FROM VIDEO GAME TO REAL ROBOT: THE TRANSFER BETWEEN ACTION SPACES

Janne Karttunen[*1, 2], Anssi Kanervisto*[*2], Ville Kyrki[3], Ville Hautamäki[2]*

[1] Karelics Oy, Joensuu, Finland
[2] School of Computing, University of Eastern Finland, Joensuu, Finland
[3] School of Electrical Engineering, Aalto University, Espoo, Finland
janne.a.karttunen@gmail.com, {anssk, villeh}@uef.fi, ville.kyrki@aalto.fi

ABSTRACT

Deep reinforcement learning has proven to be successful for learning tasks in simulated environments, but applying same techniques for robots in real-world domain is more challenging, as they require hours of training. To address this, transfer learning can be used to train the policy first in a simulated environment and then transfer it to physical agent. As the simulation never matches reality perfectly, the physics, visuals and action spaces by necessity differ between these environments to some degree. In this work, we study how general video games can be directly used instead of fine-tuned simulations for the sim-to-real transfer. Especially, we study how the agent can learn the new action space autonomously, when the game actions do not match the robot actions. Our results show that the different action space can be learned by re-training only part of neural network and we obtain above 90% mean success rate in simulation and robot experiments.

Index Terms— deep reinforcement learning, transfer learning, sim-to-real, reality gap, action space transfer

1. INTRODUCTION

When it comes to training robots to solve a given task with reinforcement learning, one feasible way to do so is by training the policy in a simulation and then using transfer learning [1] to learn the final policy on the real-world robot; a *simulation-to-real* or *virtual-to-real* transfer [2]. This way we are not hindered by expensive and slow robotics experiments. However, simulating real world accurately is hard if not impossible, and thus data obtained from simulation may not be directly applicable to real-world robot, a problem termed *reality gap* [2]. To address this, one can try to create as realistic simulation as possible, which requires vast amount of time and is costly.

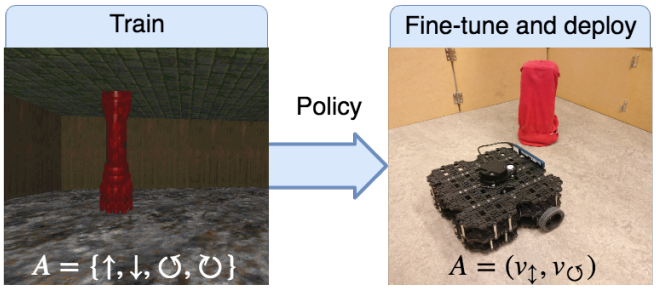

Fig. 1. A policy was trained in a video game with an action space consisting of four discrete actions, and then transferred to a robot with a different action space with small amount of training on the robot.

Video games can act as one such simulation: They can be ran fast, are readily available and are shown to be useful in control and reinforcement learning research [3, 4]. Software packages such as ViZDoom [5] are designed for reinforcement learning. When transferring the trained policies from video games to real-world robots, methods such as domain randomization [2] can be used to narrow the reality gap between visual appearances (observations) of the two worlds. However, as video games are not designed for robotics simulations, they lack the options to tune dynamics to match the real world. This leads to a mismatch between available actions between these two worlds. Differences in action dynamics, such as different rotation speeds, could be manually fine-tuned away, but in the case of completely removed actions this is not possible, e.g. when robot is not able to turn left while original simulation allowed this.

In this work we train a *deep reinforcement learning* (DRL) agent which we adapt to a different action space with as little additional training as possible. We demonstrate effectiveness

* Equal contribution. This research was partially funded by the Academy of Finland (grant #313970) and Finnish Scientific Advisory Board for Defence (MATINE) project #2500M-0106. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp and V GPUs used for this research.

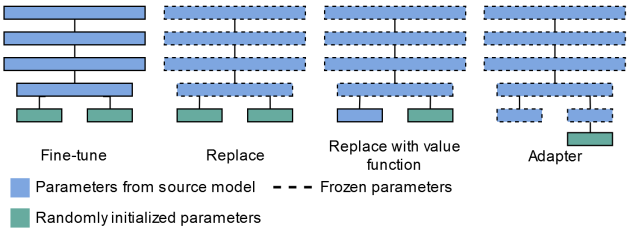

Fig. 2. Overview of the different training methods for moving pre-trained neural networks to new action spaces. Boxes represent different layers of a network with left head being value estimation and right being policy or state-action values.

of this method by transferring agent from a crude simulation (video game "Doom", 1993) to a real robot, where the task is the same, environment shares visual similarities, but the action space differs. We conduct experiments with semantically similar action space where the agent can execute similar actions as previously via new action space. We also experiment by removing possible actions from target action-space, effectively hindering agent's capabilities.

2. ACTION SPACE TRANSFER IN REINFORCEMENT LEARNING

Our work is closely related to experiments conducted by Rusu et al. 2016 [6], but here we focus on transfer between action-spaces and not tasks. Video games are a popular benchmarks in reinforcement learning scheme [7, 5, 8, 9], and video game engines have been used for robotics experiments [10, 11, 12]. Our work differs by using a video game as a simulator for robotics experiments successfully, despite the game was not designed for such purpose. We use methods from previous work on simulation-to-real transfer to overcome the visual reality gap [11, 2, 13], while our contribution lies in bridging the reality gap in action spaces.

The action space transfer can be done with neural networks by replacing the final (output) layer to fit the new action space. If randomly initialized, this final layer requires some training in the target domain to produce useful actions. At this point we have multiple choices how this training should be done. We opted for four similar and simple methods for our study (see Figure 2). These methods are similar to baseline methods in [6], but here we transfer between action spaces rather than tasks.

Fine-tuning Target model uses source model's parameters as initial values, and begins training from there, fine-tuning the previously learned parameters [1]. This is known to be prone to *catastrophic forgetting* [14], where neural network "forgets" the good parameters learned in the previous environment, and thus may not perform as well as expected.

Replace We can avoid catastrophic forgetting by not updating some of the neural network parameters at all ("freezing"). We freeze all layers except the output layers, since we assume similar dynamics and visual appearance from the two environments, allowing the re-use of features of penultimate layer.

Replace with pre-trained value function Value of a state depends on the policy, which depends on the action-space. However, with our assumption of same task and similar environment dynamics between environments, the learned value function could serve as a good initial point in the target environment. We do not freeze the value layer to allow it to adapt to the new policy.

Adapter Instead of updating parameters of the source network, we keep them all fixed and learn a mapping from source actions to new actions, essentially learning which action in source environment matches an action in the target environment. Similar method has been used successfully with policy transfer from one domain to another [15]. We implement this by adding a fully-connected layer which maps old actions to new actions.

3. EXPERIMENTS

3.1. Experimental setup

Agent's task is to navigate to a red goal pillar in a simple room without obstacles, using visual input as the observation (see Figure 1). It starts each episode from the center of the room, facing to a random direction and receives positive reward 1 for reaching the goal and negative reward -1 if episode times out after 1000 environment steps. Agent chooses an action every 10 environment steps (frameskip) and receives a color image of size 80×60 to decide the action. The image is the green channel of a RGB image to highlight the goal.

We use two RL learning algorithms for our experiments: *deep Q-learning* (DQN) [3] and *proximal policy optimization* (PPO) [16]. DQN is selected as it is known to be sample efficient, thanks to its off-policy learning and replay memory. Experiments with PPO are included for its applicability to continuous action spaces and for its closer connection to optimizing policy directly. With DQN, we use double Q-learning [17] and dueling architecture [18] to obtain the state-value function. We use implementations from stable-baselines [19]. Both learning algorithms use network described in Mnih et. al. 2015 [3].

To find suitable exploration strategy, we performed hyper-parameter search for the exploration parameters during action space transfer with *replace* method. For DQN's ϵ-greedy policy we anneal chance of a random action from 1.0 to 0.02 over first 7500 agent steps (searched over interval $[500, 25000]$). For PPO we tested entropy weight coefficients in interval $[10^{-6}, 1]$ and selected 10^{-3} for further experiments. Code and video of the results are available in GitHub [1].

[1] https://github.com/jannkar/doom_actionspace

3.1.1. *Source environment*

For the source environment, we use ViZDoom [5] platform, which is based on the Doom (1993) video game. The agent's action space consists of four different actions; move forward, move backward, turn left and turn right.

We apply domain randomization to ensure that policy can be transferred to a robot [2]. We randomize textures of the walls, ceilings and floors (68 different textures), agent's field of view (50 to 120 degrees horizontally), height of the player and head-bobbing strength. We also add small amount of white- and Gaussian noise on the image, and finally apply random amount of gamma-correction (0.6 to 1.5).

3.1.2. *Target environment*

We start experiments by transferring the agent between two Doom environments. In these simulation-to-simulation experiments ("*sim-to-sim*") the environment uses unseen textures and action-space to the agent. For PPO experiments the new action space consists of two continuous values, one for forward/backward velocity and another for rotation speed. For DQN we define discretized action space of 24 actions, each action being some combination of forward/backward speed and left/right turning, similar to the continuous action space.

The results of best method in sim-to-sim experiments are then verified with real-world experiments by transferring agent to a Turtlebot 3 Waffle robot with a RGB camera. The task is same with a similar environment, with major difference being the lack of roof (Figure 1) and larger number of environment steps per one action (15 versus original 10).

3.2. Source models

We trained three separate source models with both DQN and PPO in Doom environment, by repeating the training runs. All the following experiments were repeated over these source models, since the source model parameters can affect on final performance of the transfer. All three source models of both algorithms learned to solve the task. The three DQN source models reached a $90 - 95\%$ testing success rate and mean episode length of $18.57 - 27.54$ steps, while PPO reached $98 - 100\%$ success rate with mean of $10.45 - 14.01$ steps per episode.

3.3. Sim-to-sim experiments

Freezing most of the network (*replace* method) performs most reliably with DQN, reaching $99.2 - 100\%$ success rate (Table 1) with the final policy. The task was solved efficiently already at around $15,000 - 20,000$ steps. Compared to learning the task from scratch, which took approximately $30,000 - 40,000$ steps for efficient policy, *replace* method reduces the learning time to almost half. Other methods, *adapter*, *fine-tune* and *replace with value function* performed even worse than training from scratch (Figure 3). Interestingly, large variance is detected between source model: For source model 2 all tested methods worked well but for models 1 and 3 only *replace* method gave stable performance.

Fig. 3. Results of transferring three source models to a new action space with different transfer methods, source models (columns) and learning algorithms. Lower is better. Each line is an average over five repetitions. We omitted variance for visual clarity. Rows share same baseline result (no transfer learning, "Scratch").

PPO is not as sample efficient algorithm as DQN, and as such it did not learn the task reliably before $1,000,000$ steps. The *replace* method again resulted in robust transfer between action spaces and the performance even improved slightly on average when the value function was loaded with it. However, the task was not solved as efficiently as with the DQN, when considering the episode length.

In light of these results, *replace* method has the stablest results among tested methods. Interestingly, Rusu et. al. (2016) [6] found this method least effective in task-transfer scenario, while our results find it most promising in action-space transfer.

3.4. Robot experiments

Finally, we validate sim-to-sim experiments on a Turtlebot robot. Based on the previous results, we chose DQN algorithm with the *replace* method for these experiments. We selected DQN model 3 as the source model, due to its fastest learning in the *replace* method experiment. The agent was trained for $20,000$ steps or until the agent's performance did not increase. Turtlebot takes approximately two actions per second, which translated to $4 - 5$ hours of wall-clock time per one experiment, including the time to reset the episode.

Table 1. Mean and standard deviation of final success rates from transferring three different source models with different methods. Each result is based on average performance over last 10% training episodes and averaged over five repetitions. Experiments with average success rate above 90% are highlighted.

Method	Learning algorithm and source model					
	DQN			PPO		
	1	2	3	1	2	3
Fine-tune	58.9 ± 28.7	**100 ± 0.0**	54.1 ± 26.7	46.8 ± 28.4	31.7 ± 14.6	52.7 ± 21.2
Replace	**99.9 ± 0.5**	**100 ± 0.0**	**99.2 ± 3.0**	**95.2 ± 6.7**	**98.0 ± 2.3**	**95.9 ± 5.4**
Replace w/ value func.	60.1 ± 33.6	**95.7 ± 6.5**	86.8 ± 9.5	**98.0 ± 2.3**	**96.1 ± 3.5**	**98.0 ± 1.3**
Adapter	66.3 ± 31.4	**100 ± 0.0**	88.9 ± 19.0	29.7 ± 8.7	30.0 ± 7.4	31.6 ± 7.8
Scratch		**99.5 ± 1.1**			56.0 ± 25.4	

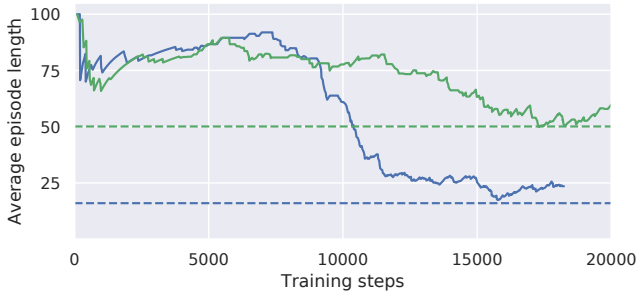

Fig. 4. Length of episode of the two Turtlebot experiment runs in different colors. Lower is better. Curves are averaged with rolling average of 50 steps. Dashed line represents the performance of the best model obtained during training.

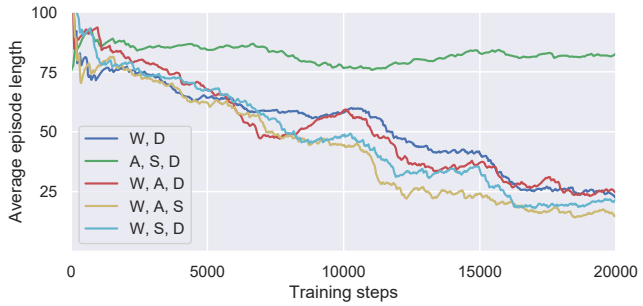

Fig. 5. Agent's performance with action space where some of the previous actions are removed. Episode length is measured in time steps and the letters in labels correspond to button presses of each action (W = move forward, A = turn left, S = move backward, D = turn right).

We conducted two training runs with the Turtlebot. The model of first run had success rate of 80% and mean episode length 50.1 with the best model (Figure 4). Second training run had mean success rate of 100% and episode length 16.0. Subjectively, the first model was attracted by the goal but repeatedly chose to reverse away from the goal. The second agent rotated in place until red pillar appeared to its field-of-view and began moving towards to goal, doing small fine-adjustments to stay on correct path and utilizing newly available actions appropriately.

3.5. Experiments with removed actions

In practical situations, changes to action space may occur from faulty or invalid hardware, preventing from executing specific actions. To study how our approach would perform in this situation, we conducted sim-to-sim transfer using DQN source model 3 with replace method with the same setup as previously. However, now one or two of the original four actions were disabled, so the agent had to find a different way to complete the task.

The results show that even if one of the turn actions or move backward action was removed, the agent still learned the new action space robustly in 20,000 training steps (Figure 5). This result suggests that with the replace method, the agent can learn the task so that it is able to adapt to action space where some of the initial actions are removed. Only when the action "move forward" was removed, the agent could not learn the task. This was expected, as agent does not have memory it is unable to navigate by reversing.

4. CONCLUSIONS

In this work we show how freezing most of the pre-trained neural network parameters can be used to effectively transfer a policy from a video game to a robot, despite the differences in action spaces between these two environments. We trained a policy on raw image data to solve a simple navigation task in Doom video game, and then successfully transferred it to a robot with a different action space where it was able to complete the same task with relatively little amount of training, including when number of available actions was reduced. These

methods have promise to utilize crude simulations like video games to train policies for robots with different physical properties.

The future work could extend the present work in terms of learning complicated abstract task in video game and then transferring to the vastly different action space structure in the physical robot. We also plan to study if Bayesian methods could be used to find good priors for the network parameters, to further speed up the learning process of new action space.

5. REFERENCES

[1] Matthew E Taylor and Peter Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.

[2] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al., "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[5] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.

[6] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[7] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, jun 2013.

[8] Gabriel Synnaeve, Nantas Nardelli, Alex Auvolat, Soumith Chintala, Timothée Lacroix, Zeming Lin, Florian Richoux, and Nicolas Usunier, "Torchcraft: a library for machine learning research on real-time strategy games," *arXiv preprint arXiv:1611.00625*, 2016.

[9] Anssi Kanervisto and Ville Hautamäki, "Torille: Learning environment for hand-to-hand combat," *arXiv preprint arXiv:1807.10110*, 2018.

[10] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*. Springer, 2018, pp. 621–635.

[11] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al., "Learning dexterous in-hand manipulation," *arXiv preprint arXiv:1808.00177*, 2018.

[12] Joshua Greaves, Max Robinson, Nick Walton, Mitchell Mortensen, Robert Pottorff, Connor Christopherson, Derek Hancock, and Jayden Milne David Wingate, "Holodeck: A high fidelity simulator," 2018.

[13] Rika Antonova, Silvia Cruciani, Christian Smith, and Danica Kragic, "Reinforcement learning for pivoting task," *CoRR*, vol. abs/1703.00472, 2017.

[14] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv preprint arXiv:1312.6211*, 2013.

[15] Fernando Fernández and Manuela Veloso, "Policy reuse for transfer learning across tasks with different state and action spaces," in *ICML Workshop on Structural Knowledge Transfer for Machine Learning*. Citeseer, 2006.

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[17] Hado Van Hasselt, Arthur Guez, and David Silver, "Deep reinforcement learning with double q-learning.," in *AAAI*, 2016, pp. 2094–2100.

[18] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas, "Dueling network architectures for deep reinforcement learning," *International Conference on Machine Learning*, 2016.

[19] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu, "Stable baselines," `https://github.com/hill-a/stable-baselines`, 2018.