Reusing Agent's Representations for Adaptation to Tuned-environment in Fighting Game

Dae-Wook Kim
Electronics and Telecommunications Research Institute
Daejeon, South Korea
dooroomie@etri.re.kr

Sung-Yun Park
Electronics and Telecommunications Research Institute
Daejeon, South Korea
tjddbs5671@etri.re.kr

Seong-il Yang
Electronics and Telecommunications Research Institute
Daejeon, South Korea
siyang@etri.re.kr

***Abstract*—Reinforcement learning agents have been used for one-on-one fighting battlefield or quality assurance (QA) of commercial games. In spite of its performance beyond human-level, it is not easy to apply because actual commercial games are frequently updated. In this paper, we propose a method to adapt reinforcement learning agent to slightly tuned environment by reusing representations of neural network. Agent trained by the proposed method converges at 2.11 million steps, which shows at least 3 times faster than those of fine-tuning and training from scratch to reach the same competence. We also tested larger representation layers with smaller actor-critic ones. Although it fails to train agent, it demonstrates distinct characteristics. Finally, action distributions of fully trained agent for each environment are analyzed. Entire process of adapting to new environment presented in this paper gives insights of game balancing framework to game developers.**

***Index Terms*—reinforcement learning, transfer learning, feature representation, FightingICE**

I. Introduction

Reinforcement learning has been studied in many game environments. It has been proved that reinforcement learning agent shows better performance than human on the certain game genre. In this trend, more and more game companies try to make a profit with the contents of reinforcement learning agents such as one-on-one fighting battlefield [1] or quality assurance (QA) for game balancing [2]. Commercial game, however, needs continuous maintenance including bug fix, contents update or game balancing. Even if reinforcement learning is a powerful tool to find an optimized strategy in a complex game environment, an agent's performance drops rapidly when it plays in a slightly adjusted game environment. Besides, it can act abnormal behavior in such an environment. Meanwhile, re-training for each game update would be a simple alternative; however, it needs a lot of computational loads. For this reason, an agent should quickly adapt to a certain new environment while using a trained model.

Considering adaptation to a new environment, we can think about two simple solutions. The first is fine-tuning that trains agents with additional epochs in the new environment while initializing parameters from the trained model. It is useful when the environment is not as much changed such as a modification of skill damage buff & nerf[1]. It does not need to learn feature representation from scratch. However, exploration in the training process can be constrained by prior knowledge of the trained model. The second one is training from scratch. It is an advantage in a drastically changed environment like substituting characters or modifying the skill triggers. In short, re-training feature representation is more effective than transferring it depending on the size of the environmental difference.

The practical scenario for the one-on-one fighting game we focus on is as follows.

- Step 1 : Set the game environment.
- Step 2 : Generate reinforcement learning agent by self-play or simulation with hand-made agents.
- Step 3 : Find whether the agent's strategy or behavior is efficient or not.
- Step 4 : If some strategy or behavior not expected by the game designer has been found, tune the parameters of skills or damages.
- Step 5 : Generate agent again in tuned-environment.
- Step 6 : Go to step 3.

In this paper, we propose a representation transfer method to promote adaptation to tuned-environment in fighting games. The contribution of this paper is as follows. As far as we know, it is the first time for representation transfer in fighting games, especially the *FightingICE* [3] which is a well-known environment for research purposes. Assuming the differences in the environment are small enough, the proposed method shows training faster than fine-tuning. In addition, We analyzed the effect of the size of the representation layers. Finally, it is possible to utilize the game balance tuning for commercial games by comparing agent behaviors in the prior environment with those in the new environment.

The rest of this paper is organized in the following order. In chapter 2, we describe works related to our study. Chapter 3 explains our proposed method with the game environment. Experiment and result are presented in chapter 4. We remark on the conclusion in chapter 5.

[1]The terms of buff and nerf are used in video games. Buff means to increase the power of a game element and nerf is the opposite of buff

II. Related Work

Many researchers have tried to apply reinforcement learning to game development process. In [1], they made one-on-one game agent in commercial game *Blade&Soul*. Using reward shaping, the agent got its own style. J. Pfau, et al. [2] generated playable agent using supervised learning in another commercial game *Aion*. They evaluated character balances from various simulations while modifying health point and attack power. Y. Zheng, et al. [4] showed that an agent trained by reinforcement learning and genetic algorithm could find bugs in games. In [5], they also found game bugs from route heatmap generated by reinforcement learning agent.

Meanwhile, many studies are trying to apply reinforcement learning agents to similar environments. Transfer learning in deep reinforcement learning is well defined in [6]. According to [6], the proposed method in this paper corresponds to the knowledge transfer and categorically belongs to the representation transfer. Representation learning generates features by itself, combining important information from inputs. A high-performance agent can be obtained without any game insight from the game designer. For example, the agent trained with representation learning beats humans without prior knowledge of Go [7].

As a prior study on representation transfer, A. A. Rusu, et al. [8] attempted to transfer the representation to another environment by designing the double size of the original network in parallel, one is copied from the original and the other is initialized by random numbers. The agent learns parallel networks in a new environment, considering whether to use knowledge of the original network or not. However, its experimental environment is somewhat difficult to apply to commercial games. They made a new environment by flipping the screen or adding image noises; therefore, the spatial relation of the input state was not preserved. In addition, since the neural network is composed of parallel structures, the computational load for training increases as the size of the network increases. In [9], for *breakouts* of the Atari game, they tested whether the agent could be transferred to the environment where the noise was added and compared it with fine-tuning and training from scratch similar to our experiment. However, they showed that the training from scratch got the best performance.

III. Method

We concentrate on the player versus player (PvP) environment, especially the environment with minor changes, rather than the player versus environment (PvE) discussed in the previous chapter.

A. Game Environment

Agents are trained in *FightingICE* which is a fighting game environment for research purpose as shown in Fig. 1. *FightingICE* has held international competition every year. Hence, there are many various agents that have already been developed. Besides, it provides customization of detailed skill design such as skill damage, the energy value needed to be committed, each action frame, and so on. The opponent agent is carefully selected as Monte Carlo Tree Search (MCTS) AI because it has ability to act local optimal strategy in most cases regardless of environmental changes. We fix the battle characters as *Zen* vs *Zen* for the experiment.

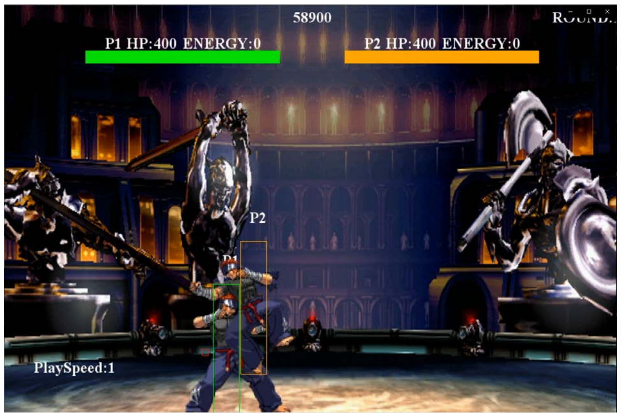

Fig. 1. Fighting game environment called *FightingICE*.

Two types of environment used in our experiments are detailed in Table I. Env A is the same as the default setting in *FightingICE* 4.40. we found that the combination of STAND_B and DASH[2] actions was the most effective strategy from a preliminary experiment based on reinforcement learning. In this environment, a fully-trained agent approaches the opponent using DASH and then pins the opponent by taking the successive action of STAND_B at the corner. For this reason, we nerf the hit damage of STAND_B in Env B to -1.[3] ‘Hit Add Energy’ means the energy the agent obtains when the agent hits the opponent. We also drastically reduced it to confine the skill use.

B. Reinforcement Learning

The configuration of state, action, reward, network structure, and hyper-parameters for reinforcement learning is assigned the same as [10]. The dimension of the state vector equals 330. The reward composed of HP, round, and time is applied. Proximal policy optimization (PPO) [11] is used for reinforcement learning algorithm.

[2]These are skill names in *FightingICE*. For example, STAND_B is one of the kick attacks.

[3]We also noticed that STAND_B decreased the opponent’s HP even if its damage was set to 0 because the combo system in *FightingICE* caused additional damage which was not able to be customized.

TABLE I
Specification of Game Environments for Experiments

Name	Description	Specification of STAND_B	
		Hit Damage	Hit Add Energy
Env A	FightingICE ver. 4.40	10	5
Env B	Nerf STAND_B from ver. 4.40	-1	1

C. Reusing Representation

Since fine-tuning trains the agent based on the agent itself trained in the previous environment, it transfers knowledge of the features. However, as the additional training progresses, error between the action policy and estimated value function affects the entire network. As a consequence, it can mess up the feature representation which the network has learned from the original environment. This problem gets worse as the network becomes bigger.

For this reason, we propose reusing representations to accelerate transfer learning. Fig. 2 (a) shows the network to generate reinforcement learning agent on *FightingICE*. Let's assume that the front part of the network corresponds to processing the input state as a feature, and the latter part of it does to find the policy. Then, we are able to cut the middle of it. Fig. 2 (b) and (c) illustrate that the network composition according to the cutting point. In Fig. 2 (b), for example, the first two layers play a role in representation and remain layers do a role in action selection. For the representation function f, the actor-critic function g, the state s and the initial network parameter θ^0, the feature z is expressed as

$$z = f(s; \theta_f^0) \tag{1}$$

Also, policy π and estimated value V is driven by

$$\pi, V = g(z; \theta_g^0) \tag{2}$$

IV. Experiment and Result

We evaluated trained agents by the number of steps to reach the average episode reward 5. This threshold was set based on the relationship between the rewards. The agent over the threshold reward showed stable win rate performance against the opponent.

The models for the experiment are shown in Table II. For the first experiment, we tested whether the proposed method improves adaptation to the tuned environment comparing other methods. Four models of baseline 1, baseline 2, RR (Reusing Representation) v1, and RR v2 were tested. Baseline 1 is vanilla reinforcement learning without transfer learning to Env B. Baseline 2 tunes trained model from Env A to Env B. As the first version of the proposed method, RR v1 freezes the parameters of the representation function f trained in Env A and fine-tunes only for the actor-critic function g. The second version, RR v2, similarly freezes representation, but the training starts with the actor-critic function initialized to a random number. We expected that it explored action space broader than the first version.

For the second experiment, models with different network divisions are trained where all models reuse representations. This experiment is to test the performance according to the number of training parameters. RR v1 L3 consists of representation function of the first three layers and actor-critic function of the remaining layers based on RR v1. RR v2 L3 is composed in the same way as RR v1 L3.

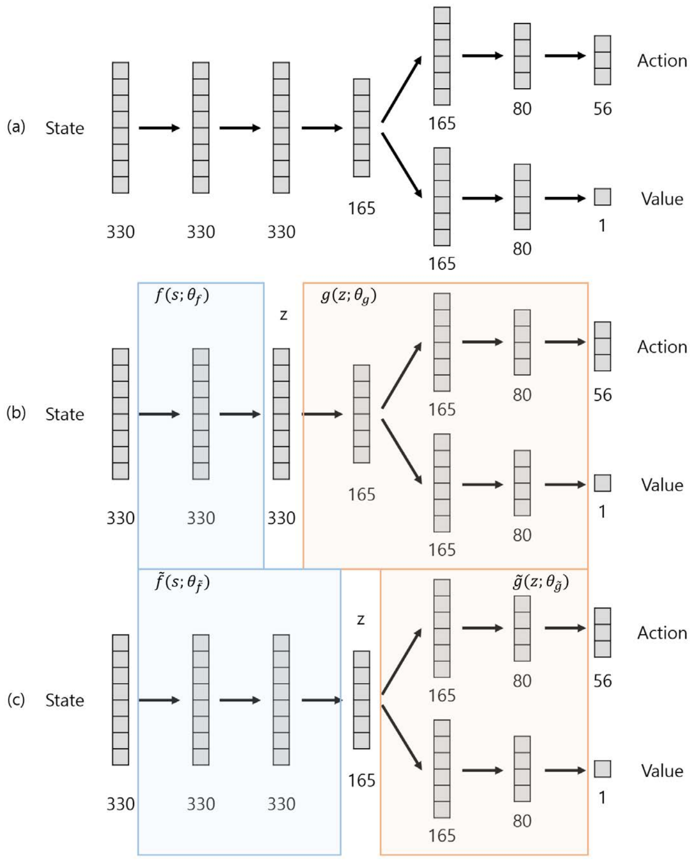

Fig. 2. Neural network configurations of the proposed method to train reinforcement learning agent. (a) Basic network without network division for baseline. (b) Representation function with the first two layers and actor-critic function with remaining layers for RR v1 and RR v2. (c) Representation function with the first three layers and actor-critic function with remaining layers for RR v1 L3 and RR v2 L3.

Detailed experimental results are described in Table III. Fig. 3 also illustrates the result of each experiment. Baseline 1 spent the longest time to reach the desired performance. Baseline 2 reached the threshold within about 7M steps. It is 5-times faster than baseline 1. When the difference between environments is small, it demonstrates that fine-tuning needs a shorter training time than the training from scratch. RR v1 shows the fastest adaptation to the tuned environment in 2.1M steps. RR v2 touched the threshold at 4.4M steps, which is slightly slower than RR v1. While RR v1 needs to tune only the action output of the network, the parameter initialization of RR v2 requires another computational cost. Nevertheless, it still shows superior performance compared to baseline 1 and baseline 2. It means the representation function compresses necessary information well from the input state and increases the training speed.

Surprisingly, RR v1 L3 and RR v2 L3 failed in training despite fewer network parameters. We stopped it at 20M steps since the performance had not improve during the long and tedious training process. There are two possibilities for training failures. First, as the size of the representation function

TABLE II
TRAINING PROCESS OF EACH MODEL

Name	Description	Training Process	Number of Parameters
Baseline 1	training from scratch	$(\theta_f^0, \theta_g^0) \rightarrow (\theta_f^B, \theta_g^B)$	359,032
Baseline 2	fine-tuning	$(\theta_f^A, \theta_g^A) \rightarrow (\theta_f^B, \theta_g^B)$	
RR v1	parameter preserved	θ_f^A frozen, $\theta_g^A \rightarrow \theta_g^B$	140,572
RR v2	parameter initialized	θ_f^A frozen, $\theta_g^0 \rightarrow \theta_g^B$	
RR v1 L3	RR v1 with 3 layers representation	$\theta_{\tilde{f}}^A$ frozen, $\theta_{\tilde{g}}^A \rightarrow \theta_{\tilde{g}}^B$	85,957
RR v2 L3	RR v2 with 3 layers representation	$\theta_{\tilde{f}}^A$ frozen, $\theta_{\tilde{g}}^0 \rightarrow \theta_{\tilde{g}}^B$	

TABLE III
STEPS TO REACH THE MEAN EPISODE REWARD 5

Name	Steps (M)	Relative Speed Compared to baseline 2
Baseline 1	35.38	19.84 %
Baseline 2	7.02	100 %
RR v1	**2.11**	**332.70 %**
RR v2	**4.41**	**159.18 %**
RR v1 L3	Failed (> 20)	-
RR v2 L3	Failed (> 20)	-

increases, that of the actor-critic function decreases. Henc layers are too small to extract optimal policy and value from representation features. Second, policy and value outputs are limited by the representation function. In the second situation, the optimal policy cannot be obtained from any feature inputs. Although reusing representation speeds up adaptation in a new environment, it cannot guarantee performance depend on network structures and parameters. Additional experiments and tests such as varying network division are left for future work.

Fig. 4 shows the action distribution of the agents that find the sub-optimal in each environment. Because of nerfing STAND_B, the proportion of STAND_B decreased to 0 in Env B. As an alternative action, the agent found CROUCH_B, CROUCH_FB, and STAND_A. In addition, it seems that STAND_B works well with DASH, and CROUCH_FB does with FORWARD_WALK.

V. CONCLUSION

Reinforcement learning agents have been used in many commercial games. However, it has a definite disadvantage of performance degradation when a game environment is changed. In this paper, we proposed the method of reusing representations of neural networks. Compared to fine-tuning and training from scratch, the proposed method shows remarkable training speed. It converges at 2.11 million steps, which is 3-times faster than fine-tuning. Meanwhile, using more layers on the representation function conversely failed to train it. We also analyzed the action distribution of trained agents for each environment. The whole process described in this paper provides insights into the game balancing framework for game developers.

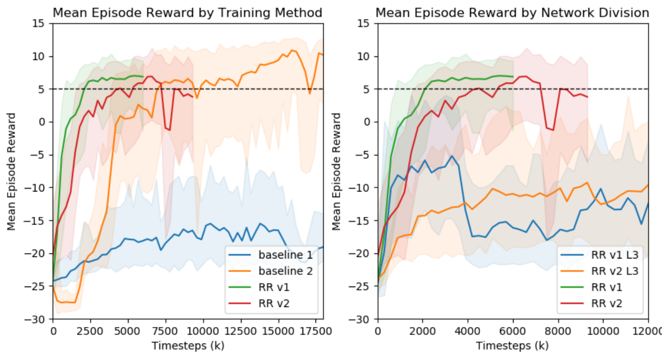

Fig. 3. Mean episode reward by training method and network division. The later parts of Baseline 1, RR v1 L3 and RR v2 L3 were cut because they took too long training time.

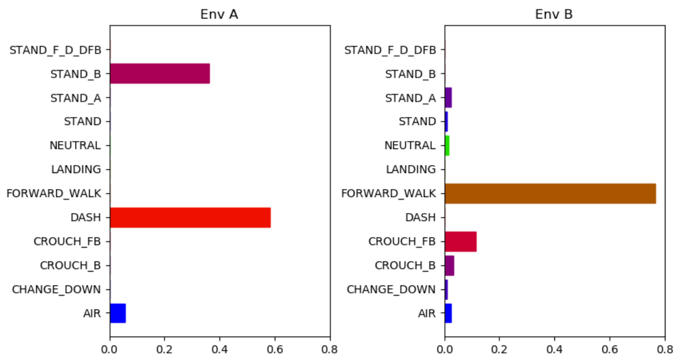

Fig. 4. Action distributions of fully trained agent for each environment. Since STAND_B was nerfed on Env B, its proportion is close to 0.

ACKNOWLEDGMENT

This research is supported by Ministry of Culture, Sports and Tourism (MCST) and Korea Creative Content Agency (KOCCA) in the Culture Technology (CT) Research & Development Program 2021 (Project Number: R2019020067).

REFERENCES

[1] I. Oh, S. Rho, S. Moon, S. Son, H. Lee, and J. Chung, “Creating pro-level AI for a real-time fighting game using deep reinforcement learning,” IEEE Transactions on Games, 2021.

[2] J. Pfau, A. Liapis, G. Volkmar, G. N. Yannakakis, and R. Malaka, "Dungeons & replicants: automated game balancing via deep player behavior modeling," 2020 IEEE Conference on Games (CoG), pp. 431–438, August 2020.
[3] F. Lu, et al., "Fighting game artificial intelligence competition platform," 2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE), pp. 320–323, October 2013.
[4] Y. Zheng, et al., "Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning," 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 772–784, November 2019.
[5] J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslén, "Augmenting automated game testing with deep reinforcement learning," 2020 IEEE Conference on Games (CoG), pp. 600–603, August 2020.
[6] Z. Zhu, K. Lin, and J. Zhou, "Transfer Learning in Deep Reinforcement Learning: A Survey," arXiv preprint arXiv:2009.07888, 2020.
[7] J. Schrittwieser, et al., "Mastering atari, go, chess and shogi by planning with a learned model," Nature, vol. 588.7839, pp. 604–609, 2020.
[8] A. A. Rusu, et al., "Progressive neural networks". arXiv preprint arXiv:1606.04671, 2016.
[9] S. Gamrian, and Y. Goldberg, "Transfer learning for related reinforcement learning tasks via image-to-image translation," International Conference on Machine Learning, pp. 2063–2072, PMLR, May 2019.
[10] D. W. Kim, S. Park, and S. I. Yang, "Mastering Fighting Game Using Deep Reinforcement Learning With Self-play," 2020 IEEE Conference on Games (CoG), pp. 576–583, August 2020.
[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.