MACRO ACTION ENSEMBLE SEARCHING METHODOLOGY FOR DEEP REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we propose to improve the performance of deep reinforcement learning (DRL) methods by searching for a feasible macro action ensemble to augment the action space of an agent. A macro action ensemble is composed of multiple macro actions, which are typically defined as sequences of primitive actions. A well-defined macro action ensemble enables a DRL agent to achieve higher performance than conventional DRL methods on a variety of tasks. However, macro actions generated by previous approaches are either not necessarily promising, or limited to specific forms. As a result, in this study, we investigate a searching method to learn the macro action ensemble from the environment of interest. The proposed method is inspired by the concepts of neural architecture search techniques, which are capable of developing network architectures for different tasks. These search techniques, such as NASNet or MetaQNN, have been proven to generate high-performance neural network architectures in large search spaces. In order to search in large macro action ensemble spaces, we propose to embrace Deep Q-Learning to search the macro action ensemble space for a good ensemble. Our approach iteratively discovers new ensembles of macro actions with better performance on the learning task. The proposed method is able to search finite macro action ensemble spaces directly, that the other contemporary methods have yet to achieve. Our experimental results show that the scores attained by the policy trained with the discovered macro action ensemble outperforms those without it. Moreover, the policies using our macro action ensemble are more efficient in exploration and able to converge faster. We further perform a comprehensive set of ablative analyses to validate the proposed methodology.

1 INTRODUCTION

Deep reinforcement learning (DRL) aims at training an agent to learn a policy by interacting with the environment (Sutton & Barto, 2018). Since deep reinforcement learning (DRL) has shown groundbreaking results in a number of various challenging tasks (Levine et al., 2015; Mnih et al., 2015a; Silver et al., 2016; Mnih et al., 2013; Xu et al., 2018), researchers have developed various techniques and attempted different aspects of DRL to improve the performance of an agent (Mnih et al., 2016; Schulman et al., 2017; Wu et al., 2017). One of the effective techniques to improve the exploration and learning process is to reduce the dimensionality of the action space through macro action.

Previous studies defined macro actions as an open loop policy composed of a finite sequence of primitive actions. They are just like high-level movements(e.g walk, jump) to human begins. Macro actions to the RL agent is what high-level movements(e.g walk, jump) to human begins. Humans complete the complex tasks(e.g playing sports) through conducting a series of different high-level behaviours instead of deciding primitive motor actions at each time steps. With the proper previous learned movements or skills, humans can explore in a new task more efficiently and reduce the sample complexity. Similar to the human being scenario, previous work has shown that the well-defined macro actions can help to reduce the curse of dimensionality of action space, reduce sample complexity and improve the exploration[Roles of macro-action, Empirical Analysis, deep with macro]. However, the macro action will bias the behaviour of the agent to explore the environment, a lousy macro action may hinder the exploration of optimal solution and degrade the learning process dreadfully (McGovern et al., 1997; McGovern & Sutton, 1998). Therefore, generating useful and

effective macro actions become essential topic. Generally, the macro action generated by the previous method can be categorized into two types: 1) various lengths of repeated action. 2) combination of primitive macro action. The paper will focus on the second type of the macro action. Moreover, we hypothesize there exists cooperation relationship between macros, so instead of single macro action at a time, our work will generate the macro action ensemble composed of multiple macro action simultaneously.

The contributions of this paper can be summarized as follows:

- We define the proposed approach as a framework.
- We provide a definition of macro action ensemble space.
- We introduce an augmentation method for action spaces.
- Our method find ensemble instead of single macro action.
- Our method can produces all types of macro action.
- Our method finds macro action ensemble leading to a better performance compare to other baseline method.
- Our method do not need any human prior or domain knowledge.

The remainder of this paper is organized as follows. Section 2 surveys relavant previous works. Section 3 reviews the background material. Section 4 walks through the propose methodology as well as our implementation details. Section 5 presents our experimental results. Section 6 concludes.

2 Related Work

In this section, we review research works in the realm of temporally extended RL framework (Sutton et al., 1999). A few early attempts (McGovern et al., 1997; Randlov, 1999; Braylan et al., 2015) have demonstrated the effectiveness of frame skip, even though manually defined skip rate is required for each environment. Later on, researchers have advanced the above concept to automatically adjust the temporal scale for each state (Durugkar et al., 2016; Vezhnevets et al., 2016; Lakshminarayanan et al., 2017; Sharma et al., 2017). These methods allows an agent to repeat the same primitive action several times until the next decision cycle. However, the lack of diversity in the repeated actions also limits the ability of the agent for exploration. A few recent techniques are proposed to generate macro actions composed of different primitive actions (Botea et al., 2005; Coles & Coles, 2007; Heecheol et al., 2019). Nevertheless, these techniques rely on structural knowledge about planners or require expert domain knowledge. Researchers have also investigated approaches to construct macros based on frequent sequences of actions from the experience of an agent (Randlov, 1999; Durugkar et al., 2016; Dulac et al., 2013; Yoshikawa & Kurihara, 2006; Onda & Ozawa, 2009). Nonetheless, frequently used action sequences may not necessarily lead the agent to higher performance. Evolution-based strategies have also been investigated to search for useful macro actions (Newton et al., 2005; 2007; Chang et al., 2019). Unfortunately, they require the use of additional utility functions during the macro evaluation procedure. Other methods such as (Hauskrecht et al., 1998; Kulkarni et al., 2016; Bacon et al., 2016; Heess et al., 2016) employ sub-policies to interacts with the environment in a certain timesteps until the termination condition is met. However, these methods do not generate reusable macro actions, and belong to the hierarchical reinforcement learning (HRL) domain which is beyond our problem scope. As a consequence, the above works are either relied on human prior, built upon expert demonstrations, or are limited to only certain types of macro actions. To the best of our knowledge, none of them have attempted to search an ensemble of macro actions simultaneously.

3 Background

In this section, we start by briefly reviewing the formulations of DRL (Sutton et al., 1999; Sutton & Barto, 2018) and deep Q-network (DQN) (Mnih et al., 2015b) in Section 3.1 on which our proposed methodology for searching ensembles of macro actions is based. The definition of macro actions and the essential concepts of neural architecture search are presented in Sections 3.2 and 3.3, respectively.

3.1 DEEP REINFORCEMENT LEARNING

DRL formulation. DRL formulates the interaction of an agent with an environment $\mathcal{E}$ as a Markov Decision Process (MDP), defined as a 5-tuple $(\mathcal{S}, \mathcal{A}, p, \gamma, r)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the space of primitive actions, p the environment transition dynamics, $\gamma \in [0, 1]$ the discount factor, and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function. A DRL agent aims at learning an optimal policy π^*, so as to maximize the expected discounted return $V^\pi(s)$ by interacting with $\mathcal{E}$ in an interleaving pattern until a horizon timestep $\mathcal{H}$ is reached. At each timestep t, the agent perceives the current state $s_t \in \mathcal{S}$, chooses an action $a_t \in \mathcal{A}$ according to its policy π, receives a reward r_t from $\mathcal{E}$, and transitions to the next state s_{t+1}. The transition dynamics p is modeled as a state-transition probability function, denoted as $p^a_{ss'} = \mathbb{P}\{s_{t+1} = s' | s_t = s, a_t = a\}$. The formulation of $V^\pi(s)$ is offered in the appendices, Eq. 5.

Deep Q-network (DQN). DQN approximates the Q function by a neural network Q_θ parameterized by a set of parameters θ. The network is trained to minimize the temporal difference (TD) error $L(\theta) = \mathbb{E}\left[\left(r_t + \gamma \max_{a'} Q_{\theta^-}(s_{t+1}, a') - Q_\theta(s_t, a_t)\right)^2\right]$, where θ^- represents the parameters of the target network Q_{θ^-}. The parameters θ are updated iteratively using gradient descent, while the target network is fixed for a few number of iterations and only being updated by parameters θ periodically. Such a procedure has been validated to be able to stabilize the training process (Mnih et al., 2015b).

3.2 MACRO ACTION

Macro action. A macro action m (or simply "*macro*"), which can be represented as a finite sequence of primitive actions $m = (a_1, \cdots, a_k)$, can be selected as one of the actions by an DRL agent. Moreover, the set of macros form an enormous macro action space $\mathfrak{M}$, which is defined as $\mathfrak{M} = \mathcal{A}^+$.

3.3 NEURAL ARCHITECTURE SEARCH

Automatic design of neural network architecture has been demonstrated its ability to design state-of-art network architectures (Baker et al., 2017; Zoph & Le, 2017; Zoph et al., 2017; Pham et al., 2018; Cai et al., 2017; Liu et al., 2018; Real et al., 2017; 2018; Liu et al., 2017; Miikkulainen et al., 2017). The methods in the realm of the neural architecture search can be roughly divided into two category: Reinforcement Learning (Baker et al., 2017; Zoph & Le, 2017; Zoph et al., 2017; Pham et al., 2018; Cai et al., 2017), Evolutionary Algorithm (Liu et al., 2018; Real et al., 2017; 2018; Liu et al., 2017; Miikkulainen et al., 2017). In the RL based method, the action is taken to perform predefined operations to trim the components of the network architecture. The agent is usually referred as controller which controls the overall architecture. In the EV based method, searched is performed with mutations and re-combinations on the architecture components. Among the both category, Neural evolution has the advantage of creating arbitrary structure instead of only linear or tree-like core structure. However, the macro actions defined in our problem scope are only in the form of linear structure. The member actions in a macro are related with each other, so it make no sense to change it arbitrary. Hence we applying one of the RL based method(Baker et al., 2017) in our framework.

4 METHODOLOGY

In this section, we first provide the formal definition of a macro action ensemble and the macro action ensemble space, and reformulate the definitions of the value function in DRL. Then, we introduce the proposed macro action ensemble searching framework, and discuss the implementation details. Finally, we walks through the pseudocode of the framework and explain the training methodology. The essential notations used in this paper can be referred in Table 1 in our supplementary appendices.

4.1 FORMULATION OF MACRO ACTION ENSEMBLE FOR DEEP REINFORCEMENT LEARNING

Macro action ensemble. A macro action ensemble $\mathfrak{e}$ (or simply "*ensemble*") is defined as a set of macros $\mathfrak{e} = \{m_1, m_2, \ldots, m_\omega\}$, where ω is some non-negative integer. A macro in an ensemble can be selected atomically as one of the actions by an agent. The set of ensembles form a macro action

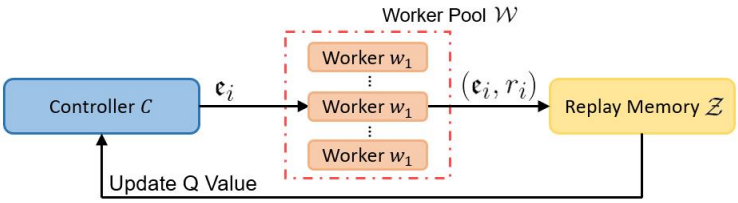


Figure 1: Overview of the proposed ensemble searching framework

ensemble space $\mathfrak{E}$, which can be represented as $\mathfrak{E} = \mathcal{P}(\mathfrak{M})$, where $\mathcal{P}(\mathfrak{M})$ stands for the power set of $\mathfrak{M}$. Please note that the empty ensemble $\{\}$ (i.e., the ensemble contains no macros) is contained in $\mathfrak{E}$.

Formulation for DRL. The environment concerned in this work is modeled as a special case of SMDP, and can also be represented as a 5-tuple $(\mathcal{S}, \mathcal{M}, p^m_{ss'}, r^m_s, \gamma)$, where $\mathcal{M}$ denotes the augmented action space. Please note that in this study, we relax the definition of $\mathcal{M}$ as the union of $\mathcal{A}$ and $\mathfrak{e}$ for all ensembles in $\mathfrak{E}$, represented as $\mathfrak{E} = \mathcal{P}(\mathfrak{M})$. The expected discounted returns the agent receives from each state s under policy ν, which is a mapping $\nu : \mathcal{S} \times \mathcal{M} \to [0, 1]$, can be denoted as $V^\nu(s)$. The optimal expected return from each state s under the optimal policy can be denoted as $V^*_\mathcal{M}(s)$. The expressions of $p^m_{ss'}, r^m_s, V^\nu(s)$ and $V^*_\mathcal{M}(s)$ can be represented as Eqs. 1, 2, 3 and 4, respectively.

$$p^m_{ss'} = \mathbb{P}\{s_{t+|m|} = s' | s_t = s, m_t = m\} \quad (1)$$

$$r^m_s = \mathbb{E}\left\{ \sum_{\tau=0}^{|m|-1} \gamma^\tau r_{t+\tau} \middle| s_t = s, m_t = m \right\} \quad (2)$$

$$V^\nu(s) = \sum_{m \in \mathcal{M}} \nu(s, m) \left[r^m_s + \gamma^{|m|} \sum_{s' \in \mathcal{S}} p^m_{ss'} V^\nu(s') \right] \quad (3)$$

$$V^*_\mathcal{M}(s) = max_{m \in \mathcal{M}} \left[r^m_s + \gamma^{|m|} \sum_{s' \in \mathcal{S}} p^m_{ss'} V^*_\mathcal{M}(s') \right] \quad (4)$$

4.2 The Proposed Macro Ensemble Searching Framework

In this section, we present the framework and the implementation details of the proposed methodology.

4.2.1 Overview of the Framework

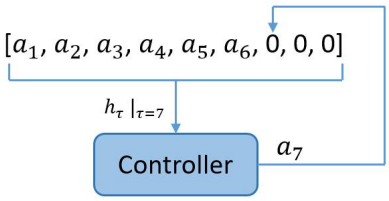
The objective of the proposed framework is to discover an $\mathfrak{e}$ such that the DRL agent is able to learn a ν on $\mathcal{M}$ and deliver a superior performance than the policies trained on $\mathcal{A}$. The proposed framework embraces an asynchronous distributed architecture, and is illustrated in Fig. 1. It consists of three main components: a controller $\mathcal{C}$, a worker pool $\mathcal{W}$ with n worker nodes $(w_1, w_2, \ldots, w_n)$ (where n is a configurable positive number), and a replay memory $\mathcal{Z}$. The controller periodically generates a new $\mathfrak{e}_i$ and assigns it to an available worker $w_i \in \mathcal{W}$, where $i \leq n$. The allocated worker then evaluates $\mathfrak{e}_i$ by training an DRL agent on $\mathcal{M}_i = (\mathcal{A} \cup \mathfrak{e}_i) \in \mathfrak{E}$ and stores the final reward r_i as well as the ensemble $\mathfrak{e}_i$ in $\mathcal{Z}$ in a special format called *controller state representation* h_τ, which is later explained in 4.2.2. Finally, the Q-value of $\mathcal{C}$ is then updated with the mini-batches sampled from $\mathcal{Z}$.

4.2.2 Controller

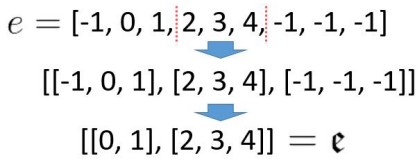
The controller is set to be a 1-layer fully-connected neural network with hidden layer size 512, activated by ReLU. The controller takes h_τ and generates a_τ until τ reaches T. With this controller mechanism, the value of T is not limited. However, the search space grows exponentially as T increase. Due to the constraint of computer power, we chose a relevantly small T as well as the controller network. And still, the experiment shows that our approach outperformed other macro action searching methodologies.

4.2.3 Evaluation Method for An Macro Action Ensemble

In our framework, ensembles are evaluated at the worker nodes. Algorithm 2 describes the evaluation procedure of $\mathfrak{e}$. The worker node w_i first reduces the received ensemble sequence e_i to $\mathfrak{e}_\text{i}$, as described in Section 4.2.2 and Fig. 2 (b). Then, it forms $\mathcal{M}$ based on $\mathcal{A}$ and $\mathfrak{e}_\text{i}$ (line 5), and trains a DRL agent to learn a ν using $\mathcal{M}$ in $\mathcal{E}$ (line 6). After the agent is trained for $\mathcal{H}$ timesteps, it is evaluated for



(a) **Controller sampling actions**. For each action, the controller takes a controller state representation h_τ and output the next action. After the action sequence e is filled up, the action sequence e than be transformed into ensemble $\mathfrak{e}$.



(b) **Ensemble sequence e to ensemble $\mathfrak{e}$**. Step one, divide ensemble sequence base on length of each macro which is a hyperparameter. Step two, eliminate every action of which value is -1.

Figure 2: Illustration of the workflow of the controller.

Algorithm 1 Ensemble Evaluation

1: **input:** Environment $\mathcal{E}$; ensemble sequence e_i
2: **output:** Clipped Reward $\hat{r_i}$
3: **function** EVALUATE ENSEMBLE($\mathcal{E}$, e_i)
4: Transform e_i into ensemble $\mathfrak{e}_i$
5: $\mathcal{M} \leftarrow \mathcal{A} \cup \mathfrak{e}_i,\ \mathfrak{e}_i \in \mathfrak{E}$
6: Learn a policy ν over $\mathcal{M}$ in $\mathcal{E}$ for $\mathcal{H}$ timesteps
7: Evaluate 100 episodes with the policy ν
8: Eliminate the highest 10 episodes rewards and lowest 10 episodes
9: **return** Average reward $\hat{r_i}$ of the rest 80 episodes
10: **end function**

another 100 episodes (line 7). Finally, Algorithm 1 returned a trimmed average r_i of the the middle 80% of the episode rewards, ignoring the top 10% and bottom 10% of the episode rewards (lines 8-9).

4.3 TRAINING METHODOLOGY OF THE CONTROLLER

The ensemble searching methodology employed by $\mathcal{C}$ is mainly based on the DQN algorithm (Mnih et al., 2013). As discussed in Section 4.2.2, the primary task of $\mathcal{C}$ is to construct e according to h_τ by sequentially generating a_τ in e. The overall training algorithm is presented in Algorithm 2. The training procedure can be modeled as episodes with fixed length T. For each episode, an empty list e_n is first initialized as a container (line 5). The controller $\mathcal{C}$ then iteratively predicts a_t based on the current contents of e_n until the termination time step T is reached (lines 7-9). Please note that a_τ is determined based on the epsilon-greedy algorithm with decaying ϵ (Sutton & Barto, 2018) (line 8). For each step τ, the selected a_τ is appended to e_n. When $\tau = \mathrm{T}$, the constructed e_n is evaluated by Algorithm 1, and a reward is received as r_τ (line 10). The *controller state representation* h_τ is then updated by $h_{\tau+1} \leftarrow \text{NEXT STATE}(\tau, h_\tau, a_\tau)$ (line 11), which is detailed in Algorithm 3 in the appendices. The tuple $(h_\tau, a_\tau, r_\tau, h_{\tau+1})$ collected at each step is stored into $\mathcal{Z}$ (line 12). As long as an episode is finished, the deep Q-network of $\mathcal{C}$ is updated using the mini-batches extracted from $\mathcal{Z}$ (line 13). The procedure is detailed in Algorithm 4. The training process proceeds until N is reached.

5 EXPERIMENTAL RESULTS

In this section, we present the experimental results and discuss their implications. We start by a brief introduction to our experimental setup in Section 5.1. Then, we investigate whether or not the proposed methodology is capable of discovering good ensembles for improving the training performance of an DRL agent in Section 5.3. We next demonstrate in Section 5.4 that the ensemble discovered by our method is more effective than a single macro action in terms of the scores received by the agent. We compare the performance of the ensembles constructed by our method against those constructed with action repeat or the most frequently used action sequences in Section 5.5. Finally, we provide an ablative analysis in Section 5.6 to examine if cooperative property exists among the macros in the discovered ensemble. Please note that we present the most representative results in this section, and strongly recommend the interested readers to refer to our appendices for further details.

Algorithm 2 Macro action ensemble searching algorithm based on DQN

Initialize the environment $\mathcal{E}$
Initialize the controller $\mathcal{C}$ with random weights
Initialize the replay memory $\mathcal{Z}$ to a null sequence [ ]
for $n = 1, 2, ..., N$ **do** ▷ the n^{th} generated ensemble
  Initialize the ensemble sequence e_n to a null sequence [ ]
  Initialize state h_1 to zero array
  for $\tau = 1, 2, ..., \mathrm{T}$ **do** ▷ the τ^{th} action in the ensemble sequence
    $a_\tau \leftarrow \begin{cases} \text{Select a random action } a_\tau & \text{with probability } \epsilon \\ \text{Action predicted by } \mathcal{C} \text{ using } h_\tau & \text{otherwise} \end{cases}$
    Append a_τ into e_n
    $r_\tau \leftarrow \begin{cases} \textsc{Evaluate Ensemble}(\mathcal{E}, e_n) & \text{if } \tau = \mathrm{T} \\ 0 & \text{if } \tau \neq \mathrm{T} \end{cases}$
    $h_{\tau+1} \leftarrow \textsc{Next State}(\tau, h_\tau, a_\tau)$
    Store transition $(h_\tau, a_\tau, r_\tau, h_{\tau+1})$ in $\mathcal{Z}$
  end for
  $\mathcal{C} \leftarrow \textsc{Update Q Value}(\mathcal{C}, \mathcal{Z})$
end for

5.1 Experimental Setup

We first present the environments used in our experiments, followed by a brief description of the baselines adopted for comparison purposes. For Sections 5.3 and 5.4, the ensembles are constructed by our proposed method. For Sections 5.5 Section 5.6 we compare and analyze the performance of the ensembles constructed in different ways. The environmental configurations, the hyper-parameters of Algorithms 1 and 2, and the hyper-parameters used during the training phase are tabularized in our appendices. Except for the training curves of $\mathcal{C}$, the rest of the curves presented in this section are generated based on five random seeds and drawn with 95% confidence interval as the shaded areas.

5.1.1 Environments

We employ the six representative games *Seaquest*, *BeamRider*, *SpaceInvaders*, *Enduro*, *KungFuMaster*, and *Q*bert* from *Atari 2600* (Bellemare et al., 2013) to evaluate our method. These games are selected based on the original DQN paper (Mnih et al., 2013) as well as one of our baseline method (Chang et al., 2019). Due to the limited space, we only present our comparisons and analyses for the former four games in this section, and leave the remainder of our results in the appendices.

5.1.2 Baselines

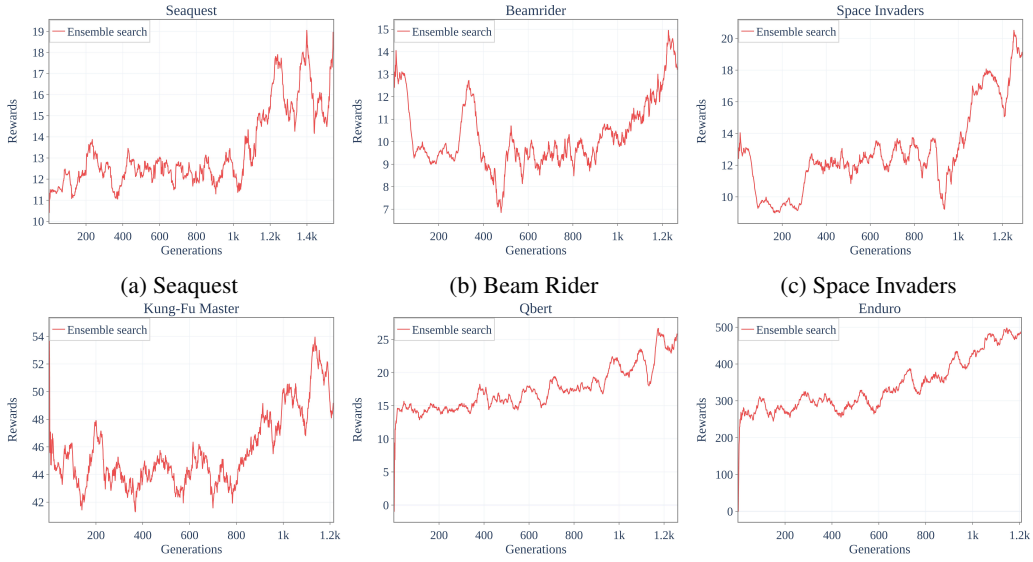
To evaluate the generated $\mathfrak{e}$, we select proximal policy optimization (PPO) (Schulman et al., 2017) for training the agents on specified $\mathcal{M}$, and compare our proposed method with the following baselines. Please note that the hyper-parameters of the baselines are provided in our supplementary appendices.

Primitive action. We compare the performance of the agents trained on $\mathcal{M}$ against those trained on $\mathcal{A}$ to demonstrate the effectiveness of the constructed ensemble discovered by our proposed method.

Single macro. We compare the performance of the agents trained on $\mathcal{M}$ against those trained on $(\mathcal{A} \cup \{m\})$. The construction method of m is based on the genetic algorithm proposed in (Chang et al., 2019). The highest performing m constructed by (Chang et al., 2019) is then selected for comparison.

Most frequent action sequence. We extract the top three most frequent action sequences from the experience of the agent trained on $\mathcal{A}$, assemble them as an ensemble, and compare it with that discovered by our method. The sequence extraction algorithm is based on (Durugkar et al., 2016).

Action repeat. We similarly analyze the top three most frequently repeated actions from the experience of the agent trained on $\mathcal{A}$, and assemble them together to form an ensemble as our baseline.



(a) Seaquest (b) Beam Rider (c) Space Invaders

(d) Kung-Fu Master (e) Q*bert (f) Enduro

Figure 3: **Learning curve of the controller**. In this plot, the x-axis represents the training episodes, which also corresponds to the ensembles generated by our controller in time order. The y-axis represents the mean rewards received by the agents over 5M timesteps using the constructed ensembles.

5.1.3 CONTROLLER SETUP

The controller can be regarded as a DRL agent trained in an environment where the goal is to generate a good $\mathfrak{e}$. In order to train $\mathcal{C}$ within a reasonable time budget, the time horizon T and the maximal length of the macros in $\mathfrak{e}$ is configured to pre-defined values $|m| = 3$ and $\mathrm{T} = 3$, respectively. As a result, an $\mathfrak{e}$ with shape $(3, 3)$ is generated at each training episode and evaluated by a worker node.

5.1.4 DRL ALGORITHM SETUP

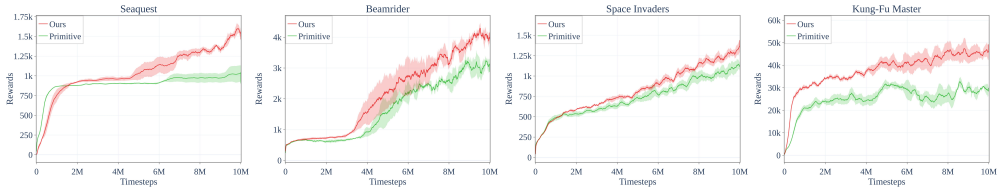
The default DRL algorithm employed in this work is set to PPO. The DRL agents are trained at the worker nodes illustrated in Fig. 1. Each of the worker node receives an ensemble $\mathfrak{e}$ from $\mathcal{C}$, and trains an DRL agent with $\mathfrak{e}$ for 5M timesteps. The ensemble is evaluated for 100 episodes according to Algorithm 1. The detailed hyper-parameter setups of the DRL agent is provided in our appendices.

5.2 ENSEMBLE SEARCH ANALYSIS

Fig. 3 plots the learning curves of $\mathcal{C}$ for *Seaquest*, *BeamRider*, *SpaceInvaders*, and *KungFuMaster*. The ensembles are randomly generated for the four environments for the initial 200 episodes as our bootstrap phase. Afterwards, the controller shifts from the exploration phase to the exploitation phase, with ϵ linearly decays from 1 to 0. It is observed that the received reward r gradually grows as ϵ start to decrease, indicating that ensembles searched by the proposed method is more effective than the randomly generated ensembles (which corresponds to the bootstrap phase of the first 200 episodes).

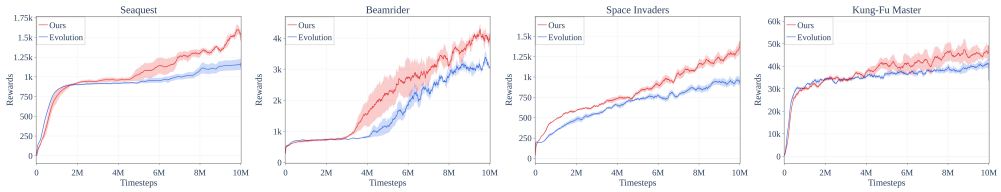
5.3 COMPARISON OF OUR MACRO ACTION ENSEMBLE VERSUS PRIMITIVE ACTIONS

In this section, we compare the performance of the DRL agents trained with the ensembles constructed by our proposed method for 10M timesteps against the *primitive action baseline* defined in Section 5.1.2. The results are depicted in Fig. 4. It is observed that the DRL agents trained with our ensembles outperform the *primitive action baseline* for all of the four games. The learning curves of



(a) Seaquest (b) Beam Rider (c) Space Invaders (d) Kung-Fu Master

Figure 4: Comparison of our proposed methodology with the *primitive action* baseline.



(a) Seaquest (b) Beam Rider (c) Space Invaders (d) Kung-Fu Master

Figure 5: Comparison of our proposed methodology with the *single macro* baseline.

the above cases also reveal that ensembles constructed by the proposed method is complementary to the default DRL algorithm, and do lead to higher episode rewards as well as faster learning speed.

5.4 Comparison of our Macro Action Ensemble versus A Single Macro

To examine if the proposed macro action ensemble is indeed superior to a single macro, we compare the performance of the DRL agent trained with our ensembles against the agents trained with a single macro action n. The macro action m considered in this experiment is constructed by the evaluation method proposed in (Chang et al., 2019). This method iteratively mutates the macro actions in a population such that the performance of the macros constructed by it gradually improves over generations. The top performing m's constructed by this method is selected to be compared with the ensembles discovered by our method. The learning curves are plotted in Fig. 5. It is observed that for all of the cases, the learning curves of ours rise faster and higher than those of the baselines. The above evidence thus validates the assumption that the discovered $\mathfrak{e}$ is indeed superior to the best m.

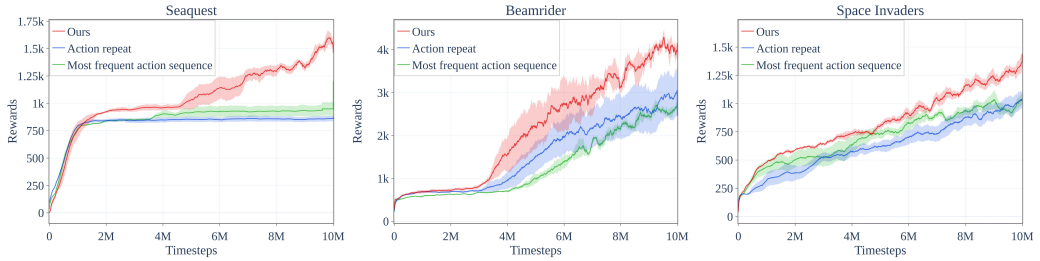
5.5 Comparison Against Action Repeat and Most Frequent Action Sequence

We further compare the proposed method against the *most frequent action sequence* and *action repeat* baselines defined in Section 5.1.2, and plot the training curves of them in Fig. 6. For all of the cases, the proposed method outperform the baselines significantly. In *Seaquest*, both the baselines receive less than 1k rewards even after 10M training timesteps, while our method is able to obtain about 1.7k rewards at the end of the training phase. For *BeamRider* and *SpaceInvaders*, our method is able to achieve about 1.3$\times$ the rewards than the baselines. The results validate that our method is able to discover better $\mathfrak{e}$'s than those constructed from most frequent action sequences or repeated actions.

5.6 Ablation Analysis

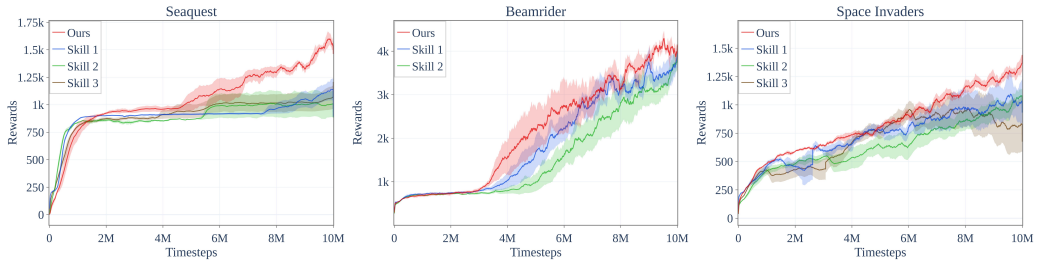
In order to inspect existence of the cooperative property among the macro actions in $\mathfrak{e}$, we additionally compare our ensembles with decoupled macro actions and greedy-based ensembles in this section.

Comparison with decoupled macro actions. To verify the existence of the cooperative property among the macro actions in $\mathfrak{e}$, we evaluate each m in our constructed $\mathfrak{e}$ separately. In other words, the agents are trained on $(\mathcal{A} \cup \{m_i\}), \forall m_i \in \mathfrak{e}$. Fig. 7 plots the learning curves of the agents trained with $\mathfrak{e}$ versus those trained with the decoupled macro actions. It is observed that the curves corresponding to $\mathfrak{e}$ (i.e., *ours*) rise faster and higher than those corresponding to decoupled macro actions. We further validate the cooperative property by qualitatively analyzing the macro actions in $\mathfrak{e}$ discovered in *Seaquest* by our proposed methodology. In Fig. 7, the three constituent macros



(a) Seaquest (b) Beam Rider (c) Space Invaders

Figure 6: Comparison of our methodology with *most frequent action sequence* and *action repeat*.



(a) Seaquest (b) Beam Rider (c) Space Invaders

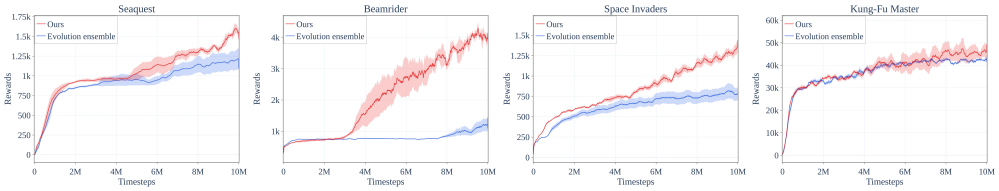
Figure 7: Comparison of $\mathfrak{e}$ discovered by our methodology with the decoupled macro actions.

in $\mathfrak{e}$ are $(1,1,1),(2,4,3),(2,2,2)$, respectively. The first macro $m_1 = (1,1,1)$ corresponds to three consecutive 'fire' actions. The second macro $m_2 = (2,4,3)$ corresponds to an 'up' followed by an immediate 'left' and 'right' moves. The third one $m_3 = (2,2,2)$ corresponds to three consecutive 'up' actions. As the goal of this game is to rescue the victims under the sea with a submarine, the functions of these three constituent macro actions can be interpreted as follows. The macro m_1 is mainly used to eliminate the enemies in front of the submarine; m_2 is responsible for searching the victims by rising the submarine and moving around its left and right; m_3 is responsible for climbing up onto the top of the water right after the victims are rescued. The diverse functions of the macro actions in $\mathfrak{e}$ demonstrates the existence of cooperative property among the constructed macros in $\mathfrak{e}$.

Comparison with greedy-based ensembles. To further validate the effectiveness of the proposed methodology in searching macro action ensembles, we select the top three performing macro actions constructed by (Chang et al., 2019), and assemble them as a greedy-based ensemble $\mathfrak{e}_\mathfrak{g}$. The comparison of the agents trained with $\mathfrak{e}_\mathfrak{g}$ and those trained with $\mathfrak{e}$ discovered by our methodology is depicted in Fig. 8. It is observed that the agents trained with $\mathfrak{e}$ outperform those trained with $\mathfrak{e}_\mathfrak{g}$ for all cases. For *Seaquest*, $\mathfrak{e}_\mathfrak{g} = \{(2,3,4,0),(2,2,0,2),(5,2,2)\}$ according to the experimental results in (Chang et al., 2019). From the constituent macro actions in $\mathfrak{e}_\mathfrak{g}$, we conclude that $\mathfrak{e}_\mathfrak{g}$ is unable to provide the agent with the ability to eliminate enemies and thus leads to a lower performance than $\mathfrak{e}$.

6 Conclusions

In this paper, we presented a methodology for searching macro action ensembles based on the concept of neural architecture search. We proposed a framework with a controller, a replay memory, and a number of worker nodes to generate candidate ensembles and evaluate them. The controller is updated with the gradients derived from the mini-batches extracted from the replay memory. We evaluated the proposed methodology in a number of *Atari* games against several representative baseline methods. Our experimental results validated that the ensembles discovered by our method are complementary to the default PPO algorithm, and outperformed all the baselines in terms of learning efficiency and episode rewards. We further provide a comprehensive set of ablative analysis, and verified the existence of the cooperative property among the macro actions contained in the discovered ensemble.



(a) Seaquest (b) Beam Rider (c) Space Invaders (d) Kung-Fu Master

Figure 8: Comparison of $\mathfrak{e}$ discovered by our methodology with greedy-based ensemble $\mathfrak{e}_g$.

REFERENCES

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, 2016.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, volume abs/1611.02167, 2017.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael H. Bowling. The arcade learning environment: An evaluation platform for general agents. In *J. Artif. Intell. Res.*, 2013.

A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer. Macro-FF: Improving AI planning with automatically learned macro-operators. *J. Artificial Intelligence Research (JAIR)*, 24:581–621, Oct. 2005.

A. Braylan, M. Hollenbeck, E. Meyerson, and R. Miikkulainen. Frame skip is a powerful parameter for learning to play Atari. In *Proc. the Twenty-Nine AAAI Conf. Artificial Intelligence (AAAI-15) Wksp.*, Jan. 2015.

Han Cai, Tianyao Chen, Weinan Zhang, Yingrui Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI*, 2017.

Yi-Hsiang Chang, Kuan-Yu Chang, H. S. Kuo, and Chun-Yi Lee. Construction of macro actions for deep reinforcement learning. *ArXiv*, abs/1908.01478, 2019.

Andrew Coles and Amanda Jane Coles. Marvin: A heuristic search planner with online macro-action learning. *J. Artif. Intell. Res. (JAIR)*, 28:119–156, 01 2007. doi: 10.1613/jair.2077.

A. Dulac, D. Pellier, H. Fiorino, and D. Janiszek. Learning useful macro-actions for planning with n-grams. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pp. 803–810, Nov 2013. doi: 10.1109/ICTAI.2013.123.

I. P. Durugkar, C. Rosenbaum, S. Dernbach, and S. Mahadevan. Deep reinforcement learning with macro-actions. *arXiv:1606.04615*, Jun. 2016.

Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pp. 220–229. Morgan Kaufmann Publishers Inc., 1998.

K. Heecheol, M. Yamada, K. Miyoshi, and H. Yamakawa. Macro action reinforcement learning with sequence disentanglement using variational autoencoder. *arXiv:1903.09366*, May 2019.

Nicolas Manfred Otto Heess, Gregory Wayne, Yuval Tassa, Timothy P. Lillicrap, Martin A. Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *ArXiv*, abs/1610.05182, 2016.

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

A. S. Lakshminarayanan, S. Sharma, and B. Ravindran. Dynamic action repetition for deep reinforcement learning. In *Proc. the Thirty-First AAAI Conf. Artificial Intelligence (AAAI-17)*, Feb. 2017.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17:39:1–39:40, 2015.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *The European Conference on Computer Vision (ECCV)*, September 2018.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *ArXiv*, abs/1711.00436, 2017.

A. McGovern, R. S. Sutton, and A. H. Fagg. Roles of macro-actions in accelerating reinforcement learning. In *Proc. Grace Hopper celebration of women in computing*, volume 1317, 1997.

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

Risto Miikkulainen, Jason Zhi Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. *ArXiv*, abs/1703.00548, 2017.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with deep reinforcement learning. *arXiv:1312.5602*, Dec. 2013.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015a.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proc. Int. Conf. Machine Learning (ICML)*, pp. 1928–1937, Jun. 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015b. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

M. Newton, J. Levine, and M. Fox. Genetically evolved macro-actions in AI planning problems. *Proc. the UK Planning and Scheduling Special Interest Group (PlanSIG) Wksp.*, pp. 163–172, 2005.

M. A. H. Newton, J. Levine, M. Fox, and D. Long. Learning macro-actions for arbitrary planners and domains. In *Proc. Int. Conf. Automated Planning and Scheduling (ICAPS)*, pp. 256–263, Sep. 2007.

H. Onda and S. Ozawa. A reinforcement learning model using macro-actions in multi-task grid-world problems. In *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, pp. 3088–3093, Oct. 2009.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4095–4104, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/pham18a.html.

J. Randlov. Learning macro-actions in reinforcement learning. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1045–1051, Dec. 1999.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *ICML*, 2017.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

S. Sharma, A. S. Lakshminarayanan, and B. Ravindran. Learning to repeat: Fine grained action repetition for deep reinforcement learning. In *Proc. Int. Conf. Learning Represenations (ICLR)*, Apr.-May 2017.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, 2018.

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

A. Vezhnevets, V. Mnih, S. Osindero, A. Graves, O. Vinyals, J. Agapiou, et al. Strategic attentive writer for learning macro-actions. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3486–3494, Dec. 2016.

Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pp. 5279–5288, 2017.

Sijia Xu, Hongyu Kuang, Ziqing Zhuang, Renjie Hu, Yang Liu, and Huyang Sun. Macro action selection with deep reinforcement learning in starcraft. *ArXiv*, abs/1812.00336, 12 2018.

T. Yoshikawa and M. Kurihara. An acquiring method of macro-actions in reinforcement learning. In *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, pp. 4813–4817, Nov. 2006.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, volume abs/1611.01578, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8697–8710, 2017.

A APPENDIX

A.1 ALGORITHM

A.1.1 VALUE FUNCTION

$$V^{\pi}(s) = \mathbb{E}\left\{Q^{\pi}(s_t, a_t)|s_t = s, \pi\right\} \tag{5}$$

$$Q^{\pi}(s, a) = r_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(s', a') Q^{\pi}(s', a') \tag{6}$$

A.1.2 PROXIMAL POLICY OPTIMIZATION (PPO)

We employ PPO [3] as the RL agent responsible for collecting training samples because of its ease of use and good performance. PPO computes an update at every timestep that minimizes the cost function while ensuring the deviation from the previous policy is relatively small. One of the two main variants of PPO is a clipped surrogate objective expressed as:

$$L^{CLIP}(\theta) = \mathbb{E}[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}\hat{A}, \text{clip}(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}, 1-\epsilon, 1+\epsilon)\hat{A}] \tag{7}$$

where $\hat{A}$ is the advantage estimate, and ϵ a hyperparameter. The clipped probability ratio is used to prevent large changes to the policy between updates. The other variant employs an adaptive penalty on KL divergence, given by:

$$L^{KLPEN}(\theta) = \mathbb{E}[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}]\hat{A} - \beta KL[\pi_{\theta_{old}}(\cdot|s), \pi_\theta(\cdot|s)] \tag{8}$$

where β is an adaptive coefficient adjusted according to the observed change in the KL divergence. In this work, we employ the former objective due to its better empirical performance.

A.2 EXPERIMENTAL DETAILS

A.2.1 PSEUDOCODE

Algorithm 3 Next State

input: Step τ; State h_τ; Action a_τ
output: State $h_{\tau+1}$
function NEXT STATE(t, h_τ, a_τ)
    ensure: Shape of h_τ is $(\mathrm{T}, |\mathcal{A}|)$
    $a_{onehot} \leftarrow$ one hot encoding(a_τ)
    $h_{\tau+1} \leftarrow h_\tau$
    $h_{\tau+1}[\tau, :] \leftarrow a_{onehot}$ ▷ Replace the τth row with a_{onehot}
    return $h_{\tau+1}$
end function

Algorithm 4 Update Q Value

input: Controller $\mathcal{C}$ with weights θ; Replay Memory $\mathcal{Z}$
output: Controller $\mathcal{C}$
function TRAIN CONTROLLER($\mathcal{C}$, $\mathcal{Z}$)
    Random shuffle $\mathcal{Z}$
    for minibatch of transitions in $\mathcal{Z}$ **do**
        set $y_t = \begin{cases} r_t & \text{for terminal } s_{t+1} \\ r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta) & \text{for non-terminal } s_{t+1} \end{cases}$
        Perform a gradient descent step on $(y_t - Q(s_t, a_t; \theta))^2$
    end for
    return $\mathcal{C}$ with updated weights
end function

A.2.2 NOTATION

Table. 1 shows the notations in this paper.

Table 1: Essential notations.

Symbol	Name
$\lvert\cdot\rvert$	Sequence length or set size
$\mathcal{A}$	Primitive action space
a	Primitive action, $a \in \mathcal{A}$
a_t	Action at timestep t
a_τ	Action generated by $\mathcal{C}$ at controller timestep τ
$\mathcal{C}$	Controller
$\mathbb{E}\{\cdot\}$	Expected value
$\mathcal{E}$	Environment
$\mathfrak{E}$	Ensemble space
$\mathfrak{e}$	Ensemble of macro actions, $\mathfrak{e} \in \mathfrak{E}$
e	Ensemble sequence
γ	Discount factor, $\gamma \in [0, 1]$
$\mathcal{H}$	Horizon timestep
h_τ	Controller hidden state at controller timestep τ
$\mathcal{M}$	Augmented action space, $\mathcal{M} = \mathcal{A} \cup \mathfrak{e}, \forall \mathfrak{e} \in \mathfrak{E}$
$\mathfrak{M}$	Macro action space
m	Macro action, $m \in \mathfrak{M}$
ν	Policy over $\mathfrak{E}$
ν^*	Optimal policy over $\mathfrak{E}$
$\mathbb{P}\{\cdot\}$	Probability
$\mathcal{P}$	Power set
p	Environment dynamics
$p^a_{ss'}$	Environment transition dynamic, $p^a_{ss'} = \mathbb{P}\{s_{t+1} = s' \mid s_t = s, a_t = a\}$
π	Policy over $\mathcal{A}$
π^*	Optimal policy over $\mathcal{A}$
Q	Q function
Q^*	Optimal Q function
Q^π	Q function under policy π
Q_θ	Q function parametrized by parameters θ
$Q^*_\mathcal{M}$	Optimal Q function over $\mathcal{M}$
r	Reward function, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
$\hat{r}$	Clipped reward
r_t	One-step reward at t
r^m_s	Expected reward after taking m on s
$\mathcal{S}$	State space of $\mathcal{E}$
s	Regular state, $s \in \mathcal{S}$
s_t	State perceived by agent at t
T	Termination timestep of the controller state
t	Timestep
τ	Timestep of the controller state
V	Value function
V^*	Optimal value function
V^π	Value function under policy π
$V^*_\mathcal{M}$	Optimal value function over $\mathcal{M}$
$\mathcal{W}$	Worker pool
w	Worker, $w \in \mathcal{W}$
$\mathcal{Z}$	Replay memory

A.2.3 ATARI PRIMITIVE ACTION SPACE

Table 2: Atari Primitive Action Space

Environment	0	1	2	3	4	5
BeamRider	NOOP	FIRE	RIGHT	LEFT		
Breakout	NOOP	FIRE	RIGHT	LEFT		
Enduro	NOOP	FIRE	RIGHT	LEFT	DOWN	
Q*bert	NOOP	UP	RIGHT	LEFT	DOWN	
Seaquest	NOOP	FIRE	UP	RIGHT	LE0FT	DOWN
SpaceInvaders	NOOP	FIRE	RIGHT	LEFT		
Note: Action space of Kung-Fu Master is [NOOP, UP, RIGHT, LEFT, DOWN , DOWNRIGHT, DOWNLEFT, RIGHTFIRE, LEFTFIRE, DOWNFIRE, UPRIGHTFIRE, UPLEFTFIRE, DOWNRIGHTFIRE, DOWN-LEFTFIRE]						

A.3 HYPERPARAMETERS

Table 3: List of the hyperparameters for the workers in our experiment.

Hyperparameter	PPO
Discount factor	0.99
Number of frame skip	4
Number of parallel environments	20
Rollout length	128
Batch size	2048
Value function coefficient	0.5
Entropy coefficient	0.0
Gradient clipping maximum	0.5
Optimizer	AdamOptimizer
Learning rate	3e-4
Clipping parameter	0.2