A Compression-Inspired Framework for Macro Discovery

Francisco M. Garcia  
University of Massachusetts  
Amherst, Massachusetts, USA  
fmgarcia@cs.umass.edu

Bruno C. da Silva  
Federal University of Rio Grande do Sul  
Porto Alegre, Rio Grande, Brazil  
bsilva@inf.ufrgs.br

Philip S. Thomas  
University of Massachusetts  
Amherst, Massachusetts, Brazil  
pthomas@cs.umass.edu

ABSTRACT

We consider the problem of how a reinforcement learning agent, tasked with solving a set of related Markov decision processes, can use knowledge acquired early on in its lifetime to improve its ability to more rapidly solve novel tasks. We propose a three-step framework that generates a diverse set of macros that lead to high rewards when solving a set of related tasks. Our experiments show that augmenting the original action-set of the agent with the identified macros allows it to more rapidly learn optimal policies in novel MDPs.

KEYWORDS

Reinforcement Learning; Hierarchical RL; Exploration

ACM Reference Format:


1 INTRODUCTION

One of the key aspects of human learning is our ability to construct building blocks upon which we can learn new skills. Humans generally bootstrap higher-level skills acquired early on in their lives to solve new problems. For example, a child learning to run does not need to re-learn the “balance” skill to stand in two feet as part of learning a “run” skill. Instead, the child uses previously acquired skills to more efficiently explore the consequences of his actions when facing novel tasks. In the Reinforcement Learning (RL) literature, higher-level actions are sometimes called options or macros [6, 8]. They introduce a bias in the behavior of the agent thereby making it non-differentiable and difficult to optimize. We posit that compression techniques provide a means to identify macros representing recurring behaviors in optimal policies for problems in the class, which therefore are good candidates for composing $M^*$. Full details and derivations of this framework can be found at https://arxiv.org/abs/1711.09048.

2 PROBLEM STATEMENT

We consider the setting where an agent is required to solve a set of tasks $c \in C$, where $C$ is a given problem class, and assume that when solving a particular task, it can interact with it for $I$ episodes. After the agent has trained on a subset $C_{\text{train}} \subset C$ of tasks, we are interested in identifying a set of macros to be used for improving learning in the set of remaining tasks $C_{\text{test}} \subset C$.

We define the performance of a set of macros $M$ in a particular task $c \in C$ to be $p(M, c) = E\left[\frac{1}{I} \sum_{i=1}^{I} \sum_{t=0}^{T} y^t R_t | M, c, \pi_0^i\right]$, where $R_t$ is the reward at time step $t$ during the $i^{th}$ episode. This quantity expresses the expected average return an agent gets over $I$ episodes on a task $c$ using an extended action set $\mathcal{A}_M = \mathcal{A} \cup M$ (an action set composed of primitives and macros). In other words, the performance of a set of macros is defined by how much the performance of an agent is improved during training by augmenting the agent’s original action set with a given set of macros $M$. Our goal is to find one (of possibly many) optimal set of macros $M^*$ for $C$ such that $M^* \in \arg \max_M \left\{ \frac{1}{I} \sum_{c \in C_{\text{test}}} p(M, c) \right\}$. Unfortunately, the domain of this objective function is discrete, making it non-differentiable and difficult to optimize. We posit that compression techniques provide a means to identify macros representing recurring behaviors in optimal policies for problems in the class, which therefore are good candidates for composing $M^*$. Full details and derivations of this framework can be found at https://arxiv.org/abs/1711.09048.

3 A HEURISTIC FOR APPROXIMATING $M^*$

The proposed framework is summarized in the diagram in Figure 1. The agent first trains on a set of tasks $C_{\text{train}} \subset C'$, thereby acquiring an optimal policy $\pi_0^*$. Once these samples have been obtained, our framework generates a set of macros $M'$ as an approximation to $M^*$ via the following 3-step process:

![Figure 1: Diagram depicting proposed framework.](image-url)
**Macro Generation:** A trajectory \( r \) of length \( l \) is the sequence of states, actions, and rewards experienced by an agent following a given policy: \( r = \{s_0, a_0, r_0, \ldots, s_{l-1}, a_{l-1}, r_{l-1}\} \). We define an action-trajectory \( r_a = \{a_0, a_1, \ldots, a_{l-1}\} \) as the sequence of actions in \( r \). Our agent is first trained on a set of training tasks and generates a set of action-trajectories by sampling them from the learned optimal policies. To generate macros, we consider each action-trajectory akin to a message we wish to compress and the set of primitive actions, \( \mathcal{A} \), analogous to the symbols in the initial alphabet used for compression. In this work we use the LZW compression algorithm [10]. Algorithm 1 shows our adaptation to encode action-trajectories as macros.

**Algorithm 1 LZW - macro codebook generation**

1. Initialize alphabet \( \Sigma = \mathcal{A} \)
2. macro \( m = () \)
3. for each action-trajectory \( r_a \) do
4. for each action \( a \) in \( r_a \) do
5. \( m = m + a \)
6. if \( m \notin \Sigma \) then
7. \( \Sigma = \Sigma \cup \{m\} \)
8. \( m = () \)
9. return \( \Sigma \)

**Macro Evaluation:** At this stage we have generated a possibly large set of candidate macros, \( \mathcal{A} \), but we do not know how useful they are in helping the agent to quickly solve new tasks. To do this, we measure the utility of a macro \( m \) w.r.t \( \mathcal{C} \) by means of a U-function defined as \( U_m^\mathcal{C}(m) = \mathbb{E}[Q^\mathcal{C}_m(s, m)] \), where the expectation is over tasks in \( \mathcal{C} \) and states \( S \) sampled from the on-policy distribution \([7]\). In other words, the utility of a macro \( m \) is the expected Q-value of \( m \) over all states in the problem class. A benefit of this approach is that given the Q-values of primitives, the Q-value of a macro can be computed efficiently, in closed-form, as:

\[
Q_m^\mathcal{C}(s, m) = \sum_{i=0}^{l_m} \Pi_{i=0}^{l_m} P(s, m_i, s_{i+1}) = \sum_{i=0}^{l_m} \Pi_{i=0}^{l_m} P(s, m_i, s_{i+1})
\]

**Macro Selection:** The utility allows us to assess which macros may lead to higher rewards. However, we must also consider that if too many macros are added to \( \mathcal{A} \), the agent’s action-set may get too large, thereby hindering learning. To address this issue, we select macros that not only have a high utility, but that are diverse from each other. Let \( S_t \) be a random variable denoting the state where \( m \) is executed and \( S_{t+l_m} \) the state where \( m \) finishes execution. Furthermore, let \( S' = d(S_{t+l_m}, S_t) \) be a random variable denoting the change in state from executing a macro \( m \), where \( d \) is a function measuring the change in state, and let \( p_m \) be a probability distribution over \( S' \) for \( m \). For two macros \( m_1 \) and \( m_2 \), we define the distance between them to be the KL divergence between \( p_{m_1} \) and \( p_{m_2} \):

\[
D_{KL}(p_{m_1} \parallel p_{m_2}) = - \sum_{S'} p_{m_1}(S') \log \left( \frac{p_{m_1}(S')}{p_{m_2}(S')} \right).
\]

The set \( \mathcal{A} \) is incrementally built by only including those macros (obtained from Algorithm 1) that have a minimum distance \( \delta \) to all other macros already included in the set. By selecting macros in descending order according to their U-value, the utility defines a preference criterion by which macros are selected. Pseudocode for framework is given in Algorithm 2.

**Algorithm 2 Macro discovery process**

1. **1. Macro Generation**
2. Learn optimal policy \( \pi^*_c \) for all \( c \in \mathcal{C}_{train} \).
3. Collect action-trajectories \( r_a \) from each \( \pi^*_c \) in task \( c \).
4. Generate macros \( M \) from all \( r_a \) by Algorithm 1
5. **2. Macro Evaluation**
6. Sort all \( m \in M \) by \( U_m^\mathcal{C}(m) \) in descending order.
7. **3. Macro Selection**
8. \( \mathcal{A}_{M'} = \mathcal{A} \)
9. for \( m \in M \) do
10. if \( \min_{m'} D_{KL}(p_m \parallel p_{m'}) > \delta \) for all \( m' \in \mathcal{A}_{M'} \) then
11. \( \mathcal{A}_{M'} = \mathcal{A}_{M'} \cup \{m\} \)

**4 EXPERIMENTAL RESULTS**

We compared the performance of primitives (P), primitive+macros (M), eigenoptions (E) [3] and the option-critic architecture (O) [1]. We evaluated our method in a maze navigation problem class where the environment dynamics are known and the true Q-values (for primitives) can be estimated accurately in tabular form. This allows us to accurately compute the U-value for any candidate macro. Figure 2 compares the learning performance of an agent using each technique with tabular Q-learning as the learning algorithm. We also extended these tests to two problems with large state spaces, where the transition functions and U-values had to be estimated by sampling: Animat [9] and Lunar Lander [2]. We tested our identified macros on 10 novel testing tasks and used DQN Mnih et al. [5] as a learning algorithm. The return of the policy learned after 1000 training episodes for the two problem classes is as follows. In Animat, P: \(-909.77 \pm 199.53\), M: \(-752.89 \pm 188.59\), E: \(-1,432.46 \pm 64.72\), O: \(-1.455.47 \pm 41.22\). In Lunar Lander, P: \(-314.03 \pm 44.09\), M: \(-298.89 \pm 28.99\), E: \(-266.43 \pm 5.22\), O: \(-265.51 \pm 7.42\).

**ACKNOWLEDGMENTS**

This work was partially supported by FAPERGS under grant no. 17/2551-000.
REFERENCES


