2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Reinforcement Learning in Latent Action Sequence Space

Heecheol Kim[1*], Masanori Yamada[2*], Kosuke Miyoshi[3], Tomoharu Iwata[4], Hiroshi Yamakawa[5]

Abstract— **One problem in real-world applications of reinforcement learning is the high dimensionality of the action search spaces, which comes from the combination of actions over time. To reduce the dimensionality of action sequence search spaces, macro actions have been studied, which are sequences of primitive actions to solve tasks. However, previous studies relied on humans to define macro actions or assumed macro actions to be repetitions of the same primitive actions. We propose encoded action sequence reinforcement learning (EASRL), a reinforcement learning method that learns flexible sequences of actions in a latent space for a high-dimensional action sequence search space. With EASRL, encoder and decoder networks are trained with demonstration data by using variational autoencoders for mapping macro actions into the latent space. Then, we learn a policy network in the latent space, which is a distribution over encoded macro actions given a state. By learning in the latent space, we can reduce the dimensionality of the action sequence search space and handle various patterns of action sequences. We experimentally demonstrate that the proposed method outperforms other reinforcement learning methods on tasks that require an extensive amount of search.**

Index Terms— **Reinforcement Learning, Transfer Learning, Learning from Demonstration**

I. Introduction

The learning-based robot control method has gained significant attention recently because it can potentially be widely used in various complex tasks without physical models of either a robot or environment. The two mainstreams of the learning-based robot control methods are reinforcement learning (e.g., [1], [2], [3]) and imitation learning (e.g., [4], [5], [6]).

Imitation learning learns the relationship between the given state and action from an expert's demonstration. Therefore, demonstrations are required for each task. This limits its application to the real world where robots have to adapt to various tasks.

Reinforcement learning, on the other hand, learns a policy that maximizes given rewards. This gives a great advantage against imitation learning on application to robots because reinforcement learning does not require an expert to give an additional demonstration for every task. Given the appropriate reward function, the agent learns adequate policies automatically if a sufficient number of trials is guaranteed.

Reinforcement learning fails when the search space of a state and/or action increases. In large state spaces such as images, a convolutional neural network (CNN) [7], [8] is used; training from raw image input has become possible by using a CNN in the input states. The large search space of the action side is especially challenging because it exponentially enlarges in tasks where long sequences of actions are essential to obtain rewards.

The search space of action sequences depends on the dimension of actions and the sequence length to obtain a reward. Macro actions have been used to diminish the search space of action sequences, where a macro action is a sequence of primitive actions to solve tasks [9], [10]. Previous studies defined macro actions as repetitions of the same primitive actions [9] or required humans to manually define them [10]. However, repeating the same primitive actions is inefficient for compressing a wide variety of action sequences. Manually defining macro actions incurs high cost and is difficult when knowledge about tasks is incomplete. On the other hand, a data-driven approach can naturally extract macro actions from a demonstration without complete prior knowledge about tasks.

We propose encoded action sequence reinforcement learning (EASRL), which is a novel algorithm to learn flexible macro actions in a latent space for a high-dimensional action sequence search space. With the proposed method, we train an autoencoder (AE) to obtain disentangled representations of macro actions using expert demonstrations on the basis of factorized action variational autoencoders (FAVAE) [11], which is an extension of a variational autoencoder (VAE) [12] for learning disentangled representations of sequences. Then, the proposed method learns a policy network that outputs the encoded macro actions instead of primitive actions. By learning in the latent space, the proposed method reduces the dimensionality of the action sequence search space. In addition, the proposed method can represent flexible action sequence patterns since it can encode any kind of action sequences using the encoder. Also, this method can transfer macro actions acquired from the previous task to learn new tasks. This facilitates convergence to new tasks that share the same action sequence patterns as policies with which macro actions were acquired.

We experimentally demonstrate that the proposed method can learn policy networks in environments with high-dimensional action sequence search space.

[1] Laboratory for Intelligent Systems and Informatics, Graduate School of Information Science and Technology, The University of Tokyo (e-mail: h-kim@isi.imi.i.u-tokyo.ac.jp)
[2] NTT Secure Platform Laboratories (e-mail: masanori.yamada.cm@hco.ntt.co.jp)
[3] narrative nights inc. (e-mail: miyoshi@narr.jp)
[4] NTT Communication Science Laboratories (e-mail: tomoharu.iwata.gy@hco.ntt.co.jp)
[5] Dwango Artificial Intelligence Laboratory (e-mail: hiroshi_yamakawa@dwango.co.jp)
[*] Both authors equally contributed to this paper.

II. Related work

Many studies have used sequences of actions in reinforcement learning [9], [13], [14], [15]. Fine Grained Action Repetition (FiGAR) successfully adopts macro actions into deep reinforcement learning [9], showing that Asynchronous Advantage Actor-Critic (A3C) [16], an asynchronous variant of deep reinforcement learning algorithm, with a learning time scale of repeating the action as well as the action itself scores higher than that with primitive actions in Atari 2600 Games. There are two main differences between the proposed EASRL and FiGAR. First, FiGAR can only generate macro actions that are repetitions of the same primitive actions. On the other hand, macro actions generated with EASRL can be a combination of different primitive actions because EASRL finds an encoded representation of a sequence in a continuous latent space and uses the decoded sequence as macro actions. Second, EASRL learns to encode action sequences and optimizes the policy for the target task independently, whereas FiGAR learns both simultaneously. Although EASRL cannot learn macro actions end-to-end, this algorithm can easily recycle the trained decoder to new target tasks because the decoder is acquired independent of target tasks.

Hausknecht and Stone proposed a reinforcement learning method that uses a parameterized continuous action space [10]. This approach, however, is limited to the tasks where the action has to be selected at every time step, and humans need to parameterize the actions. EASRL can be viewed as extending this model to action sequences, where the latent action sequence space is automatically learned from expert demonstrations.

Macro actions are closely related to the hierarchical reinforcement learning. The option framework [17], [18], [19] is a formulation for considering the problem with two-level hierarchy, and the option is bottom level sub-policy. The option framework provides tools to learn each option from interacting with the environment, whereas EASRL learns options from demonstrations as latent macro actions. The option framework is the semi-Markov decision process (SMDP), which is MDP with durative actions. By reinterpreting the timing of selecting a sequence of actions as a step, EASRL can be interpreted on the MDP framework, s.t. policy $p\left(\hat{a}|s\right)$, MDP $p\left(s'|s,\hat{a}\right)$, and reward $r\left(s',s,\hat{a}\right)$ where $\hat{a}$ is an action sequence, s is the current state, and s' is the state when the macro action is next selected.

III. Proposed algorithm

The proposed method is composed of three steps: action segmentation, representation learning, and reinforcement learning. With the action segmentation step, we segment sequences of actions of experts. With the representation learning step, we obtain encoder and decoder networks by using the segmented sequences on the basis of FAVAE [11], which extends VAE for learning disentangled representations of sequences. In the reinforcement learning step, we search for the optimal policy of a target task in the latent space using the trained decoder network instead of primitive actions. The target task can differ from the task for which the expert demonstrations are generated. Figure 1 shows the overview of our proposed approach.

The expert demonstrations or their segmentations can be directly applied to the reinforcement learning agent to learn a new target task. However, the proposed method learns disentangled representations of segmented action sequences for two reasons. First, if the agent explores these expert demonstrations only, it can only mimic expert demonstrations to solve the task, which results in serious deficiencies in generalizing macro actions. Consider a set of demonstrations containing actions of $\langle$turn right 70°, turn right 60°, , turn right $10^\circ\rangle$. If the environment requires the agent to turn right 75°, the agent cannot complete the task because no combination of the demonstration actions can generate the desired action. On the other hand, latent variables trained with the expert demonstrations acquire generated macro actions to *turn right* θ°. Thus, the agent can easily adapt to the target task. Second, making macro actions from memorized demonstration data reduces the action sequence search space, but we can compress the space more by encoding to a low-dimensional continuous disentangled space.

A. Unsupervised segmentation of macro actions

An episode of an expert demonstration is composed of a series of macro actions, e.g., when humans show a demonstration of moving an object by hand, the demonstration is composed of 1) extending a hand to the object, 2) grasping the object, 3) moving the hand to the target position, and 4) releasing the object. Therefore, expert demonstrations need to be segmented into each macro action. One challenge is that there are usually no ground-truth labels for macro actions. One possible solution is to ask experts to label their actions. However, this is another burden and incurs additional cost.

We use Algorithm 1, which was originally proposed for segmenting signals on the basis of AE [20], for macro action segmentation. This method trains an AE with sliding windows of signal data, acquiring the temporal characteristics of the sliding windows. Then, the distance between the encoded features of two adjacent sliding windows is calculated. All the peaks of the distance curve are selected as segmentation points. One advantage of this method is that it is not domain-specific. This method can be easily applied to expert demonstration data since it is assumed that there are no specific data characteristics. When we use this segmentation method, distance is defined as $\parallel q_{\mathrm{seg}}(a_{ij}) - q_{\mathrm{seg}}(a_{i,j-1}) \parallel_2$, where $q_{\mathrm{seg}}(a_{ij})$ refers to the encoded feature of the jth sliding window on the ith action data. We used a sliding window size of four. Any distance point that is highest among ten adjacent points with a margin of 0.05 is selected as a peak.

B. Learning disentangled latent variables with FAVAE

Once the expert demonstrations are segmented, the proposed method learns continuous latent representations of macro actions with FAVAE. FAVAE contains encoder and decoder networks; the encoder network maps an action sequence into a latent continuous vector, and the decoder

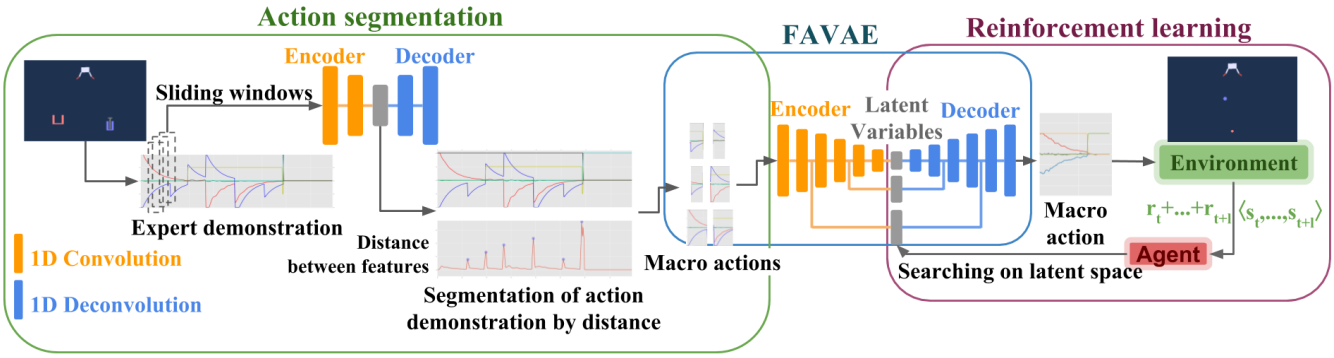


Fig. 1: Overview of EASRL.

network generates an action sequence given a latent vector. FAVAE cannot directly take in segmented macro actions because segmented macro actions may have different lengths. Moreover, FAVAE cannot compute data with different lengths because it uses a combination of a 1D convolution and multilayer perceptron that requires a unified input size across all datasets. To address this issue, macro actions are padded with trailing zeros to match the data length of L, which is the input size of FAVAE. Also, two additional dimensions $\text{action}_{\text{on}}$ and $\text{action}_{\text{off}}$ are added to macro actions to identify if action a_m at timestep m is a real action or zero-padded one. The $\text{action}_{\text{on}}$ is $\langle 1_1, 1_2, ...1_\ell, 0_{\ell+1}, 0_{\ell+2}, ...0_L\rangle$ and $\text{action}_{\text{off}}$ is $\langle 0_1, 0_2, ...0_\ell, 1_{\ell+1}, 1_{\ell+2}, ...1_L\rangle$ when the length of the macro action is ℓ. The loss function of FAVAE is given by

$$\begin{aligned} &- E_{q_\theta(z|x_{1:L})}\left[\log p_\phi\left(x_{1:L}|z\right)\right] \\ &\quad + \beta\left|D_{\text{KL}}\left(q_\theta\left(z|\left(x_{1:L}\right)\right)||p\left(z\right)\right) - C'\right|, \end{aligned} \tag{1}$$

where $x_{1:L} \equiv \langle x_1, x_2, \ldots, x_L\rangle$, is the sequence of actions and their real action identifications, $x_\ell = (a_\ell, \text{action}_{\text{on}\ell}, \text{action}_{\text{off}\ell})$, $p\left(z\right) = \mathcal{N}\left(0, 1\right)$ is a standard Gaussian distribution, z is a latent continuous vector, θ and ϕ represent parameters of encoder and decoder networks, respectively, D_{KL} is the KL divergence, C' increases linearly along with epochs from 0 to hyperparameter C [21] and β is a constant greater than zero that encourages disentangled representation learning. FAVAE consists of three layers of a laddered network architecture proposed by Zhao et al. [22] that can learn the hierarchical features of data. With FAVAE, we used the mean squared loss for actions and the softmax loss for $\text{action}_{\text{on/off}}$ in reconstruction loss.

C. Learning policy with proximal policy optimization (PPO) in the latent space

Our key idea of diminishing the search space of action sequences is learning a policy in the latent space of the macro actions instead of primitive actions directly. We used proximal policy optimization (PPO) [23] as the reinforcement learning algorithm[1], although any reinforcement learning algorithms can be used in our proposed framework.

[1]Our implementation of PPO is based on `https://github.com/Anjum48/rl-examples`

Algorithm 1 Unsupervised segmentation of macro actions

Input: Expert demonstration $D \leftarrow \langle A_1, A_2, ..., A_I\rangle$, where $A_i \leftarrow \langle a_{i1}, a_{i2}, ..., a_{iM_i}\rangle$ is the action sequence of the ith episode, and window size W

Parameter: Encoder $q_{\text{seg}}(\cdot)$

1: $D_{\text{slice}} \leftarrow \langle\langle d_{11}, d_{12}, ...\rangle, \langle d_{21}, d_{22}, ...\rangle, ..., \langle d_{I1}, d_{I2}, ...\rangle\rangle$ // Slice all A_i with window size W
2: Train $q_{\text{seg}}(d_{ij})$ with $d_{ij} \in D_{\text{slice}}$
3: $\text{distance}_{ij} \leftarrow |q_{\text{seg}}(d_{ij}) - q_{\text{seg}}(d_{i,j-1})|$ // Calculate distances between slices
4: $D_{\text{seg}} \leftarrow \langle\langle x_{11}, x_{12}, ...\rangle, \langle x_{21}, x_{22}, ...\rangle, ..., \langle x_{n1}, x_{n2}, ...\rangle\rangle$ // Segment $A_i \in D$ with the distances

We integrate PPO with the latent space learned by FAVAE by using a policy network $\pi_\psi(z|s_t)$ that outputs a continuous vector in the latent space given a state. Let z_t be a latent vector generated by the policy network given current state s_t. The latent vector is transformed by the trained FAVAE decoder ϕ into an action sequence $\langle a_t, a_{t+1}, a_{t+2}, ..., a_{t+L}\rangle$, where subscript L is the output length of the decoder. Then the action sequence is trimmed using the values of $\text{action}_{\text{on}}$ and $\text{action}_{\text{off}}$, which are also decoded from decoder ϕ, to $\langle a_t, a_{t+1}, ..., a_{t+\ell}\rangle$, where subscript ℓ is the first timestep at which $\text{action}_{\text{off}}$ is one. By replacing the primitive action of every timestep with the macro action with a step interval ℓ, the model of the environment with respect to a macro action can be written as $s_{t+\ell}, r^{\text{tot}}_{t+\ell} \sim p(s, r|z_t, s_t)$, where $p(s|z_t, s_t) = \sum_{s_{t+1},...,s_{t+\ell}} p(s_{t+1}|a_t, s_t)p(s_{t+2}|a_{t+1}, s_{t+1})\cdots p(s|a_{t+\ell}, s_{t+\ell})$ is the state transition probability given latent macro action z_t and state s_t, and $r^{\text{tot}}_{t+\ell} = \sum_{k=1}^{\ell} \mathbb{E}_{p(s_{t+k},...,s_{t+1}|a_{t+k-1},...,a_t)} r(a_{t+k}, s_{t+k})$ is the total expected reward when action sequence $\langle a_t, \ldots, a_{t+k}\rangle$ is applied. The total reward between t and $t+\ell$ is summed and regarded as the reward for the result of output z_t. PPO finds a policy by maximizing the sum of discounted rewards while clipping to control the change of the policy. Thus, the loss function of PPO in the latent

Algorithm 2 Reinforcement learning in the latent space with PPO

Input: Decoder of FAVAE ϕ
Parameter: PPO parameter ψ

repeat
  $z_t \sim \pi_\psi(z|s_t)$ // Generate a latent vector given state s_t using policy π_ψ
  $\langle a_t, a_{t+1}, ..., a_{t+\ell} \rangle \leftarrow \mathrm{trim}(\phi(z_t))$ // Map latent vector z_t into an action sequence and trim it using decoder ϕ
  $r_{\mathrm{tot}} \leftarrow 0$ // Initialize a reward
  for $k \leftarrow t$ to $t+\ell$ **do**
    $s_{t+k+1}, r_{t+k} \leftarrow p(a_{t+k}, s_{t+k})$ // Transit the state and obtain a reward with action a_{t+k}
    $r_{\mathrm{tot}} \leftarrow r_{\mathrm{tot}} + r_{t+k}$ // Approximate the expected total reward by Monte Carlo sampling
  end for
  Update the policy by minimizing Eq.(2) using r_{tot}
until Converge

space L^{CLIP} is given by

$$\mathbb{E}_{t'}\left[\min\left(\rho_{t'}(\psi)\hat{A}_{t'}, \mathrm{clip}\left(\rho_{t'}(\psi), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_{t'}\right)\right], \tag{2}$$

where t' is the timestep from the perspective of the macro action, $\rho_{t'}(\psi) = \frac{\pi_\psi(a_{t'}|s_{t'})}{\pi_{\mathrm{old}}(a_{t'}|s_{t'})}$ is the probability ratio between the policy π_ψ and old policy π_{old}, $\hat{A}_{t'}$ is the advantage function at t', ε is a hyperparameter for clipping and ψ is the parameters of the PPO models. Algorithm 2 shows our proposed reinforcement learning algorithm with PPO in the latent space.

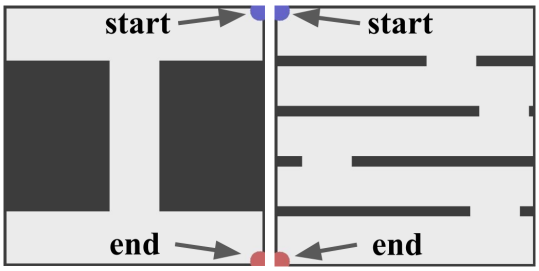
IV. Experiments

We evaluated the proposed method in two environments: ContinuousWorld, a simple 2D environment with continuous action and state spaces, and RobotHand, a 2D environment with a simulated robot hand made by Box2D, a 2D physics engine[2].

A. *ContinuousWorld*

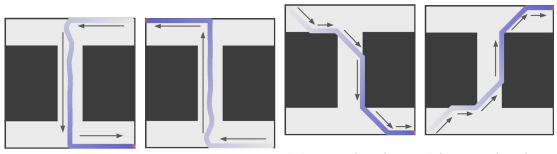
The objective with this environment is to find the optimal trajectory from the starting position (blue dot in Figure 2) to the goal position (red dot in Figure 2). The reward of this environment is negative Euclidean distance $-\parallel x - g \parallel_2$, where x is the position of the agent and g is the position of the goal. The two-dimensional action space is defined by the $\langle$acceleration to the horizontal axis, acceleration to the vertical axis$\rangle$, where each acceleration takes a continuous value in $[-1, 1]$.

There are two tasks in ContinuousWorld: Base and Maze. In Base, agents and goals are randomly placed at the top, bottom, left corner, or right corner, and goals and agents are always on vertically opposite sides. Thus, the number of cases for initial positions of the agent and goal is $2\,(\mathrm{left/right\ of\ agent}) \times 2\,(\mathrm{left/right\ of\ goal}) \times 2\,(\mathrm{up/down}) = 8$. To acquire latent representations of macro actions regardless of scale, the map size is selected between $[2.5, 5]$ randomly. In Maze, the agent and goal are always placed at the same position across different episodes. However, the entrances in the four walls are set randomly for each episode so that the agent has to find an optimal policy for different entrance positions. This makes this environment difficult because walls act like strong local optima of a reward; the agent has to make a long detour with lower rewards to finally find the optimal policy.



(a) Base task (b) Maze task

Fig. 2: ContinuousWorld tasks



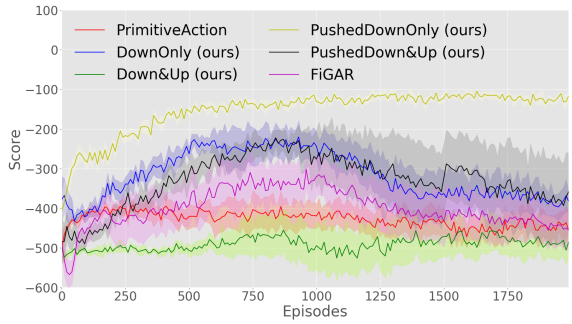
(a) Down (b) Up (c) Pushed Down (d) Pushed Up

Fig. 3: Examples of script trajectories of expert demonstrations in Base of ContinuousWorld. Pushed Down (c) and Pushed Up (d) are Down (a) and Up (b) with accelerates upward or downward. DownOnly uses only trajectories in (a), Down&Up uses those in (a) and (b), PushedDownOnly uses those in (c), and PushedDown&Up uses those in (c) and (d)

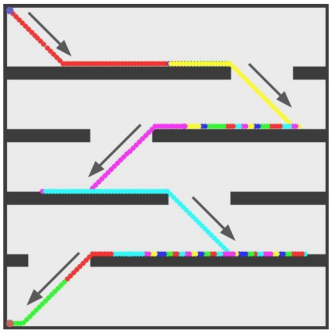
Our purpose was to find disentangled latent macro actions from expert demonstrations in Base and use the latent macro actions to complete the maze tasks. One-hundred episodes of the expert demonstrations were generated in Base using programmed scripts. We compared four different scripts: DownOnly, Down&Up, PushedDownOnly, and PushedDown&Up. All scripts are illustrated in Figure 3. For DownOnly, the goal is only initialized at the bottom of the aisle; therefore, the macro actions do not include upward movements. On the other hand, Down&Up does not limit the position of the goal; thus, upward and downward movements are included in the macro actions. For PushedDownOnly and PushedDown&Up, the agent always accelerates upward or downward in accordance with the goal position.

With the expert demonstrations generated in Base, we used EASRL in Maze, where the disentangle hyperparameter of

[2]The dataset, our experimental setup (hyper-parameters and computing infrastructure), and other supplementary results are available at `https://github.com/EASRLSupplement/EASRLSupplement`

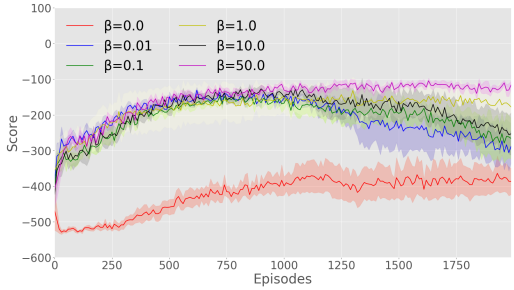


(a) Score averaged over 10 experiments. The shaded area is the standard deviation.

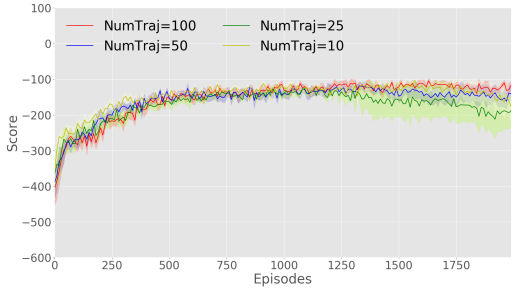


(b) Example trajectory of macro actions. Color change indicates change in macro action

Fig. 4: Results of Maze



(a) KL hyperparameter β



(b) Numbers of expert trajectories

Fig. 5: Comparison between different disentangle hyperparameters β (a) and numbers of expert trajectories (b) using EASRL in PushedDownOnly demonstrations with standard deviations in 10 experiments.

FAVAE was set to $\beta = 50$. We compared EASRL with two existing methods: PrimitiveAction and FiGAR. PrimitiveAction is a standard reinforcement learning method with PPO that learns a policy that outputs primitive actions. FiGAR is a reinforcement learning method that learns macro actions of repeating primitive actions [9]. EASRL performed best for this task, whereas the other two algorithms failed to improve scores (Figure 4a). The choice of the macro action was critical. Whereas PushedDownOnly outperformed the primitive action, EASRL with the other expert demonstrations could not complete the task. Because PushedDownOnly did not contain any demonstrated actions of moving upwards, this can dramatically diminish the action space to search. On the other hand, Down&Up was similar to just repeatedly moving in one direction, which was not sufficient for completing the task. We show an example trajectory of macro actions in Figure 4b. It is difficult to go around the wall because it gives a negative reward in the distance to the goal. The agent made a long detour to avoid getting stuck in the wall by using the long macro actions in the success case.

Figure 5a compares different β of Eq. 1 with FAVAE in using EASRL with PushedDownOnly demonstrations. FAVAE did not learn latent representations in a disentangled manner when β was low. The entangled latent representation of macro actions severely deters matching the state space with macro action space for an optimal policy because the latent space, which actually matches the state space, is distorted. In ContinuousWorld, we found that $\beta \geq 1.0$ is enough to complete Maze. Figure 5b illustrates the average score with different numbers of expert trajectories. Even though we used 100 expert trajectories across all experiments, the number of trajectories did not affect the performance of EASRL.

B. *RobotHand*

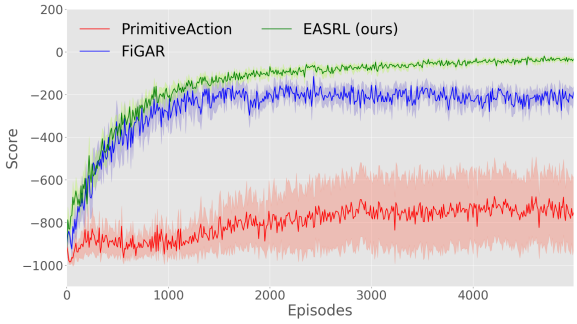
RobotHand has four degrees of freedom (DOFs), i.e., moving along the horizontal axis, moving along the vertical axis, rotation, and grasping operation. The entire environment was built with Box2D `https://box2d.org/` and rendered with OpenGL `https://www.opengl.org/`.

Similar to Base in ContinuousWorld, we provide 100 expert demonstrations with the Peg-insertion task to learn disentangled latent macro actions. In Peg-insertion (Figure 6a), the robot hand is moved from a blue basket to a red one. We chose this task because Peg-insertion is complex enough to contain all macro actions that might be used in target tasks.

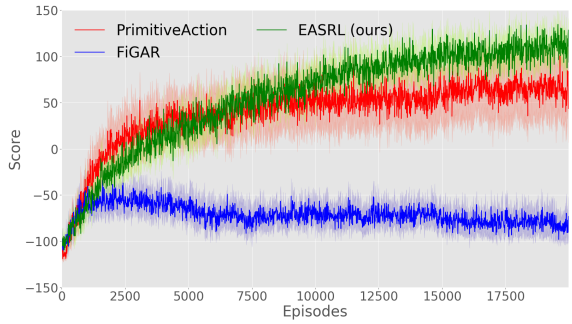
The target tasks are Reaching and Ball-placing. Reaching (Figure 6b) is simple: the robot hand has to reach for a randomly generated goal position (red) as fast as possible. To make this task sufficiently difficult, we used a sparse reward setting in which the robot hand only receives a positive reward of +100 for reaching the goal position within a distance of 0.5; otherwise, there is a time penalty of -1. In Ball-placing (Figure 6c), the robot hand has to carry the ball (blue) to the goal position (red). The ball is initialized at random positions within a certain range, and the goal position

(a) Peg-insertion (b) Reaching (c) Ball-placing

Fig. 6: RobotHand tasks. Blue circle is ball, red circle is goal position, and upside object is robot hand.



(a) Reaching

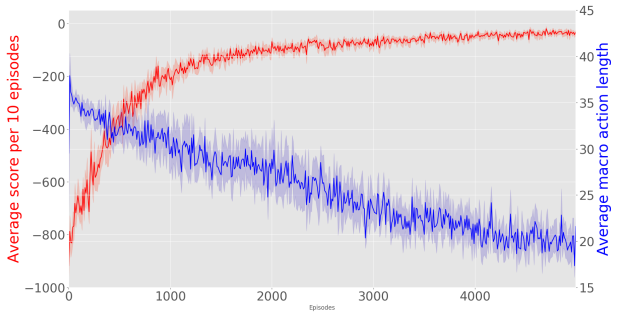


(b) Ball-placing

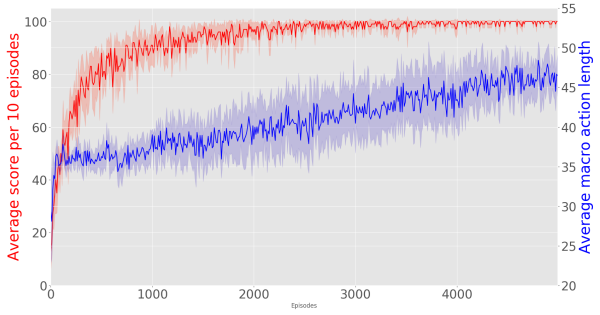
Fig. 7: Comparison of EASRL, PPO with primitive actions, and FiGAR in RobotHand tasks with standard deviations in 10 experiments.

is fixed. The reward is defined by the negative distance between the ball and goal $- \parallel b - g \parallel_2$ where b is the position of the ball and g is the position of the goal. An episode ends when the ball hits the edges or reaches the goal position within a distance of 0.5. An additional reward of +200 is given when the ball reaches the goal.

Figure 7 compares EASRL with disentangled hyperparameter $\beta = 0.1$, PPO with primitive actions, and FiGAR on both Reaching and Ball-placing. PPO with primitive actions and FiGAR failed to learn Reaching and Ball-placing, while EASRL successfully learned both tasks. Because the reward of Reaching is sparse, using primitive actions fails to find rewards. On the other hand, even though the reward of Ball-placing is not sparse, it requires precisely controlling a ball to the goal. FiGAR, which repeats the same primitive actions a number of times, could not precisely control the ball. EASRL is the only algorithm that completed both tasks.



(a) Reaching with time penalty



(b) Reaching without time penalty

Fig. 8: Average macro action length and rewards in Reaching with/without time penalty with standard deviations by 10 experiments.

Note that in the RobotHand experiments EASRL optimized its behavior by shortening macro actions, while fully using the advantages of exploring with latent macro actions. In Reaching, the average length of macro actions gradually diminished (Figure 8a). However, when the time penalty (in Reaching, time penalty of -1 was added to the reward at every time step) is eliminated, the length of a macro action did not diminish (Figure 8b). This is because the agent did not need to optimize its policy in accordance with speed. Long macro actions can be less efficient in optimizing policy than a primitive action when the optimal policy for the task may not match long macro actions. That is why EASRL gradually uses primitive-like actions (macro actions with lengths of 1–3) instead of keeping macro actions with dozens of primitive actions. This result indicates that EASRL can flexibly learn macro actions with different lengths using latent continuous representations.

V. Limitations of EASRL

a) Lack of feed-back control: Searching on latent macro actions instead of primitive actions facilitates searching on the action space in exchange for fast response to unexpected changes in state. We failed to train the bipedal locomotion task in OpenAI Gym (BipedalWalker-v2[3]) with EASRL on the basis of the expert demonstrations

[3]https://gym.openai.com/envs/BipedalWalker-v2/

at BipedalWalker-v2. Because the bipedal-locomotion task requires highly precise control for balancing due to the instability of the environment, it was not suitable to diminish the search space by macro actions in exchange for faster response.

b) Compatibility of macro actions with target tasks: Figure 4 shows that the type of macro actions is critical. If the targeted task does not require the macro actions abstracted from expert demonstrations, ESARL will easily fail because the actions an optimal policy requires are not present in the acquired macro actions. Thus, appropriate expert demonstrations for a targeted task need to be chosen for transferring macro actions to target tasks.

VI. Discussion

We proposed encoded action sequence reinforcement learning (EASRL), an algorithm that uses expert demonstrations to learn disentangled latent variables of macro actions, instead of primitive actions directly, to efficiently search latent spaces. EASRL exhibited higher scores than other reinforcement learning algorithms in tasks that require extensive iterations of search when proper expert demonstrations were provided. This is because EASRL reduces the dimensionality of the action sequence search space using latent macro actions. We consider our work is a promising first step for practical applications of macro actions in reinforcement learning. However, EASRL could not complete tasks that require actions other than macro actions in expert demonstrations. Possible solutions for this include searching for an optimal policy with both macro actions and primitive actions.

References

[1] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research (JMLR)*, vol. 17, no. 1, pp. 1334–1373, 2016.

[2] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *International Conference on Robotics and Automation*. IEEE, 2017, pp. 3389–3396.

[3] T. Haarnoja, A. Zhou, S. Ha, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *arXiv preprint arXiv:1812.11103*, 2018.

[4] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, 1989, pp. 305–313.

[5] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," *Springer Handbook of Robotics*, pp. 1371–1394, 2008.

[6] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," in *International Conference on Robotics and Automation*. IEEE, 2018, pp. 1–8.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[8] 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*, vol. 518, no. 7540, p. 529, 2015.

[9] S. Sharma, A. S. Lakshminarayanan, and B. Ravindran, "Learning to repeat: Fine grained action repetition for deep reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2017.

[10] M. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," in *International Conference on Learning Representations (ICLR)*, 2016.

[11] M. Yamada, H. Kim, K. Miyoshi, and H. Yamakawa, "Favae: Sequence disentanglement using information bottleneck principle," *arXiv preprint arXiv:1902.08341*, 2019.

[12] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *International Conference on Learning Representations (ICLR)*, 2013.

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

[14] A. S. Lakshminarayanan, S. Sharma, and B. Ravindran, "Dynamic action repetition for deep reinforcement learning." in *Association for the Advancement of Artificial Intelligence (AAAI)*, 2017, pp. 2133–2139.

[15] I. P. Durugkar, C. Rosenbaum, S. Dernbach, and S. Mahadevan, "Deep reinforcement learning with macro-actions," *arXiv preprint arXiv:1606.04615*, 2016.

[16] 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 *International Conference on Machine Learning (ICML)*, 2016, pp. 1928–1937.

[17] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[18] D. Precup, *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

[19] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[20] W.-H. Lee, J. Ortiz, B. Ko, and R. Lee, "Time series segmentation through automatic feature learning," *arXiv preprint arXiv:1801.05394*, 2018.

[21] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, "Understanding disentangling in β-vae," in *Neural Information Processing Systems (NIPS) 2017 Disentangled Workshop*, 2018.

[22] S. Zhao, J. Song, and S. Ermon, "Learning hierarchical features from generative models," in *International Conference on Machine Learning*, 2017.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.