learns *options* specified by an initialization set, an intra-option policy, and a termination condition (Randlov 1999; Barto and Mahadevan 2003; Bacon, Harb, and Precup 2016; Machado, Bellemare, and Bowling 2017; Riemer, Liu, and Tesauro 2018; Kulkarni et al. 2016; Le et al. 2018). Our idea of learning at multiple temporal scales originates from Hierarchical Reinforcement Learning (Kulkarni et al. 2016), which jointly learns a meta-controller over options and bottom-level modules to achieve the targets specified in each option. No demonstrations are involved in this work. PolicyBlocks (Pickett and Barto 2002) attempts to discover reusable options from optimal policies. However, it requires a family of tasks to discover options. Some recent work (Fox et al. 2017; Krishnan et al. 2017; Kipf et al. 2019; Shankar et al. 2020) proposes to discover options from demonstrations and train a controller upon abstracted options. Unlike options adopted in these approaches, our routines are state-independent, and we leave the job of connecting the state with higher-level actions to the phase of policy learning. Furthermore, learning sub-task policies would consume a large number of demonstrations to overcome unseen dynamics, while our approach requires only a single demonstration. We compare to two option learning baselines (ComPILE (Kipf et al. 2019) and OptionCritic (Bacon, Harb, and Precup 2016)) in our experiments.

Routine-Augmented Policy Learning (RAPL)

Model Basic

MDPs. During a timestep t on an Markov Decision Process (MDP) Γ, the agent chooses an action a_t from a predefined primitive action set $\mathcal{A}$ after receiving an observation state $s_t \in \mathcal{S}$. The environment provides a transition function $\mathcal{T}(s_t, a_t)$, a reward r_t (not available in imitation learning), and a discount factor γ. The core problem of an MDP is to find a policy function $\pi(a_t|s_t)$. In this paper, we focus on MDPs with high-dimensional states and discrete actions.

Routines and Routine Policies. We define a routine ρ to be a sequence of primitive actions $(a^{(1)}, a^{(2)}, ..., a^{(|\rho|)})$ and $|\rho|$ to be its length. The notion of routine appeared in Fikes and Nilsson (1971) and we emphasis that routines are abstracted from demonstrations in this paper (different from hand-crafted macro actions). A *routine library* $\mathcal{L}$ is defined to be a set of discovered routines for a task. After routines are introduced, an agent can choose one routine $\rho_t \in \mathcal{L}$ or a primitive action $a_t \in \mathcal{A}$ based on a state $s_t \in \mathcal{S}$. When a routine ρ_t is chosen, the primitive actions in ρ_t are executed sequentially, and the agent would make the next decision after the execution of $a^{(|\rho_t|)}$. For convenience, we use $\widetilde{\mathcal{L}} = A \cup \mathcal{L}$ to represent the routine-augmented action space and $\widetilde{\rho} \in \widetilde{\mathcal{L}}$ to represent an *extended routine*. Plus, we define $|\widetilde{\rho}|$ to be the length of $\widetilde{\rho}$ (the length of a primitive action is one). The goal is to find a *routine policy* $\pi(\widetilde{\rho}_t|s_t)$, which specifies the distribution of extended routines for a state at timestep t.

Routine Discovery

We propose a two-phase algorithm for routine discovery from a single demonstration. During the first phase, we construct a set of routine proposals from the demonstration. After that, we select the best routines from the routine candidates measured by frequency and length. Those selected best routines form a routine library to augment policy learning. The pseudocode of routine discovery is provided in the supplementary material.

Routine Proposal. The key idea is that one can decompose the demonstration and consider each segment as a routine proposal. We adopt a non-parametric algorithm, Sequitur (Nevill-Manning and Witten 1997), to recover the structure of the demonstration. Sequitur takes the demonstrated action trajectory as input and outputs a context-free grammar generating the whole action sequence. The grammar is represented as a set of rules. Each rule in the grammar connects from a variable to a sequence of variables. Sequitur introduces intermediate variables, each of which can be transferred to a sequence of terminal variables (variables that do not connect to any variables in the rules). Each terminal variable corresponds to a primitive action in the demonstrated action sequence. Therefore, each intermediate variable can be considered as a routine candidate. We refer readers to Nevill-Manning and Witten (1997) for more details about Sequitur.

Routine Selection. After acquiring the routine candidates, we use a selection procedure to limit the routine library's size to be K, a hyper-parameter. We adopt a hybrid metric, considering both the frequency and length of the routine proposals. On the one hand, routines frequently appear in the demonstration may entail useful skills to conquer tasks. On the other hand, we encourage selecting longer routines to encode more expert policy patterns. Denote the occurrence time of one routine ρ in the demonstrated action sequence to be $f(\rho)$, and its length to be $|\rho|$. The score of a routine can be written as $f(\rho) + \lambda^{\text{length}}|\rho|$, where λ^{length} is a balancing factor. To prevent introducing too many similar routines, we only leave the routine with the highest score when similar routines are detected. The similarity is measured by the Levenshtein distance (Miller, Vandome, and McBrewster 2009), which is the edit distance of two sequences. Finally, the K routine candidates with the highest scores are selected to form a routine library.

Routine Policy Learning

After introducing routine library $\mathcal{L}$, the agent's action space becomes $\widetilde{\mathcal{L}} = A \cup \mathcal{L}$. One naïve approach is to regard routines as black-box actions and use an off-the-shelf policy learning algorithm to train an agent with the augmented action space $\widetilde{\mathcal{L}}$ (Durugkar et al. 2016; Garcia, da Silva, and Thomas 2019). Such an approach fails to consider the temporal structure of routines and would slow down policy learning when $\widetilde{\mathcal{L}}$ consists of more and longer routines. We propose to reuse experiences at multiple temporal scales to update policy efficiently.

We instantiate this idea in two settings. On the one hand, when the reward is not available, routines are used to augment SQIL (Reddy, Dragan, and Levine 2019), a state-of-the-art imitation learning algorithm, to enable imitation learning over multiple temporal scales. On the other hand, we use routines to promote the standard reinforcement learning algorithm A2C. We formulate the learning targets for those two algorithms in the following paragraphs.

RAPL-SQIL. SQIL (Reddy, Dragan, and Levine 2019) is a recently proposed simple yet effective imitation learning approach. It gives all the experiences from the demonstration a constant reward $r = 1$. Besides, all the newly explored experiences are given a reward $r = 0$. This can encourage the agent to go back to the demonstrated states. The demonstration is represented as D_{prim}, where each element in D_{prim} is a tuple (s_t, a, s_{t+1}). We find all the occurrences of every discovered routine $\rho \in \mathcal{L}$ in the demonstrated action sequence. Combining each occurrence with the states before and after routine execution in the demonstration, we get a higher-level demonstration D_{routine}. Each entry in D_{routine} is represented as $(s_t, \rho, s_{t+|\rho|})$, where s_t and $s_{t+|\rho|}$ are the states before and after the execution of ρ correspondingly. Therefore, D_{routine} and D_{prim} contain experiences in routine-level and primitive-level accordingly. The squared soft Bellman error is given as

$$\delta^2(\mathcal{D}, r) = \frac{1}{|\mathcal{D}|} \sum_{(s_t, \widetilde{\rho}, s_{t+|\widetilde{\rho}|}) \in \mathcal{D}} \left(Q_\theta(s_t, \widetilde{\rho}) - Q_{\text{target}}(\widetilde{\rho}, s_{t+|\widetilde{\rho}|}, r)\right)^2, \tag{1}$$

$$Q_{\text{target}}(\widetilde{\rho}, s_{t+|\widetilde{\rho}|}, r) = R_{sq}(\widetilde{\rho}, r) + \Gamma(\widetilde{\rho}) \log \left(\sum_{\widetilde{\rho}' \in \widetilde{\mathcal{L}}} \exp\left(Q_\theta\left(s_{t+|\widetilde{\rho}|}, \widetilde{\rho}'\right)\right) \right), \tag{2}$$

where $R_{sq}(\widetilde{\rho}, r)$ and $\Gamma(\widetilde{\rho})$ are the reward function and the discount factor defined for the extended routine $\widetilde{\rho}$. Since the execution of routines connects two states with an interval of $|\widetilde{\rho}|$, we define the extended routine's reward function to be the sum of discounted primitive rewards and its discount factor to be λ discounted by $|\widetilde{\rho}|$ times. Formally,

$$R_{sq}(\widetilde{\rho}, r) = \sum_{\tau=1}^{|\widetilde{\rho}|} \gamma^{\tau-1} r, \qquad \Gamma(\widetilde{\rho}) = \gamma^{|\widetilde{\rho}|}. \tag{3}$$

The final loss of SQIL with routines is

$$\mathcal{L}^{\mathcal{SR}} = \delta^2(\mathcal{D}_{\text{prim}} \cup \mathcal{D}_{\text{routine}}, 1) + \lambda_{\text{sample}} \delta^2(\mathcal{D}_{\text{sample}}, 0), \tag{4}$$

where D_{sample} represents the collected experiences of interactions with the environments and λ_{sample} is the balancing hyperparameter between the demonstrated and explored transitions.

RAPL-A2C. We apply the augmented action space to a state-of-the-art reinforcement learning method Advantage Actor Critic (A2C) (Mnih et al. 2016). A2C with routines learns a policy function $\pi(\widetilde{\rho}_t|s_t; \theta_\pi)$ and a state value function $V(s_t; \theta_v)$. We compute two advantage functions to backtrack delayed rewards to the current state, differing in their temporal granularity. In the first advantage function A_{routine}, we compute the return from N-step of routine experiences. Denote the explored on-policy experiences of routine execution to be $\{(s_{t_\tau}, \widetilde{\rho}_{t_\tau}, R_{t_\tau}, s_{t_{\tau+1}})|0 \le \tau \le N-1\}$, where $t_i = t_0 + \sum_{\tau=0}^{i-1} |\widetilde{\rho}_{t_\tau}|$. Note the total primitive steps are $\sum_{\tau=0}^{N-1} |\widetilde{\rho}_{t_\tau}|$, which could be much larger than N. The reward of a routine is the sum of discounted primitive rewards, so we have $R_{t_i} = \sum_{\tau=t_i}^{t_{i+1}-1} \gamma^{\tau - t_i} r_\tau$. Then we can write the routine-level advantage function as

$$A_{\text{routine}} = \sum_{i=0}^{N-1} \gamma^{t_i - t_0} R_{t_i} + \gamma^{t_N - t_0} V(s_{t_N}) - V(s_{t_0}). \tag{5}$$

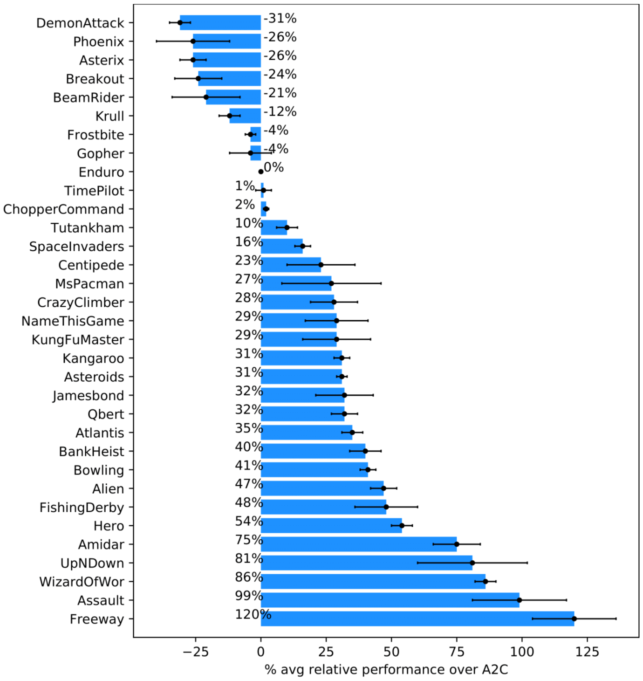

Figure 2: Relative performance of RAPL-A2C over A2C on Atari. Denote S_R as the score of RAPL-A2C and S_A is the score of A2C. The relative performance is calculated by $(S_R - S_A)/|S_A| \times 100\%$. Each number is averaged over five random agents and we also plot the stand error of the numbers.

In the second advantage function, we take care of the primitive-level value approximation and compute N-step bootstrapping for primitives. From the experiences of routine execution, we randomly sample an N-step consecutive primitive experience, represented as $\{(s_\tau, a_\tau, r_\tau, s_{\tau+1})|t_j \le \tau \le t_j + N - 1\}$ (note that we can get access to the intermediate states during routine execution). Then we give the primitive-level advantage function as

$$A_{\text{prim}} = \sum_{i=0}^{N-1} \gamma^i r_{t_j+i} + \gamma^N V(s_{t_j+N}) - V(s_{t_j}). \tag{6}$$

To optimize the policy function, we pose a policy gradient loss and an entropy loss:

$$\mathcal{L}^{\text{policy}} = -A_{\text{routine}} \log \pi(\widetilde{\rho}_t|s_{t_0}; \theta_\pi), \tag{7}$$

$$\mathcal{L}^{\text{entropy}} = \sum_{\widetilde{\rho}} \pi(\widetilde{\rho}|s_{t_0}; \theta_\pi) \log \pi(\widetilde{\rho}|s_{t_0}; \theta_\pi). \tag{8}$$

The final loss for A2C with routines is

$$\mathcal{L}^{\mathcal{AR}} = \mathbb{E}(\mathcal{L}^{\text{policy}} + \lambda^{\text{entropy}} \mathcal{L}^{\text{entropy}} + \lambda^{\text{value}}(\|A_{\text{routine}}\|^2 + \lambda^{\text{prim}} \|A_{\text{prim}}\|^2)), \tag{9}$$

where the expectation is taken over all sampled experiences. We denote λ^{value}, λ^{prim}, λ^{entropy} to be the balancing factors for each loss term.

Experiments

We investigate the following questions by experiments: 1) Does RAPL improve imitation learning and reinforcement

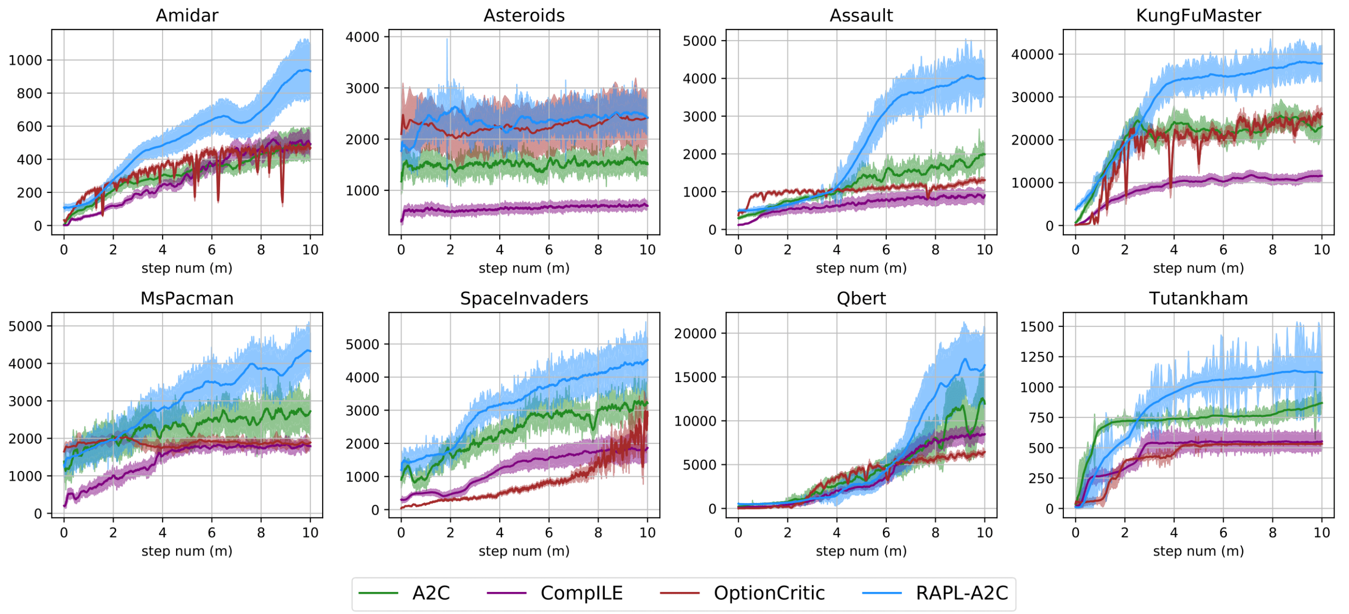

Figure 3: Training curves on eight randomly selected Atari games in comparison with several RL baselines. We plot both the mean and standard deviation in those curves across five agents with random seeds.

	Alignment (± std)	Mean (± std)
BC	0.18 (± 0.03)	18.3% (2.1%)
GAIL	0.16 (± 0.08)	26.4% (1.6%)
SQIL	0.28 (± 0.07)	29.4% (3.2%)
RAPL-SQIL	**0.34 (± 0.07)**	**36.1% (± 3.6%)**

Table 1: Comparing with several imitation learning baselines on 33 Atari games. We shown both alignment scores (defined in Eq. 10) and mean of human-normalized scores (Mnih et al. 2015) which indicates the alignment performance with regarding to the demonstration. Each number in the table is averaged over five random seeds.

learning methods? 2) Does our approach outperform other baselines to learn from demonstrations? 4) How does our approach perform when scaling to more and longer routines? 4) Can discovered routines generalize to unseen scenarios?

Experimental Setting

Environment Description. Our experiments are conducted on the Atari benchmark (Bellemare et al. 2012) and CoinRun (Cobbe et al. 2018). We use 33 Atari games selected by Sharma, Lakshminarayanan, and Ravindran (2017) (including all the games in their experiments expect for Koolaid that is not supported in our experiment platform Gym (Brockman et al. 2016)). We use a frame-skip of 4, a frame-stack of 4, and the minimal action space (Bellemare et al. 2012). We use the convolutional neural network described in Mnih et al. (2015) on Atari games. CoinRun is a recent benchmark that has different levels that enable quantifying the generalization ability of RL methods. It also provides two difficulties modes: easy and hard. We adopt a minimal action space composed of `Left, Down, Up, Right, Nope` for the convenience of presentation. We do not paint velocity information in the observation. No frame-stack is used in CoinRun as in Cobbe et al. (2018). For CoinRun, we use the IMPALA-CNN architecture (Espeholt et al. 2018). All the environmental settings are kept the same for all approaches to ensure fairness.

Demonstration Collection. For all the games, we only use one demonstration generated by a trained A2C agent. We use $\lambda^{\text{value}} = 0.5$ and $\lambda^{\text{entropy}} = 0.01$ to balance the value loss and entropy loss accordingly. We set $\lambda^{\text{prim}} = 1.0$ when using routine augmentation. The optimizer is RMSProp with a learning rate 7×10^{-4}, a linear decay of 10^{-5} per timestep. We use entropy regularization with $\beta = 0.02$. The return is calculated for $N = 5$ steps. Each agent is trained for 10 million steps.

Routine Discovery. In all experiments, we set the balancing factor between frequency and length to be $\lambda^{\text{length}} = 0.1$. Moreover, the number of routines is set to $K = 3$. We would leave the best routine between routines whose Levenshtein distance is smaller than $\alpha = 2$. These hyper-parameters are coarsely selected by validating on a few games (refer to Supplementary for details), and they are kept all the same for all the other games.

Imitation Learning with Routines

We validate whether discovered routines can improve SQIL (Reddy, Dragan, and Levine 2019) and compare our results with Behavior Cloning (BC) (Esmaili, Sammut, and Shirazi 1995), which conducts supervised learning from demonstra-

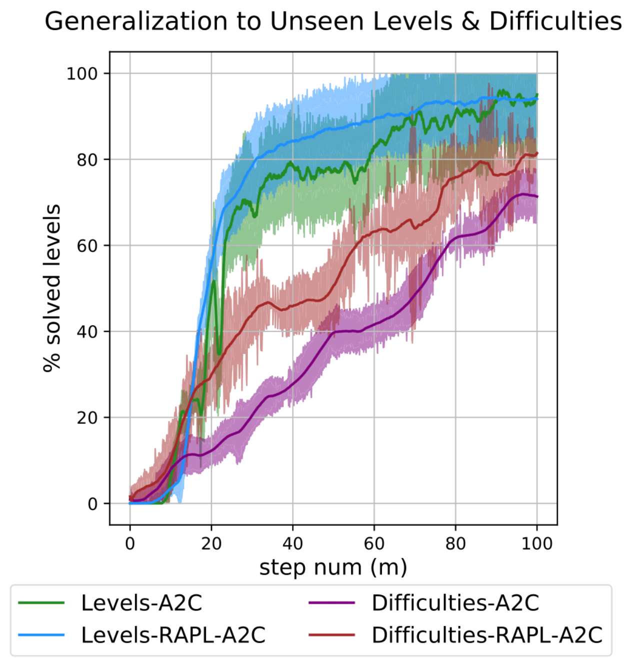

Figure 4: Generalization curves on CoinRun. We use ”Levels” and ”Difficulties” to indicate generalization to unseen levels and difficulties accordingly. We show both the mean and the standard deviation across five random seeds.

tion data without any environment interaction. Moreover, we compare with a standard model-free imitation learning algorithm GAIL (Ho and Ermon 2016). We thank the author of SQIL (Reddy, Dragan, and Levine 2019) for providing the implementation of these algorithms. As described in Reddy, Dragan, and Levine (2019), we use $\lambda^{\text{sample}} = 1$. The optimizer is Adam (Kingma and Ba 2015) with a learning rate 10^{-3}. The agent is trained via 10^5 on-policy rollouts. Each score reported is the average reward on 100 episodes after training.

We propose a metric of alignment score to measure how well the imitator imitates the expert. Given the demonstrated action trajectory ι_d and the action trajectory produced by the trained agent ι_t (note ι_t is padded or cut to have the same length with ι_d), we compute the alignment score s as

$$s = 1 - \frac{D(\iota_d, \iota_t)}{|\iota_d|}, \tag{10}$$

where D is the Levenshtein distance and $|\iota_d|$ denotes the length of the demonstration.

We present the results in Table 1. We notice that RAPL-SQIL could help the agent perform in line with the demonstration. The agent effectively learns when to use routine through a single demonstration and environmental interactions. The results indicate that routines can effectively force the agent to follow the patterns of the single demonstration. Besides, this fact suggests that imitating expert's policy at multiple temporal scales would enhance imitation learning.

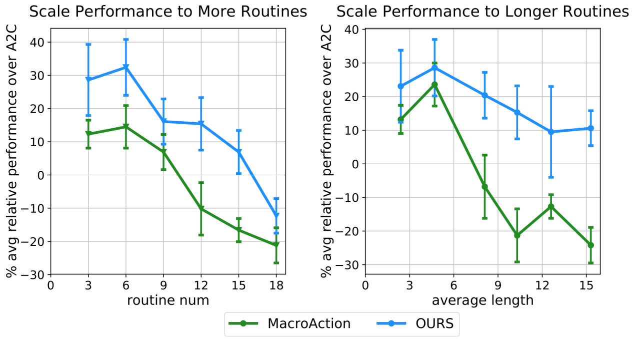

Figure 5: The scalability of our approach on Atari games. Each number represents the relative performance over A2C averaged on 33 Atari games. Mean and standard error over five random agents are shown in the figure.

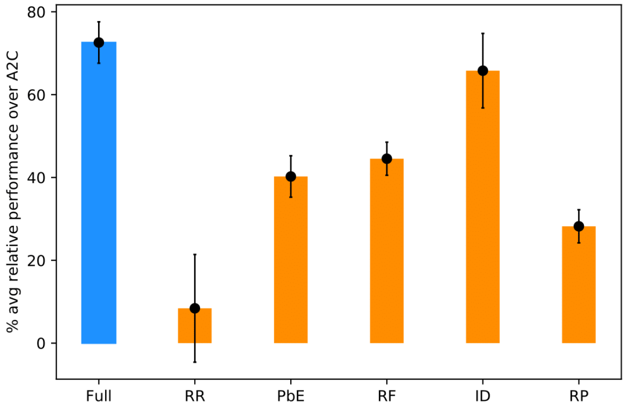

Figure 6: Comparison of ablated routine discovery models on Atari games. Mean and standard error over five random agents are shown in the figure.

Reinforcement Learning with Routines

We first study whether routine discovery can improve model-free reinforcement learning method A2C (Mnih et al. 2016). We then compare with a recent proposed parametric routine discovery approach ComPILE (Kipf et al. 2019). ComPILE first decomposes the demonstration into segments via a parametric recognition model; it then trains sub-policy and the termination condition for each segment via supervised learning. After that, it trains an A2C controller over an augmented space composed of those segments and primitives. We further compare to an option learning baseline, OptionCritic (Bacon, Harb, and Precup 2016), which is also based on the actor-critic architecture and uses the two-layer optimization of Bellman targets. For all the agents trained with A2C, we use the same hyper-parameters used in expert training.

We list the relative performance of routine-augmented A2C over A2C in Figure 2, which indicates that our approach achieves the same or better performance in 25 out of 33 games. This fact indicates that the routines discovered from the demonstration can effectively enhance the exploration of reinforcement learning. The training curves of comparison on Atari games are shown in Figure 3. Our approach outperforms baselines on most of the games. We notice that ComPILE usually deteriorates the baseline of A2C. The first reason for this is that ComPILE requires many demonstrations from a family of tasks to train the sub-task policies and termination conditions. When only a single demonstration of a task is given, those parametric policies and conditions cannot generalize to unseen states. The OptionCritic does not use the demonstration but uses more parameters to model the option policy, intra-option policy, and termination conditions. Therefore, it achieves relatively limited performance gain over A2C. In contrast, our proposed approach successfully discovers effective routines from a single demonstration, which further generalizes to states that are not seen from the demonstration.

Scalability of RAPL

We study the performance of RAPL when scale to more or longer routines in comparison with a naïve baseline MacroAction (Durugkar et al. 2016). MacroAction appends routines into the agent's action space and adopts an off-the-shelf A2C algorithm to train the controller. To ensure fairness, we adopt the same routines discovered from the demonstration for MacroAction.

The results are shown in Figure 5. Our approach performs better on more and longer routines. The first reason is that MacroAction does not reuse the experience from routine execution to update the value function at the primitive-level as in Eq. 6. So when using longer routines, the value function's bootstrap involves too many primitive steps (they do not interrupt during the execution of routines (Sutton, Precup, and Singh 1999)). Therefore the value estimation of middle states during execution is less accurate, leading to inferior performance. When using more routines, RAPL-A2C can efficiently share experiences of routines to primitives, so more routines deteriorate the performance to a less extent. Furthermore, it does not take care of the temporal discount relationship when the execution of routines triggers temporal abstraction. For example, it defines the reward of a routine execution to be the sum of rewards during its execution, which contradicts to Eq. 5.

Effectiveness of Routine Discovery

We compare the full model (Full) to the following ablated versions to validate routine discovery effectiveness. Each model is tested on eight Atari games listed in Figure 3. (1) Random Routines (RR), where each routine is generated randomly. (2) The proposal by Enumeration (PbE) where we enumerate all the possible combinations of primitive actions to form routine candidates. (3) Random Fetch (RF) where we random fetch sub-sequences from the demonstration to form routines. (4) Imperfect Demonstration (ID) where the expert is only trained with 1 million steps. (5) Repeat (RP), where the routines are the repetition of most frequently used atomic actions in the demonstration (Sharma, Lakshminarayanan, and Ravindran 2017).

Despite the specified ablated component, other details are the same as the full model (including the number and the length of each routine). We run each model for five random seeds and report both the mean and standard deviation in Figure 6. We observe that ablating any of the components would harm the performance of discovered routines. Random Routines and Proposal by Enumeration perform worst among all the models because they do not leverage the demonstration's information and only select routines by the heuristic. The inferior performance of Random Fetch suggests it is beneficial to propose routines via Sequitur. Our model also outperforms simply repetition. We can also find that our approach is robust to imperfect demonstrations because useful skills exist in the imperfect experts.

Generalization of Routines

We conduct various experiments on CoinRun to validate the generalization ability of RAPL. We train two agents by both A2C and RAPL-A2C on the same 100 easy levels. Then we test them on 100 unseen easy levels to test the generalization ability to unseen levels. After that, we test both agents on 100 hard levels to test the generalization ability across difficulties.

The results are shown in Figure 4. Both A2C and RAPL-A2C fit well in the training levels. Notably, we find RAPL-A2C improves generalization ability. On the one hand, we observe that the discovered routines can successfully generalize to unseen levels. On the other hand, discovering useful skills from relatively simple domains might also promote policy learning in unseen hard domains. These facts indicate that routines may alleviate over-fitting problems of deep neural networks.

Visualization of Trained Agents. We provide a visualization of two trained agents in the Supplementary. The discovered routines represent the ability to jump far and high, helping the agent to overcome obstacles. Besides, the policy trained by plain A2C is pretty noisy due to the sparse reward in CoinRun (the agent only gets positive rewards at the end of each episode). Routines regularize the policy towards the optimal policy, which contributes to the improvement in generalization. Finally, we observe that adopting routines can benefit the interpretability of policy since routines are higher-level actions that are easier for a human to understand.

Conclusion

In this paper, we have presented *routine-augmented policy learning* (RAPL) to discover a set of routines from a single demonstration and augment policy learning via the discovered routines. From extensive experiments on Atari, we found that routines can enhance imitation learning by learning at multiple temporal scales, and routines can promote exploration in reinforcement learning. Besides, from experiments on CoinRun, we found that the discovered routines can generalize to unseen levels and harder domains. We hope that our proposed approach can inspire further work to extend RAPL to continuous action domains. Moreover, discovering routines with rich semantic information would be a promising future direction.

Acknowledgements

This work was supported in part by the Center for Brains, Minds and Machines (CBMM, NSF STC award CCF-1231216), ONR MURI N00014-16-1-2007, MIT-IBM Watson AI Lab, and MERL.

References

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57(5): 469–483.

Bacon, P.; Harb, J.; and Precup, D. 2016. The Option-Critic Architecture. *CoRR* abs/1609.05140.

Barto, A. G.; and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* 13(4): 341–379.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2012. The Arcade Learning Environment: An Evaluation Platform for General Agents. *CoRR* .

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. *CoRR* abs/1606.01540.

Chang, Y.-H.; Chang, K.-Y.; Kuo, H.; and Lee, C.-Y. 2019. Construction of Macro Actions for Deep Reinforcement Learning. *arXiv preprint arXiv:1908.01478* .

Christodoulou, P.; Lange, R. T.; Shafti, A.; and Faisal, A. A. 2019. Reinforcement Learning with Structured Hierarchical Grammar Representations of Actions.

Cobbe, K.; Klimov, O.; Hesse, C.; Kim, T.; and Schulman, J. 2018. Quantifying Generalization in Reinforcement Learning. *CoRR* abs/1812.02341.

Dawson, C.; and Siklossy, L. 1977. The Role of Preprocessing in Problem Solving Systems: "An Ounce of Reflection is Worth a Pound of Backtracking". In *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'77, 465–471.

Duan, Y.; Andrychowicz, M.; Stadie, B.; Ho, O. J.; Schneider, J.; Sutskever, I.; Abbeel, P.; and Zaremba, W. 2017. One-shot imitation learning. In *Advances in neural information processing systems*, 1087–1098.

Durugkar, I. P.; Rosenbaum, C.; Dernbach, S.; and Mahadevan, S. 2016. Deep Reinforcement Learning With Macro-Actions. *CoRR* .

Esmaili, N.; Sammut, C.; and Shirazi, G. M. 1995. Behavioural cloning in control of a dynamic system. In *1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century*, volume 3, 2904–2909 vol.3.

Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; Legg, S.; and Kavukcuoglu, K. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. *CoRR* abs/1802.01561. URL http://arxiv.org/abs/1802.01561.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and Executing Generalized Robot Plans. *Artif. Intell.* 3(1): 251–288.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3-4): 189–208.

Fox, R.; Krishnan, S.; Stoica, I.; and Goldberg, K. 2017. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294* .

Garcia, F. M.; da Silva, B. C.; and Thomas, P. S. 2019. A Compression-Inspired Framework for Macro Discovery. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19.

Hester, T.; Vecerik, M.; Pietquin, O.; Lanctot, M.; Schaul, T.; Piot, B.; Horgan, D.; Quan, J.; Sendonaris, A.; Osband, I.; et al. 2018. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Ho, J.; and Ermon, S. 2016. Generative Adversarial Imitation Learning. *CoRR* abs/1606.03476. URL http://arxiv.org/abs/1606.03476.

Kang, B.; Jie, Z.; and Feng, J. 2018. Policy Optimization with Demonstrations. In *Proceedings of the 35th International Conference on Machine Learning*, 2469–2478.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In Bengio, Y.; and LeCun, Y., eds., *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. URL http://arxiv.org/abs/1412.6980.

Kipf, T.; Li, Y.; Dai, H.; Zambaldi, V.; Sanchez-Gonzalez, A.; Grefenstette, E.; Kohli, P.; and Battaglia, P. 2019. CompILE: Compositional Imitation Learning and Execution. In *International Conference on Machine Learning (ICML)*.

Krishnan, S.; Fox, R.; Stoica, I.; and Goldberg, K. 2017. Ddco: Discovery of deep continuous options for robot learning from demonstrations. *arXiv preprint arXiv:1710.05421* .

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, 3675–3683.

Le, H. M.; Jiang, N.; Agarwal, A.; Dudík, M.; Yue, Y.; and III, H. D. 2018. Hierarchical Imitation and Reinforcement Learning. *CoRR* abs/1803.00590. URL http://arxiv.org/abs/1803.00590.

Machado, M. C.; Bellemare, M. G.; and Bowling, M. 2017. A Laplacian Framework for Option Discovery in Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, 2295–2304. JMLR.org.

McGovern, A.; and Sutton, R. S. 1998. Macro-actions in reinforcement learning: An empirical analysis. *Computer Science Department Faculty Publication Series* 15.

Miller, F. P.; Vandome, A. F.; and McBrewster, J. 2009. *Levenshtein Distance: Information Theory, Computer Science, String (Computer Science), String Metric, Damerau?Levenshtein Distance, Spell Checker, Hamming Distance*. Alpha Press. ISBN 6130216904.

Minton, S. 1985. Selectively Generalizing Plans for Problem-Solving. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'85.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T. P.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous Methods for Deep Reinforcement Learning. *CoRR* abs/1602.01783.

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

Nair, A.; McGrew, B.; Andrychowicz, M.; Zaremba, W.; and Abbeel, P. 2018. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 6292–6299. IEEE.

Nevill-Manning, C. G.; and Witten, I. H. 1997. Identifying Hierarchical Structure in Sequences: A linear-time algorithm. *J. Artif. Intell. Res.* 7(cs.AI/9709102): 67–82.

Pathak, D.; Agrawal, P.; Efros, A. A.; and Darrell, T. 2017. Curiosity-driven Exploration by Self-supervised Prediction. *CoRR* abs/1705.05363.

Pickett, M.; and Barto, A. G. 2002. PolicyBlocks: An Algorithm for Creating Useful Macro-Actions in Reinforcement Learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, 506–513. Morgan Kaufmann.

Randlov, J. 1999. Learning Macro-Actions in Reinforcement Learning. In Kearns, M. J.; Solla, S. A.; and Cohn, D. A., eds., *Advances in Neural Information Processing Systems 11*, 1045–1051. MIT Press.

Reddy, S.; Dragan, A. D.; and Levine, S. 2019. SQIL: Imitation Learning via Regularized Behavioral Cloning URL http://arxiv.org/abs/1905.11108.

Riemer, M.; Liu, M.; and Tesauro, G. 2018. Learning Abstract Options. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 31*, 10424–10434. Curran Associates, Inc.

Ross, S.; Gordon, G. J.; and Bagnell, J. A. 2010. No-Regret Reductions for Imitation Learning and Structured Prediction. *CoRR* abs/1011.0686. URL http://arxiv.org/abs/1011.0686.

Salimans, T.; and Chen, R. 2018. Learning Montezuma's Revenge from a Single Demonstration. *arXiv preprint arXiv:1812.03381* .

Shankar, T.; Tulsiani, S.; Pinto, L.; and Gupta, A. 2020. Discovering Motor Programs by Recomposing Demonstrations. In *International Conference on Learning Representations*.

Sharma, S.; Lakshminarayanan, A. S.; and Ravindran, B. 2017. Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint arXiv:1702.06054* .

Stolle, M.; and Precup, D. 2002. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, 212–223. Springer.

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

Thrun, S. B. 1992. Efficient exploration in reinforcement learning .

Tsividis, P. A.; Pouncy, T.; Xu, J. L.; Tenenbaum, J. B.; and Gershman, S. J. 2017. Human learning in Atari. In *2017 AAAI Spring Symposium Series*.

Vecerik, M.; Hester, T.; Scholz, J.; Wang, F.; Pietquin, O.; Piot, B.; Heess, N.; Rothörl, T.; Lampe, T.; and Riedmiller, M. 2017. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817* .

Wingate, D.; Goodman, N. D.; Roy, D. M.; Kaelbling, L. P.; and Tenenbaum, J. B. 2011. Bayesian policy search with policy priors. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Supplementary to Augmenting Policy Learning with Routines Discovered from a Single Demonstration

Anonymous Submission

Algorithm 1 RoutineDiscovery

```
Input: demonstrated action trajectory $\iota = (a_1, a_2, ..., a_n)$, routine candidate select number $K$, similarity threshold $\alpha$, length balancing factor $\lambda^{length}$
Input the first action $a_1$ to create the production $S \rightarrow a_1$
# Routine Proposal by Sequitur
for every action $a_i \in \iota$ do
    Try to match existing rules to generate $(a_1, ..., a_i)$
    Create a new rule if a new diagram of actions appears and repeats twice
    Remove a rule if it is used only once
end for
Initialize a collection of routine candidates $\mathcal{C} \leftarrow \emptyset$
for each introduced non-terminal variable $\rho$ do
    Transfer $\rho$ to a sequence of action primitives by appling the rules
    Append $\rho$ to $\mathcal{C}$
end for
# Routine Selection
Initialize a routine library $\mathcal{L} \leftarrow \emptyset$
for each routine candidate $\rho \in \mathcal{C}$ do
    $f(\rho)$ = the frequency of $\rho$ in $\iota$
    Calculate the score of $\rho$ by $f(\rho) + \lambda^{length}|\rho|$
    Put $\rho$ into $\mathcal{L}$
    if there are routines in $\mathcal{L}$ whose Levenshtein distance is smaller than $\alpha$ then
        Leave one with the highest score
    end if
end for
Remove routines with lowest scores so that $|\mathcal{L}| = K$
return $\mathcal{L}$
```

Pseudo-codes

In Algorithm 1, we present the procedure of routine discovery from a single demonstration.

Experimental Details

Hyper-parameters. We select hyper-parameters by coarsely running experiments on eight Atari games (the same games used in the main text). For each experiment, we repeat for five agents and report both the mean and standard error in Figure 1. We can observe that RAPL-A2C works well for a wide range of hyper-parameters.

Implementation. We thank the author of **?** for providing the implementation of BC, GAIL, and SQIL. We adopt the public code under Pytorch to implement A2C (**?**). The implementation of the ComPILE model (**?**) originates from its official release, despite that we train the controller via A2C.

Additional Experiment Results

We provide training curves for the imitation learning experiments in Figure 2. Moreover, we provide detailed scores for imitation learning on all the Atari games in Table 1. We also provide detailed scores for reinforcement learning on all the Atari games in Table 2.

Extended Review

Macro-actions Macro-actions are temporally extended actions that are widely studied in various domains. Sutton (**??**) refers macro-actions (or called *options*) to policies with termination conditions. He has empirically justified that the macro actions can sometimes boost exploration and value propagation. However, in the STRIPS system (**?**), a macro action is a sequence of pre-defined operators that contribute to problem-solving. **?** instead uses the notion of *macro-operators* built upon primitive operators to efficiently solve problems. In our work, we specify a *routine* to be a sequence of primitive actions, which can be regarded as one specific case of macro actions.

Macro-action Discovery In the field of theorem proving, MACLEARN (**?**) conducts searching to find new macros so that the problems can be solved faster or more economically. Besides, Macro-FF (**?**) improves AI planning by abstracting macro actions from the problem's domain structure. The abstraction is achieved by a clustering procedure over the problem's graph. These methods are not designed for general MDPs. **?** proposes to learn macro actions by finding frequently visited states. Their method is only suitable for environments with low-level states such as the grid world. **?** implicitly learns macro actions by modeling the transition

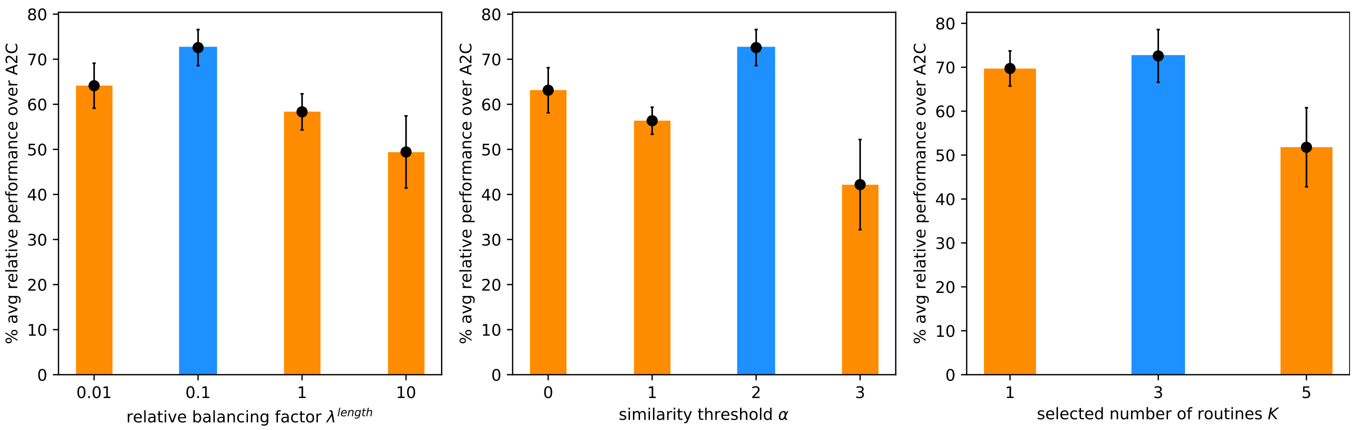

Figure 1: Ablation study on various hyper-parameters.

from one action to another. **?** proposes to acquire macro-actions with tree structures in Reinforcement Learning. They assume that the states are discrete (e.g., in a maze); thus, their methods cannot generalize to high-dimension tasks. **?** proposes to abstract macro actions during the learning of an RL agent. However, as indicated by their experiments, the frequently changed action space would make the training unstable. The concurrent work **?** utilizes a genetic algorithm to construct macro actions for deep reinforcement learning. Nevertheless, their methods require training RL agents multiple times to find a set of good macro actions, which is very time-consuming.

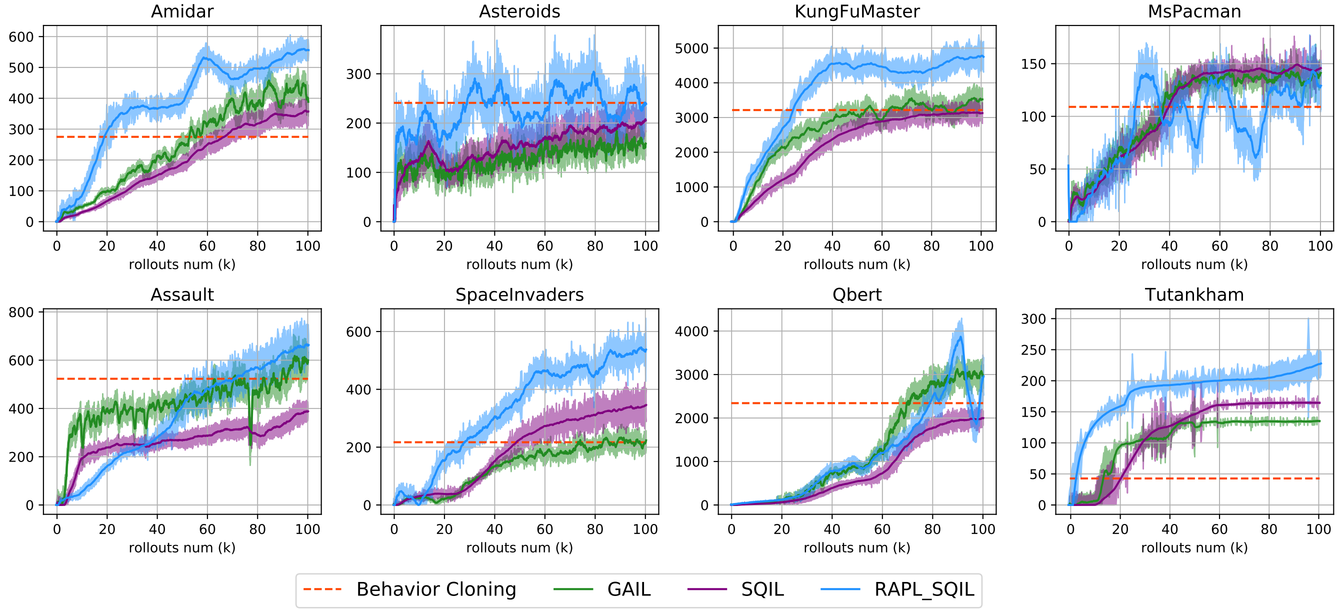

Figure 2: Training curves on Atari games in comparison with several imitation learning baselines. We plot both the mean and standard deviation in those curves across five agents with random seeds.

Table 1: Results to compare RAPL-SQIL with several imitation learning baselines on Atari games. Each number in the table represents average score and standard over five random seeds. Denote the scores of RAPL-SQIL and human are S_R and S_H accordingly. The **Normalized %** is calculated by $S_R/S_H * 100$ (**?**). The human score of Phoenix is tested by us, which is 6002.0.

	BC	GAIL	SQIL	**RAPL-SQIL**	**Normalized %**
Freeway	4.1 ± 1.2	12.4 ± 2.9	5.9 ± 1.9	13.1 ± 3.7	44.3 ± 1.3
Assault	523.8 ± 32.7	631.7 ± 12.3	529.1 ± 30.7	677.3 ± 29.8	45.3 ± 2.0
WizardOfWor	1239.4 ± 111.7	1980.7 ± 183.2	2140.5 ± 116.1	2212.6 ± 113.9	46.5 ± 2.4
UpNDown	1331.3 ± 12.7	1398.1 ± 17.5	3012.3 ± 21.3	428.0 ± 11.9	4.7 ± 0.0
Amidar	275.3 ± 115.9	416.9 ± 124.6	441.8 ± 127.5	559.1 ± 131.1	33.4 ± 7.8
Hero	321.7 ± 12.1	459.2 ± 32.8	443.6 ± 49.4	639.0 ± 67.2	2.5 ± 0.1
FishingDerby	3.1 ± 1.3	0.4 ± 2.0	1.3 ± 2.0	2.4 ± 0.8	43.6 ± 14.5
Alien	324.3 ± 126.8	305.6 ± 154.3	523.1 ± 112.8	450.0 ± 179.1	6.5 ± 2.6
Bowling	81.5 ± 8.7	129.3 ± 9.1	192.4 ± 4.6	32.1 ± 3.4	20.7 ± 2.2
BankHeist	42.1 ± 2.8	104.2 ± 12.3	106.3 ± 14.2	421.1 ± 28.4	57.3 ± 3.9
Atlantis	1920.7 ± 338.1	3321.5 ± 274.9	2043.5 ± 521.7	2978.6 ± 792.2	10.3 ± 0.0
Qbert	2342.5 ± 421.3	2939.6 ± 319.8	2539.1 ± 260.4	3012.6 ± 331.4	22.4 ± 2.4
Jamesbond	42.2 ± 18.5	109.6 ± 17.2	302.7 ± 21.8	375.1 ± 32.8	92.2 ± 8.0
Asteroids	241.9 ± 11.8	180.1 ± 22.7	243.6 ± 44.2	260.7 ± 56.9	2.0 ± 0.0
Kangaroo	1211.3 ± 19.4	1294.5 ± 21.8	2043.6 ± 42.8	2412.9 ± 44.3	79.5 ± 1.4
KungFuMaster	3214.9 ± 391.7	3572.1 ± 320.8	4053.2 ± 412.4	4512.9 ± 188.2	19.8 ± 0.8
NameThisGame	1023.2 ± 314.8	2031.3 ± 422.5	2065.7 ± 191.2	2104.7 ± 59.8	51.6 ± 1.5
CrazyClimber	3321.6 ± 222.9	1324.9 ± 533.4	4212.2 ± 121.8	4231.7 ± 317.9	12.0 ± 0.9
MsPacman	4012.9 ± 848.7	4209.1 ± 531.9	5378.9 ± 634.5	5469.2 ± 928.6	34.9 ± 5.8
Centipede	2011.3 ± 117.2	2310.4 ± 224.1	3032.4 ± 98.0	3139.5 ± 122.3	26.2 ± 1.0
SpaceInvaders	421.8± 41.2	590.7 ± 99.0	1232.3 ± 37.8	1420.4 ± 253.9	86.0 ± 15.3
Tutankham	42.7 ± 5.2	134.1± 6.3	203.5 ± 29.5	241.8± 25.7	144.3 ± 15.3
ChopperCommand	201.2 ± 37.1	667.0 ± 45.9	332.3 ± 49.1	720.0 ± 85.6	7.3 ± 0.9
TimePilot	592.8 ± 22.4	802.1± 33.5	1321.3± 44.8	1349.5± 52.0	22.8 ± 0.8
Enduro	0.0 ± 0.0	3.3 ± 1.2	2.1± 1.0	13.2 ± 3.5	4.3 ± 0.0
Gopher	1231.3± 41.8	1532.8± 53.1	1432.5± 48.3	1952.7± 57.9	84.1 ± 2.5
Frostbite	431.6± 52.1	512.8± 83.1	578.6± 61.9	642.1± 72.8	14.8 ± 1.7
Krull	1391.7±114.3	1785.0±152.9	653.9±121.4	2325.0±223.8	97.1 ± 9.3
BeamRider	512.5±41.2	1021.8±89.6	330.3±42.6	1258.9±251.3	21.8 ± 4.3
Breakout	3.2± 2.1	8.1±1.0	5.3±0.4	10.4±4.1	32.7 ± 0.0
Asterix	1324.1± 122.3	2635.0± 224.8	2178.2±89.5	1210.0±82.5	14.2 ± 1.0
Phoenix	504.4±63.1	617.2±44.8	201.9±64.8	391.5±73.1	6.5 ± 1.2
DemonAttack	664.8± 78.1	1794.2±91.3	401.4±144.8	2445.1±129.4	71.9 ± 3.8

Table 2: Results to compare RAPL-A2C with several reinforcement learning baselines on Atari games. Each number in the table represents average score and standard over five random seeds.

	ComPILE	OptionCritic	A2C	**RAPL-A2C**
Freeway	4.2±2.6	11.7±1.2	10±1.6	22.1±2.4
Assault	962.1±37.9	1329.7±304.9	2020±245.4	4032±148.7
WizardOfWor	687.3±64.7	779.7±146.1	787.2±34.7	1470±87.2
UpNDown	14211.7±2241.7	15874.9±622.9	12722.4±982.1	23147.5±261.8
Amidar	501.2±4.4	531.7±6.8	132±3.2	231.2±12.8
Hero	5964.7±1001.6	6642.6±287.8	9716.1±459	14970.9±1020.7
FishingDerby	-44.2±3.6	-55.2±0.7	-66±0.7	-33.5±0.3
Alien	444.2±19.1	769.4±26.7	721.3±44.8	1067.8±42.5
Bowling	13.4±2.9	21.6±2.1	24±0.7	34±1.5
BankHeist	643.8±71.4	883.5±29.8	584.6±36.4	822.1±2.7
Atlantis	129930.5±24009.6	224449.3±8131.6	197997.2±3370.6	269010.1±1660.2
Qbert	6729.4±1200	7864.1±475.8	12942.5±1646.2	16527.4±261.8
Jamesbond	134.5±19.9	122.3±1.6	199±20.7	263.7±24.3
Asteroids	735.1±178.3	2412.9±62.2	1535.4±88.4	2560.2±8.6
Kangaroo	149.6±26.3	220.5±19	221.8±2.5	290±0.2
KungFuMaster	12049.4±406.1	28741.2±467.7	24939±4297.5	47218.8±1775.8
NameThisGame	3305.3±162.7	4621.8±498.2	4073.8±375.5	5262.6±156.2
CrazyClimber	64319.3±6147.1	55691.4±10913.6	88588.6±8677.5	113559.3±10043.1
MsPacman	1895±388.5	1886.5±260	2733.5±198.2	4422±62.4
Centipede	1322.6±142.4	2682.4±11.1	2542.6±129.2	3132±206.8
SpaceInvaders	1894.6±79.6	2583.9±349.2	3312.8±205.3	4592±269.7
Tutankham	532.3±33.3	512.5±123	779.4±79.1	1422.5±32.7
ChopperCommand	443.1±36.6	532.5±41.9	888.6±54.4	911.9±66.5
TimePilot	1354.9±16.5	3442.6±163.6	3453.6±289.5	3490.7±9.1
Enduro	0±0.6	0±0.8	1.9±0.1	1.8±0.7
Gopher	644.2±17.3	553.6±73.7	966±64.8	928.1±19.3
Frostbite	44.2±22.5	236.8±18.3	256.3±2.8	248±15.4
Krull	5590.9±554.9	6689.7±437	6791.3±565.4	6041.7±166.8
BeamRider	2124.1±121.5	552.1±130.1	2191.2±100.4	1745.4±67.3
Breakout	62.7±10.9	144.2±7.9	182.9±7.5	139.8±7.9
Asterix	3153.1±150.8	1235.8±76	2463.1±140.7	1823.9±92.6
Phoenix	5521.3±112.9	6642.9±274.1	6742.5±467.3	5009±158
DemonAttack	3352.1±207	5546.8±274.7	5957.6±214.4	4121±93.9