Adaptive Selection of Auxiliary Tasks in UNREAL

Hidenori Itaya[1] , **Tsubasa Hirakawa**[1] , **Takayoshi Yamashita**[1] and **Hironobu Fujiyoshi**[1]
[1]Chubu University
{itaya, hirakawa}@mprg.cs.chubu.ac.jp, {yamashita, hf}@mprg.cs.chubu.ac.jp

Abstract

Deep reinforcement learning (RL) has a difficulty to train an agent and to achieve higher performance stably because complex problems contain larger state spaces. Unsupervised reinforcement learning and auxiliary learning (UNREAL) has achieve higher performance in complex environments by introducing auxiliary tasks. UNREAL supports the training of the main task by introducing auxiliary tasks in addition to main tasks during the training phase. However, these auxiliary tasks used in UNREAL are not necessarily effective in every problem setting. Although we need to design auxiliary tasks that are effective for a target tasks, designing them manually takes a considerable amount of time. In this paper, we propose a novel auxiliary task called “auxiliary selection.” Our auxiliary selection adaptively selects auxiliary tasks in accordance with the task and the environment. Experimental results show that our method can select auxiliary tasks and can train a network efficiently.

1 Introduction

Reinforcement learning (RL) problems seek optimal actions to maximize cumulative rewards. Unlike supervised learning problems, the correct labels are not given in RL problems, and an agent needs to determine which actions lead to better results. Because of the recent success of deep neural networks, deep RL methods have been proposed [Mnih *et al.*, 2013; Mnih *et al.*, 2015; Mnih *et al.*, 2016; Jaderberg *et al.*, 2016; Horgan *et al.*, 2018]. Unlike conventional (i.e., non-deep) RL methods, deep RL can deal with more complex tasks such as playing Go [Silver *et al.*, 2016] and video games [Justesen *et al.*, 2017; Firoiu *et al.*, 2017], controlling autonomous systems [Gu *et al.*, 2017], and grasping objects with hand manipulators [Levine *et al.*, 2016].

For RL problems, an agent explores an environment and collects data for training. This exploration is very difficult in that it takes a lot of time to obtain useful data for training a deep RL agent. There have been several works on solving this problem [Lin, 1992; Nair *et al.*, 2015; Mnih *et al.*, 2016; Jaderberg *et al.*, 2016; Horgan *et al.*, 2018]. Among them, unsupervised reinforcement learning and auxiliary learning (UNREAL) [Jaderberg *et al.*, 2016] has been proposed based on an A3C framework. The key idea of UNREAL is introducing auxiliary tasks in addition to training the main tasks. Auxiliary tasks contribute updates to network parameters to solve main tasks, which outperforms scores of A3C.

However, not all auxiliary tasks used in UNREAL necessarily contribute to solving main tasks in every task and every environment. On the contrary, some auxiliary tasks might disturb the training of the main task. Although this issue could possibly resolved by carefully designing auxiliary tasks that depend on each task, designing efficient auxiliary tasks and verifying the efficiency is costly.

In this paper, we focus on using auxiliary tasks in deep RL approaches to propose a novel auxiliary task called *auxiliary selection*. Auxiliary selection adaptively selects auxiliary tasks to be used for network updates depending on the main task and the environment by outputting weights for each tasks. The output weights are multiplied by the loss values of auxiliary tasks. The network of the auxiliary selection and the network of the main and auxiliary tasks are trained at the same time. During the training, the auxiliary selection finds appropriate auxiliary tasks. In our experiments with the DeepMind Lab, we showed that our method selects appropriate auxiliary tasks for each task by analyzing the selected auxiliary tasks and the acquired game scores.

1.1 Contributions

The contribution of this paper is two-fold:

1. The proposed method can select auxiliary tasks adaptively. This enables us to find auxiliary tasks and results by improving the score of the main task and/or suppressing the use of unnecessary auxiliary tasks automatically.
2. Our method can ignores unuseful auxiliary tasks that do not contribute for training of the main task. Therefore, we can train an agent appropriately even if such unuseful auxiliary tasks are included.

2 Related work

One major problem of deep RL is the very long training time because it is difficult to substantially explore a large state space. Auxiliary information is introduced to train an agent efficiently in such complex tasks. Jaderberg *et al.*

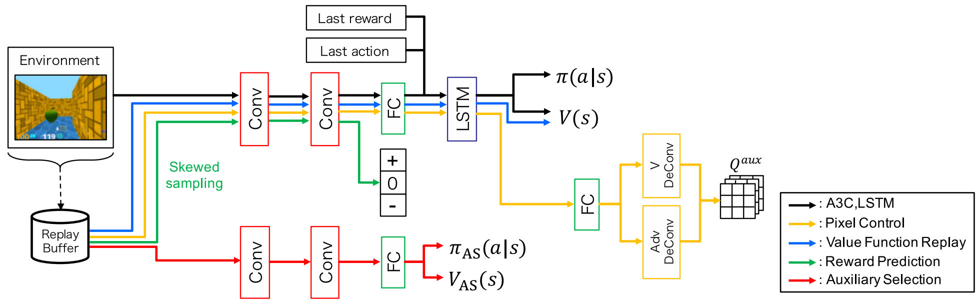

Figure 1: The network architecture of our method. Our method is based on an UNREAL framework. The proposed auxiliary selection is built separately from the UNREAL network and outputs a policy to select auxiliary tasks and a state value.

[2016] proposed UNREAL. UNREAL introduces unsupervised learning-based auxiliary tasks into the A3C [Mnih *et al.*, 2016] framework. In UNREAL, three auxiliary tasks are introduced: i) *pixel control* (PC), which trains actions that vary pixel values, ii) *value function replay* (VR), which shuffles past experiences and trains a state value function, and iii) *reward prediction* (RP), which trains experience that obtains higher rewards and predicts future rewards. These auxiliary tasks are performed within the network of the main tasks. These tasks support the training of the network to solve the main task and achieve higher performance.

If we leverage unnecessary auxiliary tasks that do not contribute to the main task, these tasks might disturb the network's training to solve the main task. To avoid using unnecessary tasks, Du *et al.* [2018] proposed a method that blocks the training data obtained from an auxiliary task if the auxiliary task is not effective. This approach calculates a cosine similarity between gradients for main and auxiliary tasks. If these gradients are similar, the auxiliary task is used for training. Otherwise, training data of the auxiliary task is blocked. Meanwhile, our approach builds an independent network for selecting auxiliary tasks. This can adaptively select efficient auxiliary tasks without interfering with the training of the main task.

3 Proposed method

In this section, we introduce the details of our method, *auxiliary selection*.

3.1 Auxiliary selection

Figure 1 shows the network structure of our method. Our method was built based on UNREAL, and we added the auxiliary selection. In the auxiliary selection, images stored in a replay buffer are input and then a state value $V_{\mathrm{AS}}(s)$ and a policy π_{AS} are output. Among them, the policy π_{AS} represents whether we use each auxiliary task for the main task training or not. Here, we denote weights for each task as $C_{\mathrm{PC}} = \{0, 1\}$, $C_{\mathrm{VR}} = \{0, 1\}$, and $C_{\mathrm{RP}} = \{0, 1\}$. The policy π_{AS} is defined as

$$\pi_{\mathrm{AS}} = (C_{\mathrm{PC}}, C_{\mathrm{VR}}, C_{\mathrm{RP}}). \tag{1}$$

Unlike auxiliary tasks, the network for auxiliary selection is not shared with the network of the main task. Therefore, we train the network of auxiliary selection independently. Thus, we adaptively select auxiliary tasks depending on the environment.

3.2 Loss functions

We formulated the loss function of our method by using the loss function of conventional UNREAL as follows:

$$L = L_{\mathrm{A3C}} + C_{\mathrm{PC}} \sum_c L_Q^{(c)} + C_{\mathrm{VR}} L_{\mathrm{VR}} + C_{\mathrm{RP}} L_{\mathrm{RP}}, \tag{2}$$

where L_{A3C} is a loss value of the main task (i.e., A3C), and $\sum_c L_Q^{(c)}$, L_{VR}, and L_{RP} are loss values of each auxiliary task. Note that, in terms of the pixel control, we split an input image into $n \times n$ grid and compute losses for each grid. Hence, $L_Q^{(c)}$ represents the loss of n-step Q-learning for a grid c. In our method, we select auxiliary tasks by multiplying loss value and binary weight obtained from auxiliary selection.

In case that we train the auxiliary selection by using Eq. (2) simultaneously, the network is trained so that C_{VR}, C_{PC}, and C_{RP} become zero. Therefore, we define another loss function to train the network of the auxiliary selection and train the network apart from training the main and auxiliary tasks. The loss function of the auxiliary selection can be formulated using loss functions of the state value $V_{\mathrm{AS}}(s)$ and the policy $\pi_{\mathrm{AS}}(a|s)$ as follows:

$$L_{\mathrm{AS}v} = (r + \gamma V_{\mathrm{AS}}(s_{t+1}, \theta^-) - V_{\mathrm{AS}}(s_t, \theta))^2 \tag{3}$$

$$L_{\mathrm{AS}p} = -\log(\pi_{\mathrm{AS}}(a|s))A(s, a) - \beta H(\pi_{\mathrm{AS}}), \tag{4}$$

where θ^- is the network parameters before a network update. And, an entropy $H(\pi_{\mathrm{AS}})$ promotes explorations that prevent the network parameters from converging into a local minima. And β is a scale parameter for the entropy $H(\pi_{\mathrm{AS}})$.

Finally, the loss function of auxiliary selection is defined by adding losses of Eqs. (3) and (4) as

$$L_{\mathrm{AS}} = L_{\mathrm{AS}v} + L_{\mathrm{AS}p}. \tag{5}$$

3.3 Algorithm

First, we synchronize the network parameters of each thread $\theta'_{\mathrm{UNREAL}}$ and θ'_{AS} with shared parameters θ_{UNREAL} and θ_{AS}, respectively. Then, agents of each thread repeat taking actions in an environment by following policy $\pi(a_t|s_t)$ until agents lead to a termination condition or t_{max} steps. The experiences (s_{t+1}, r_t, a_t) are stored in a replay buffer. Next,

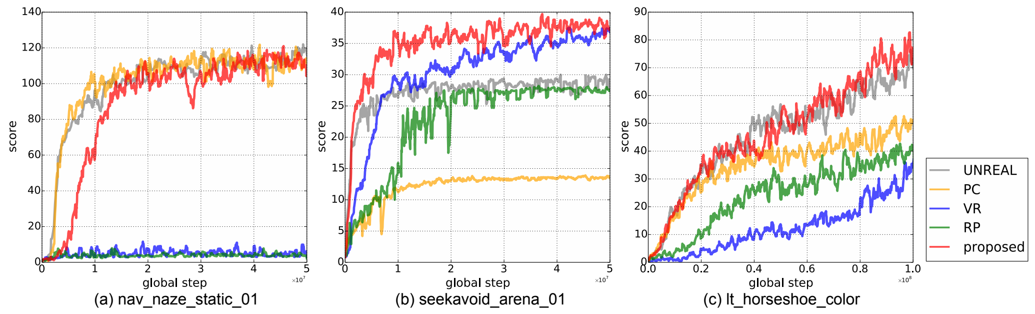

Figure 2: The scores of DeepMind Lab over different global steps. The horizontal axis shows the number of global steps to update network parameters and the vertical axis shows scores of each task. From left to right: the scores of nav_mazze_static_01, seekavoid_arena_01, and lt_horseshoe_color.

we execute the auxiliary selection and three auxiliary tasks in turn, and we compute the gradients $d\theta_{\text{UNREAL}}$ and $d\theta_{\text{AS}}$ shown in Eqs. (2) and (5). By using these gradients, we update the network parameters θ_{UNREAL} and θ_{AS}. The above procedures are processed on each thread asynchronously and we repeat these processes until $T_{\max}$ steps.

4 Experimental results

4.1 Experimental settings

We tested our method in a DeepMind Lab environment [Beattie *et al.*, 2016]. DeepMind Lab mainly contains three games: i) nav_maze_static_01 (maze), ii) seekavoid_arena_01 (seekavoid), and iii) lt_horseshoe_color (horseshoe).

We compared our method with the following baselines:

UNREAL: A method uses three auxiliary tasks for training.

PC, VR, and RP: Each of them uses an auxiliary task for training, respectively.

We did not change the hyperparameters during the training phase of each method. For maze and seekavoid, we trained networks in 5.0×10^7 steps. For horseshoe, we updated the network parameters in 1.0×10^8 steps.

4.2 Results

nav_maze_static_01 Figure 2(a) shows scores for nav_maze_static_01. UNREAL and PC achieved higher performances, while scores of VR and RP are almost zero. This means that VR and RP did not improve the main task. The PC promoted an agent to take action changing pixel values. In other words, this enabled an agent to move in every corner of the maze environment. Our method also achieved a higher score as UNREAL and PC.

seekavoid_arena_01 Figure 2(b) shows the score of seekavoid_arena_01. The performances of PC and RP were inadequate because actions changing pixel values are not suitable for this task. Moreover, if an agent can obtain several rewards in this task, RP would be inefficient. On the other hand, UNREAL and VR achieved higher scores. Surprisingly, VR outperformed UNREAL, and our method also achieved higher performance as VR.

lt_horseshoe_color Figure 2(c) shows the score of lt_horseshoe_color. PC was the most effective. The reason is that actions that defeat enemies change pixel values significantly. However, UNREAL outperforms the other methods, and our method achieved the same performance as UNREAL.

Table 1: The number of times each auxiliary task was selected in one episode. The percentage in each bracket indicates the selection ratio of each auxiliary task in one episode.

Env.	Auxiliary task		
	PC	VR	RP
maze	435.4 (48.3%)	487.8 (54.1%)	369.0 (41.0%)
seekavoid	0.3 (0.1%)	300.0 (100.0%)	0.0 (0.0%)
horseshoe	8545.1 (94.9%)	14.1 (0.1%)	8998.2 (99.9%)

4.3 Analysis of the selected auxiliary tasks

Figure 3 shows the number of actions output from the auxiliary selection during one episode on each game. Also, Tab. 1 shows the number of times each auxiliary task was selected in one episode. Note that the number of selected auxiliary tasks was calculated by averaging over 50 episodes. The number of action steps in an episode was 900 for the maze, 300 for seekavoid, and 9,000 for horseshoe.

The results of the maze show that $\{C_{\text{PC}}, C_{\text{VR}}, C_{\text{RP}}\} = \{0, 1, 0\}$, and $\{1, 0, 1\}$ were often selected. All auxiliary tasks were equivalently selected, as shown in Tab. 1. Because appropriate auxiliary tasks for the maze task were UNREAL or PC, our method equally selected all auxiliary tasks.

In seekavoid, $\{C_{\text{PC}}, C_{\text{VR}}, C_{\text{RP}}\} = \{0, 1, 0\}$ was selected, which means our method stably selected the value function replay. Since these results correspond to the results in Fig. 2(b), our method only selects auxiliary tasks that contribute to the training of the main task.

In horseshoe, $\{C_{\text{PC}}, C_{\text{VR}}, C_{\text{RP}}\} = \{1, 0, 1\}$ was selected; that is, pixel control and reward prediction were often selected. Although the best score was achieved by UNREAL, auxiliary selection for horseshoe did not select value function replay. To analyze the reason of the selection, we conducted additional experiments. In addition to the results of baselines shown in Fig. 2(c), we added the following baselines:

A3C: A method without auxiliary tasks.

PC+RP: A method uses pixel control and reward prediction.

Figure 4 shows the scores of each baseline and our method. This results shows that the score of VR was lower than that of

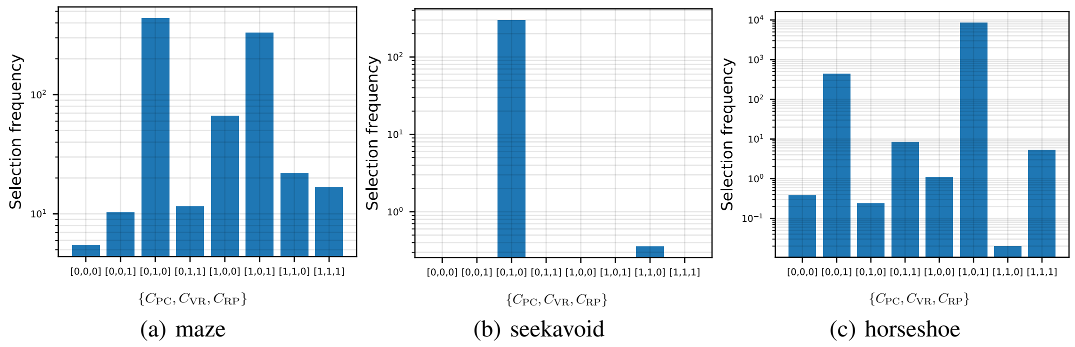

Figure 3: The number of selected actions of the auxiliary selection during one episode. Horizontal axis shows actions $\{C_{\mathrm{PC}}, C_{\mathrm{VR}}, C_{\mathrm{RP}}\}$ output from auxiliary selection, and the vertical axis shows the number of selected actions. From left to right: results of (a) nav_maze_static_01 trained with 5.0×10^7 steps, (b) seekavoid_arena_01 trained with 5.0×10^7 steps, and (c) lt_horseshoe_color trained with 1.0×10^8 steps.

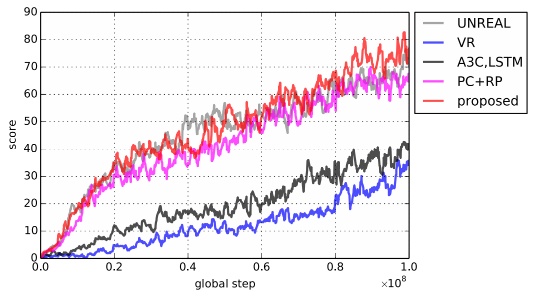

Figure 4: Game scores of horseshoe over different global steps. The horizontal axis shows the number of global steps to update network parameters and the vertical axis shows scores of each task.

A3C. And PC+RP achieved the same score as UNREAL and our method. Therefore, our method successfully removes the value function replay from the training of horseshoe.

The results above show that our approach can select auxiliary tasks that contribute to training the main task.

5 Conclusion

In this paper, we proposed auxiliary selection that adaptively selects auxiliary tasks to be used for training. Our method selects auxiliary tasks by multiplying the loss values of each auxiliary task and the weights obtained from auxiliary selection in the training phase. This enables us to train efficiently by not having to design auxiliary tasks for each environment manually. Experimental results show that our method can select proper auxiliary tasks and train the network to solve main tasks efficiently. Our future work includes experiment in various environments and introduce other auxiliary tasks.

References

[Beattie *et al.*, 2016] C. Beattie, J. Z. Leibo, et al. DeepMind Lab. *arXiv preprint*, 2016.

[Du *et al.*, 2018] Y. Du, W. M. Czarnecki, et al. Adapting Auxiliary Losses Using Gradient Similarity. *arXiv preprint*, 2018.

[Firoiu *et al.*, 2017] V. Firoiu, W. F Whitney, et al. Beating the World's Best at Super Smash Bros. Melee with Deep Reinforcement Learning. *arXiv preprint*, 2017.

[Gu *et al.*, 2017] S. Gu, E. Holly, et al. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *ICRA*, 2017.

[Horgan *et al.*, 2018] D. Horgan, J. Quan, et al. Distributed Prioritized Experience Replay. In *ICLR*, 2018.

[Jaderberg *et al.*, 2016] M. Jaderberg, V. Mnih, et al. Reinforcement Learning with Unsupervised Auxiliary Tasks. *arXiv preprint*, 2016.

[Justesen *et al.*, 2017] N. Justesen, P. Bontrager, et al. Deep Learning for Video Game Playing. *arXiv preprint*, 2017.

[Levine *et al.*, 2016] S. Levine, P. Pastor, et al. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. In *ISER*, 2016.

[Lin, 1992] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, May 1992.

[Mnih *et al.*, 2013] V. Mnih, K. Kavukcuoglu, et al. Playing Atari with Deep Reinforcement Learning. In *NIPS workshop*, 2013.

[Mnih *et al.*, 2015] V. Mnih, K. Kavukcuoglu, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Mnih *et al.*, 2016] V. Mnih, A. P. Badia, et al. Asynchronous Methods for Deep Reinforcement Learning. In *ICML*, 2016.

[Nair *et al.*, 2015] A. Nair, P Srinivasan, et al. Massively parallel methods for deep reinforcement learning. In *ICML workshop*, 2015.

[Silver *et al.*, 2016] D. Silver, A. Huang, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.