

Distilling Reinforcement Learning Tricks for Video Games

Anssi Kanervisto*

School of Computing

University of Eastern Finland

Joensuu, Finland

anssk@uef.fi

Christian Scheller*

Institute for Data Science

University of Applied Sciences

Northwestern Switzerland

Windisch, Switzerland

christian.scheller@fhnw.ch

Yanick Schraner*

Institute for Data Science

University of Applied Sciences

Northwestern Switzerland

Windisch, Switzerland

yanick.schraner@fhnw.ch

Ville Hautamäki

School of Computing

University of Eastern Finland

Joensuu, Finland

villeh@uef.fi

Abstract—Reinforcement learning (RL) research focuses on general solutions that can be applied across different domains. This results in methods that RL practitioners can use in almost any domain. However, recent studies often lack the engineering steps (“tricks”) which may be needed to effectively use RL, such as reward shaping, curriculum learning, and splitting a large task into smaller chunks. Such tricks are common, if not necessary, to achieve state-of-the-art results and win RL competitions. To ease the engineering efforts, we distill descriptions of tricks from state-of-the-art results and study how well these tricks can improve a standard deep Q-learning agent. The long-term goal of this work is to enable combining proven RL methods with domain-specific tricks by providing a unified software framework and accompanying insights in multiple domains.

Index Terms—reinforcement learning, machine learning, video games, artificial intelligence

I. INTRODUCTION

Reinforcement learning (RL) is a form of machine learning that builds on the idea of learning by interaction and learns solely from a numerical reward signal. Due to its generality, reinforcement learning has found applications in many disciplines and achieved breakthroughs on many complex tasks, including modern video games [1], [2].

Alas, such feats often require a large amount of training data to learn. For example, in the case of the “OpenAI Five” agent, it took roughly 40,000 years of in-game time to train the agent [2]. Instead of more training, one can incorporate human domain-knowledge of the task and the environment to aid the training of the agent (“tricks”), which are especially prominent in the RL competitions [3]–[5]. As these tricks are domain-specific, they are rarely covered by academic publications. Nevertheless, their prevalence across domains hints that they are a necessary step in practical applications. If a trick works, it also tells something about the structure of the task and how it could be solved in a more general fashion.

To broaden the knowledge on these tricks, we summarize the approaches used by participants in RL competitions and

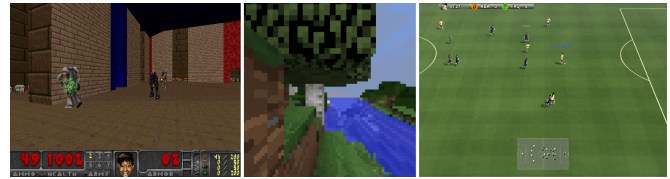


Fig. 1. Environments used in the experiments: ViZDoom Deathmatch, MineRL ObtainDiamond, and Google Research Football environment, in this order. Pictures of ViZDoom and MineRL depict image inputs the agent receives, while in Football agent receives direct information, such as the location of the players.

state-of-the-art results. We categorize these tricks and then apply them on three complex RL environments to conduct preliminary ablation experiments to study how they affect performance.

II. REINFORCEMENT LEARNING TRICKS IN THE WILD

In this work, a “trick” refers to a technique outside the RL algorithm that improves the training results. We treat the RL algorithm as a black-box function that takes in observations and outputs actions, and given the reward signal learns to pick actions with higher rewards. Tricks in this work take this black-box function and build up around it to create an agent. This means we do not include the choice of RL algorithm, code-level improvements (e.g. normalizing values), or network architecture choices, which can lead to improvements [1], [11]. We also do not include observation or action space shaping, as these have been explored in previous work [17]. We emphasize that we are interested in any reported tricks which improve the results, regardless of the lack of generalizability.

Table I categorizes the tricks described in the reviewed works. Some of these are common knowledge, like reward shaping (RS) [18], but others are less explored. For example, Lample et al. (2017) [6] assist the learning agent by modifying the actions (MA) to move the crosshair on top of the enemies before shooting, but not during evaluation. This action is added to the agent’s learning buffers, from which it learns to shoot by itself. Another common trick is to hardcode the agent’s behavior with scripted actions (SA), which can support the agent with easy-to-script solutions [8]. Augmenting scripted

*Equal contribution, alphabetical ordering.

©2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

TABLE I
SUMMARY OF DOMAIN-SPECIFIC METHODS USED TO SUPPORT THE TRAINING OF RL AGENTS.

Trick name and abbreviation	Description	References
Reward sharpening (RS)	Modify reward function for a denser reward	[1]–[4], [6]–[9]
Manual curriculum learning (CL)	Increase difficulty gradually	[2], [3], [6], [8], [10]–[12]
Manual hierarchy (MH)	Split task into subtasks by manual rules	[3], [4], [6], [13]–[16]
Modified actions (MA)	Modify agent’s actions while learning	[3], [6]
Scripted actions (SA)	Augment the agent with hardcoded actions	[2], [3], [8]

behavior with RL has also been shown to be an effective strategy [16]. Another common solution to sparse reward environments is to use curriculum learning to initially start with an easy problem and gradually increase the difficulty [19]. This can be done by increasing the health and difficulty of the opponents [8] or by gradually adding more tools to the agent’s disposal [11].

Sometimes the task can be split into multiple subtasks, and assigning one learning agent to each has yielded significant performance gains [3], [13], [16]. This is credited to the simplicity of the subtasks (easier for agents to learn) and to catastrophic forgetting, where learning skills for new tasks might interfere with skills learned for a previously learned task. In most cases, one or more of these tricks were combined into one. For example, a scripted logic decides which of the subtask agents should take control next [3].

III. EXPERIMENTAL SETUP

With the above categorization of the tricks, we pick the most prominent ones, implement them along with a deep Q-learning (DQN) [20] RL agent and study their performance in three different domains: Doom, Minecraft, and a Football environment. We chose these environments as we, the authors, are experienced in using these environments, and are familiar with their challenges. We spread experiments over multiple environments to gain more general insights rather than domain-specific knowledge.

We choose DQN as it is an off-policy RL algorithm, meaning we can modify the actions for learning. Also, DQN’s performance in discrete action spaces is well documented and reputable implementations of it exist. We use the implementation from the stable-baselines3 repository [21]. The source code used for the experiments is available at <https://github.com/Miffyli/rl-human-prior-tricks>.

A. Reinforcement learning agent

By default, the DQN agents have a replay buffer of one million transitions, start learning after 10K steps and update the target network after 10K training steps. If the agent input is an RGB image, it is processed with a convolutional layer network of the original DQN paper [20]. The agent is updated at every step to emphasize sample-efficient learning. For exploration, we use an ϵ -greedy strategy, where agents pick a random action with a small probability. In ViZDoom and MineRL the chance of a random action is linearly decayed from 100% to 5%/10% in the first 10%/1% training steps, respectively. In

GFootball the probability is fixed at 1%. Other settings are taken from default settings of stable-baselines3 [21].

B. ViZDoom Deathmatch

The ViZDoom learning environment [22] provides an interface to train RL agents in the Doom environment, based on visual input (image pixels) similar to what a human player would see (see Fig. 1 for an example). We use the “Deathmatch” scenario where the agent is rewarded for killing continuously spawning enemies. The scenario also includes weapons and pickups for health and armor. Since killing an enemy takes multiple shots, especially with the starting weapon (a pistol), the reward is sparse, and with a crude exploration strategy the RL agent has a hard time learning the task (see results in Section IV).

The same challenges arose in the ViZDoom competitions [3], where instead of game enemies, the agents had to play against each other. Competition participants employed various tactics, most notably MH of agents [6], [13], CL by starting with weaker enemies [8] and modifying agent’s actions (MA) to optimal ones during training [6]. When combined and tuned correctly, this turns basic RL algorithms into capable agents in these competitions, approaching human-level performance but not quite surpassing it [3].

We implement three of these tricks and try them in the deathmatch scenario: RS, MA, and MH. With RS, the agent is given a small reward (0.1) for hitting an enemy or when picking up pick-ups. With MA, we force the agent to shoot if an enemy is under the crosshair (used only during training). In MH we create two learning agents, one for navigation and one for combat, where the combat agent is activated when an enemy is visible on the screen, otherwise, the navigation agent plays. These two agents are separated with their own replay buffers for training. When combined with RS and MA, both agents use the same RS and the shooting agent also uses MA.

Agents are provided with an RGB image of size 80x60, and on each step can choose one of 66 actions, which consist of different combinations of buttons allowed (turning left/right, moving forward/backward/left/right, shooting, selecting next/previous weapon, turning 180 degrees, and enabling sprinting). The agent chooses an action every four environment steps. During the evaluation, we always pick the action with the highest Q-value.

C. MineRL ObtainDiamond

MineRL ObtainDiamond [23] is an environment built in Minecraft, where the goal is to obtain a diamond. The player

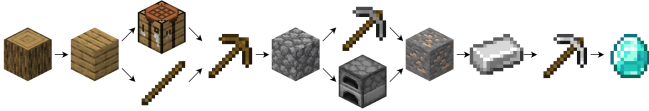


Fig. 2. MineRL ObtainDiamond item hierarchy. Subsequent items yield exponentially higher rewards.

starts with no item in a randomly generated level. To mine a diamond, the player must first collect and craft a list of prerequisite items that hierarchically depend on each other (see Fig. 2). The agent is rewarded the first time they obtain an item in this hierarchy tree. The game ends when the player obtains a diamond, dies, or after a maximum of 15 minutes in-game time.

The MineRL ObtainDiamond task poses multiple challenges for reinforcement learning agents. The sparsity of the reward signal and the complexity of the state- and actions-space make it a particularly hard exploration problem. Furthermore, random level generation means that agents must generalize across a virtually infinite number of levels.

To address these challenges, we use three tricks: RS, MH, and SA. With RS, the agent receives a reward each time a needed item is obtained, until the required amount is reached. This is opposed to the default reward function that only awards an item once when it is collected for the first time. As for MH, we train one agent per intermediate item. This type of hierarchical solution has had high performance in the MineRL competitions [5]. Finally, some of the items require a specific sequence of crafting actions, which can be easily scripted. With SA, we replace sub-policies for crafting items with fixed policies that follow a pre-defined action sequence.

Agents perceive the environment through an RGB render image of size 64x64 and a list of the items currently equipped and in the inventory. We simplify the action space of all agents following [17], by discretizing camera actions, dropping unnecessary actions (these vary between agents) and flattening multi-discrete actions to discrete actions. Agent picks an action once every eight frames.

D. The Football Project (GFootball)

Google Research Football [24] is a 3D RL environment that provides a stochastic and open-source simulation. The engine implements an 11 versus 11 football game with the standard rules, including goal kicks, free kicks, corner kicks, yellow and red cards, offsides, handballs, and penalty kicks. Each game lasts 3000 frames. In the 11 versus 11 football game, the RL agent faces a built-in scripted agent whose skill level can be adjusted with a difficulty parameter between 0 and 1. The agent controls one player at a time, the one in possession of the ball if attacking or the one closest to the ball if defending. The agent is rewarded with +1 when scoring and -1 when conceding a goal.

Playing football is challenging as it requires a balance of short-term control, learned concepts such as tackling, and high-level strategy. The agents face a sparse reward signal,

TABLE II
FINAL PERFORMANCES OF THE MINERL AGENTS, EVALUATED ON 200 EPISODES, AND AVERAGED OVER FIVE TRAINING RUNS. “BEST” IS THE AVERAGE PERFORMANCE OF THE BEST AGENT ACROSS THE FIVE RUNS. “MAX” IS THE HIGHEST PER-EPISODE SCORE REACHED WHILE EVALUATING THE AGENTS FROM THE FIVE RUNS.

Experiment	Mean	Best	Max
RL	3.8 ± 1.0	5.4	35
RL+RS+MH	4.5 ± 0.7	5.4	99
RL+RS+MH+SA	33.5 ± 5.6	41.4	547

complex action- and observation-spaces, and a stochastic environment.

We approach these challenges in the 11 versus 11 easy stochastic environment with two tricks: MH and CL. With MH we evaluate a simple manual hierarchy with separate sub-policies for situations with and without ball possession. This allows each sub-policy to specialize in attacking or defending independently. In CL we gradually increase the game difficulty parameter as described in [12]. The agents receive a 115-dimensional vector summarizing the game state. The agent can choose one of 19 actions consisting of different football-specific movements like passing, tackling, shooting, and dribbling at every step.

IV. RESULTS AND DISCUSSION

The learning curves are reported in Fig. 3. In ViZDoom, all tricks improve upon the DQN, but only RS, MA and MH combined provide significant improvements. Most notably, MH from an average score of less than one to an average score of three, despite only splitting the task into two separate learning agents.

In the MineRL experiments, MH and RS only slightly improved the expected mean reward compared to DQN. However, the maximum achieved reward increased from 35 to 99 (Table II), which corresponds to two steps in the item hierarchy. SA in combination with RS and MH resulted in the significantly fastest training and best final performance.

In the GFootball environment, Only CL improved the mean reward over the baseline. The use of MH isolated and in combination with CL did not improve the average score, in the latter case, it even led to decreased performance. When using MH the attacking and defending agents have their separate neural networks with no weight sharing. Therefore, each network has fewer updates than the DQN or CL agents, which could cause the performance drop. One could mitigate this problem by implementing the agents with weight sharing and their independent value heads.

In summary, most of the evaluated tricks did lead to improvements over vanilla DQN, with a right combination yielding significantly better results. Out of the individual tricks, manually separating the task into subtasks (MH) yielded the most promising results so far, but with a negative impact on the GFootball environment. This concurs with the original observation that, while these tricks are likely beneficial, the exact effect is highly dependent on the environment.

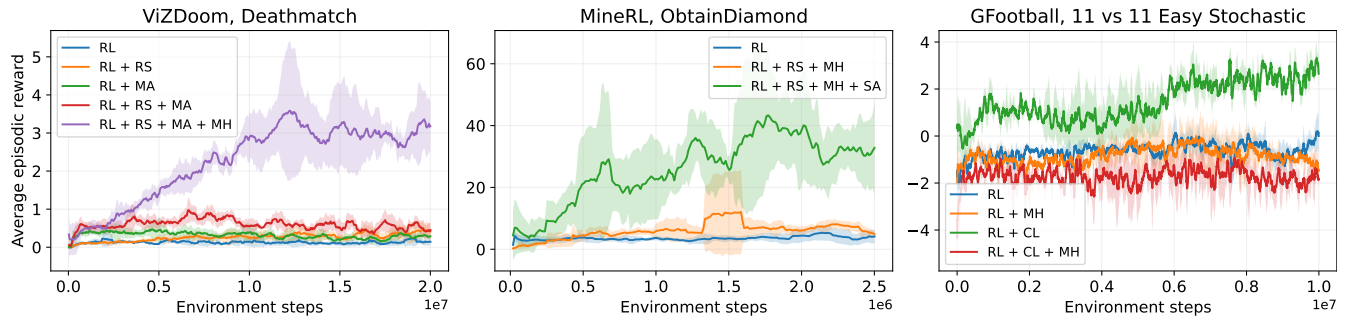


Fig. 3. Training results with different tricks on top of a normal DQN agent (RL). Abbreviations are as in Table I. RL is a vanilla DQN learning agent without tricks. We average results over five training runs for ViZDoom and MineRL, and three for GFootball.

V. CONCLUSIONS AND FUTURE WORK

In this work, we summarized the different domain-knowledge techniques (“tricks”) used by researchers in state-of-the-art results, both in academic publications and competitions. While individual tricks rarely generalize outside the specific environment and task, we find that tricks of the same category (e.g. reward shaping, hierarchy) span various environments and, overall, are reported as significant steps to obtain the reward. We then took the most prominent tricks, implemented them in three different environments along with a DQN agent, and ran preliminary experiments to study their impact on the learning. The results so far indicate that manually splitting the task into subtasks and hardcoding the agent’s actions are very effective tricks to improve the agent’s performance, especially when these are easy to implement.

The future of this work consists of more systematic study and categorization of the tricks, providing more formal definitions for the tricks to understand them better, and finally conducting large-scale experiments over various environments for general information. Along with this, a unified software framework for combining scripted and learned behavior would be a useful tool for easing the use of RL in practical applications. Finally, even though the tricks assume human domain knowledge is available, we believe these large-scale results on them would serve as a useful tool for RL applications and a source of directions for RL researchers.

REFERENCES

- [1] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, “Grandmaster level in starcraft ii using multi-agent reinforcement learning,” *Nature*, pp. 1–5, 2019.
- [2] OpenAI *et al.*, “Dota 2 with large scale deep reinforcement learning,” *arXiv:1807.01281*, 2019.
- [3] M. Wydmuch, M. Kempka, and W. Jaśkowski, “Vizdoom competitions: Playing doom from pixels,” *IEEE Transactions on Games*, vol. 11, no. 3, pp. 248–259, 2018.
- [4] A. Juliani and J. Shih, “Announcing the obstacle tower challenge winners and open source release.” <https://blogs.unity3d.com/cn/2019/08/07/a>, 2019.
- [5] S. Milani, N. Topin, B. Houghton, W. H. Guss, S. P. Mohanty, K. Nakata, O. Vinyals, and N. S. Kuno, “Retrospective analysis of the 2019 minerl competition on sample efficient reinforcement learning,” in *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, PMLR, 2020.
- [6] G. Lample and D. S. Chaplot, “Playing fps games with deep reinforcement learning,” in *AAAI*, 2017.
- [7] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, *et al.*, “Human-level performance in 3d multiplayer games with population-based reinforcement learning,” *Science*, vol. 364, no. 6443, pp. 859–865, 2019.
- [8] Y. Wu and Y. Tian, “Training agent for first-person shooter game with actor-critic curriculum learning,” in *ICLR*, 2017.
- [9] A. Nichol, “Competing in the obstacle tower challenge.” <https://blog.aqnichol.com/2019/07/24/competing-in-the-obstacle-tower-challenge>, 2019.
- [10] A. Dosovitskiy and V. Koltun, “Learning to act by predicting the future,” in *ICLR*, 2017.
- [11] C. Winter, “Mastering real-time strategy games with deep reinforcement learning: Mere mortal edition.” <https://clemenswinter.com/2021/03/24/m>, 2021.
- [12] M. Rychlicki, M. Patek, and M. Mikula, “The football project.” <https://sites.google.com/view/rl-football/singleagent-team>, 2020.
- [13] S. Song, J. Weng, H. Su, D. Yan, H. Zou, and J. Zhu, “Playing fps games with environment-aware hierarchical reinforcement learning,” in *IJCAI*, 2019.
- [14] V. P. Patil, M. Hofmarcher, M.-C. Dinu, M. Dorfer, P. M. Blies, J. Brandstetter, J. A. Arjona-Medina, and S. Hochreiter, “Align-rudder: Learning from few demonstrations by reward redistribution,” *arXiv preprint arXiv:2009.14108*, 2020.
- [15] A. Skrynnik, A. Staroverov, E. Aitygulov, K. Aksenov, V. Davydov, and A. I. Panov, “Forgetful experience replay in hierarchical reinforcement learning from demonstrations,” *arXiv preprint arXiv:2006.09939*, 2020.
- [16] J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslén, “Augmenting automated game testing with deep reinforcement learning,” in *CoG*, 2020.
- [17] A. Kanervisto, C. Scheller, and V. Hautamäki, “Action space shaping in deep reinforcement learning,” in *CoG*, 2020.
- [18] A. Y. Ng, D. Harada, and S. Russell, “Policy invariance under reward transformations: Theory and application to reward shaping,” in *ICML*, 1999.
- [19] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *ICML*, 2009.
- [20] 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.
- [21] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, “Stable baselines3.” <https://github.com/DLR-RM/stable-baselines3>, 2019.
- [22] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, “Vizdoom: A doom-based ai research platform for visual reinforcement learning,” in *CIG*, 2016.
- [23] W. H. Guss, C. Codeł, K. Hofmann, B. Houghton, N. Kuno, S. Milani, *et al.*, “The minerl competition on sample efficient reinforcement learning using human priors,” *arXiv:1904.10079*, 2019.
- [24] K. Kurach, A. Raichuk, P. Stańczyk, M. Zajac, O. Bachem, *et al.*, “Google research football: A novel reinforcement learning environment,” in *AAAI*, 2020.