Action Space Shaping in Deep Reinforcement Learning

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, Swizerland
christian.scheller@fhnw.ch

Ville Hautamäki
School of Computing
University of Eastern Finland
Joensuu, Finland
villeh@uef.fi

***Abstract*—Reinforcement learning (RL) has been successful in training agents in various learning environments, including video games. However, such work modifies and shrinks the action space from the game's original. This is to avoid trying "pointless" actions and to ease the implementation. Currently, this is mostly done based on intuition, with little systematic research supporting the design decisions. In this work, we aim to gain insight on these action space modifications by conducting extensive experiments in video game environments. Our results show how domain-specific removal of actions and discretization of continuous actions can be crucial for successful learning. With these insights, we hope to ease the use of RL in new environments, by clarifying what action-spaces are easy to learn.**

***Index Terms*—video game, reinforcement learning, deep learning, action space, shaping**

I. Introduction

Reinforcement learning [1] has been successfully applied to various video game environments to create human-level or even super-human agents [2]–[7], and show promise as a general way to teach computers to play games. However, these results are accomplished with a significant amount of engineering, including questions like: how should the agent perceive the environment, what should the rewards be and when should the game terminate, just to name a few examples. One of these concerns is the *action space*: how does the agent act in the environment? Do we restrict the number of available actions? Should we simplify it by creating combinations of actions? How do we deal with continuous controls like mouse movement? Intuitively, learning to control a system with more buttons is harder, as the agent has to learn what each of the actions mean. Reducing the number of buttons might ease the learning but comes with the risk of limiting the agent's performance.

Such *transformations* of the action space, which we call *action space shaping*, are prominent in reinforcement learning research and competitions, especially when it comes to video games. Environments like Doom [8] and Minecraft [9] have large action spaces with dozens of buttons, and in related competitions *all top participants* modified the actions [5], [10]. This action space shaping comes in the forms of removing actions, combining different actions into one action and discretizing continuous actions. The goal is to ease the learning for the agent, similar to *reward shaping* [11].

Along with the well-known work on mastering Starcraft II [2] and Dota 2 [3] with reinforcement learning, other games have received similar attention, such as modern first-person shooters [4], [12], Minecraft [9], [13], popular fighting-games like Super Smash Bros [14], other massive online battle arenas (MOBAs) [7] and driving in GTA V [15]. All of these works do action space shaping, either because of limitations of the learning environment or because of the sheer number of actions, *e.g.* in strategy games and MOBAs.

The effect of different action spaces is no stranger to RL research. A large number of possible actions is known to lead to over-optimistic estimates of future rewards [16]. Previous research has addressed this problem by removing unpromising actions [17], or by finding the closest neighbors of promising ones [18]. Other works extended existing methods by adding support for different action spaces [19], [20] or by supporting large, complex action spaces [21]. The work by Delalleau *et al.* [22] shares our focus and mindset, where authors discussed different ways of processing complex action spaces, specifically for video games. However, this and other related works have not included experiments that study specific changes to the action-spaces. We fill this gap by running experiments on various video game environments, testing transformations successfully used in different competitions. Our core research question is "**do these transformations support the training of reinforcement learning agents?**"

By answering this question in the context of video games, we aim to reduce the number of dials to tune when applying RL to games. This is especially useful in the case of new environments, where it is unknown if RL agents can learn to play the game. If an agent fails to learn, it is often unclear if the issue lies in the learning algorithm, the observation space, the rewards, the environment dynamics or the action space. By studying which types of action spaces work and which do not, we hope to remove one of these possibilities, making prototyping and further research easier.

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

978-1-7281-4533-4/20/$31.00 ©2020 IEEE

We start our contribution by summarizing different types of action space shaping done, recognizing common transformations and then verifying their effectiveness with empirical experiments. We start with experiments in a toy-environment, and then move on to Atari, ViZDoom, Starcraft II and Obstacle Tower environments, to cover a broad range of different environments.

II. Action space shaping

When applying RL to a specific task, it is common to use reward shaping [11] to create a denser or more informative reward signal and thus making learning easier (agent's average score increases sooner in the training process). We define action space shaping in a similar manner: modifying the original action space by using transformations, with the intent to make learning easier. By original action space, we refer to the action space provided by the learning environment authors or the one defined by the game. For many computer games, this would be the buttons on a keyboard and the movement of the mouse.

A. Reinforcement learning background

In reinforcement learning we consider agents that interact with environments at discrete time-steps t by taking actions a_t from a set of possible actions $\mathcal{A}$. Each step, an agent receives the current environment state s_t from the set of all possible states $\mathcal{S}$ and a numerical reward r_t. The reward is given by a reward function $\mathcal{R}(s,a) = \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a]$. The goal is then to find a policy $\pi(a \mid s) = \mathbb{P}[a_t = a \mid s_t = s]$ that maximizes the episodic reward in expectation $\mathbb{E}\left[\sum_t r_t\right]$, where expectation goes over the distribution of states and actions induced by the environment and the policy.

B. Types of action spaces

Similar to the action spaces established in the OpenAI Gym [23], we define the fundamental action spaces as follows:

- `Discrete`. Arguably the most used action space, where each action is an integer $a \in \{0, 1, \dots N\}$, where $N \in \mathbb{N}$ represents the number of possibilities to choose an action from. For example, an agent playing the game Pong can choose between either *move up*, *move down* or *stay*.
- `MultiDiscrete`. An extension to `Discrete` space[1], where action $\mathbf{a}$ is a vector of individual discrete actions $a_i \in \{0, 1, \dots N_i\}$, each with possibly different number of possibilities N_i. Arguably, this is closer to natural action spaces. for example, a keyboard is a large `MultiDiscrete` space, where each `Discrete` variable can be either down or up.
- `Continuous`. Action $a \in \mathbb{R}$ is a real number/vector, rather than a discrete choice of many options. The amount of mouse movement [8], [9] or acceleration applied are `Continuous` actions, for example.

These action spaces can then be combined into more complex ones, where one action can consist of mixture of all of them, as described in [22]. A set of keyboard buttons and mouse control could be represented as a combination of `MultiDiscrete` and two `Continuous` actions, one continuous action per mouse movement axis, for example.

`MultiDiscrete` spaces are often treated as independent `Discrete` decisions [22], [24]. Policies for `Continuous` spaces have been implemented in various ways, that come with different advantages and disadvantages, one of which is the bounding of possible actions to a certain range [25], [26]. Put quite simply, support for `Continuous` spaces is often harder to implement correctly than for `Discrete` spaces.

C. Action space shaping in video game environments

Table I summarizes action space shaping done by top-participants of different video game competitions and authors using video game environments for research. In this section, we give an overview of the three major categories of action space transformations found throughout these works.

a) RA *Remove actions:* Many games include actions that are unnecessary or even harmful for the task. In Minecraft [9], [13], the "sneak" action is not crucial for progressing in the game, and therefore is often removed. The action "backward" is also often removed [10], [29]. Otherwise, the agent would waste training time by constantly switching between moving forward and backward, effectively jittering in place rather than exploring the environment. Removed actions maybe set to "always on", which was a popular transformation in the Minecraft MineRL competition, where always executing "attack" helped the agents to learn gathering resources [10], [27], [28].

Reducing the number of actions helps with exploration, as there are less actions to try, which in return improves the sample efficiency of the training. However, this requires domain knowledge of the environment, and it may restrict agent's capabilities.

b) DC *Discretize continuous actions:* Many environments include `Continuous` actions, *e.g.* in the form of mouse movement or camera turning speed. These actions are often discretized, either by splitting them into a set of bins, or by defining three discrete choices: negative, zero and positive. This is especially common with camera rotation, where agents can only choose to turn the camera left/right and up/down at a fixed rate per step [10], [27], [28], [36]. A downside is that this turning rate is a hyper-parameter, which requires tuning. If the rate is too high, the actions are not fine-grained, and the agent may have difficulties in aiming at a specific spot. If too small, it may slow down the learning or lead to sub-optimal behaviour as it takes more steps to aim at a specific target.

c) CMD *Convert multi-discrete actions to discrete:* Especially in ViZDoom [5] and Minecraft [13], it is common to turn `MultiDiscrete` actions into a single `Discrete` action, with all possible combinations of the `MultiDiscrete` actions. Since the resulting action space combinatorially explodes quickly with an increasing `MultiDiscrete` space, this is usually combined with removing some of the actions. This can be either dropping unnecessary actions as described

[1] `Discrete` is a special case of MultiDiscrete, but we separate the two as `Discrete` is used so wildly.

Environment Name	Original action space	Transformation	Transformed action space	Performance	Reference
MineRL	Multi-discrete(2, 2, 2, 2, 2, 2, 2, 2, 7, 8, 5, 8, 3), Continuous(2)	DC RA CMD	Discrete(36)		[1]
		DC RA CMD	Discrete(10)	1st place	[27]
		DC RA CMD	Discrete(216)	2nd place	[10]
		DC RA	Multi-discrete(2, 2, 3, 3, 7, 8, 5, 8, 3, 40)	3rd place	[28]
		DC RA	Multi-discrete(2, 2, 2, 5, 8, 3, 8, 7, 3, 3)	5th place	[10]
Unity Obstacle Tower Challenge	Multi-discrete(3, 3, 2, 3)	RA CMD	Discrete(12)	1st place	[29]
		RA	Discrete(6)	2nd place	[2]
VizDoom (Doom)	38 binary buttons, 5 continuous	RA CMD	Discrete(256)	1st place (Track 2)	[30]
		RA	Discrete(6)	1st place (Track 1)	[31]
Atari	Discrete(18)	RA	Discrete(4 - 18)		[6]
StarCraft II	Multi-discrete	DC RA	Multi-discrete		[2], [21], [32], [33]
Dota 2	Multi-discrete	DC RA	Multi-discrete		[3]
GTA V (car driving only)	Multi-discrete	RA CMD	Discrete(3)		[15]
Torcs	Multi-discrete	RA CMD	Discrete(3)		[15], [34]
DMLab (Quake 3)	Multi-discrete(3, 3, 2, 2, 2), Continuous(2)	RA CMD	Discrete(9)		[35], [36]
Honor of Kings (MOBA)	Multi-Discrete, Continuous	RA CMD	Multi-discrete, Continous		[7]
Little Fighter 2 (lf2gym)	Multi-Discrete(2,2,2,2,2,2,2)	CMD	Discrete(8)		[37]

TABLE I

SUMMARY OF ACTION SPACE SHAPING DONE IN DIFFERENT VIDEO GAME-BASED COMPETITIONS AND LEARNING ENVIRONMENTS. BY "ORIGINAL" SPACE, WE REFER TO ACTION SPACE ORIGINALLY PROVIDED BY THE ENVIRONMENT DESIGNERS. "MULTI-DISCRETE(·)" SHOWS THE NUMBER OF DISCRETE VARIABLES, AND NUMBER OF CHOICES FOR EACH. DC: DISCRETIZE CONTINUOUS ACTIONS, RA: REMOVE ACTIONS, CMD: CONVERT MULTI-DISCRETE TO DISCRETE.

[1] https://github.com/minerllabs/baselines/tree/master/general/chainerrl
[2] https://slideslive.com/38922867/invited-talk-reinforcement-learning-of-the-obstacle-tower-challenge

above, manually selecting the allowed combinations (as done in MineRL [10], [27]) or by limiting maximum number of pressed buttons (as done in ViZDoom [30], [31]).

This transformation is intuitively motivated by the assumption that it is easier to learn a single large policy than multiple small policies, as well as technical limitations of some of the algorithms. For example, methods like Q-learning [6] only work for `Discrete` action spaces. While a modified version of Q-learning exists for `MultiDiscrete` spaces [19], this is not commonly used.

III. EXPERIMENTS

With the major action-space transformations summarized above, we move onto testing if these transformations are truly helpful for the learning process. We do this by training RL agents in a larger variety of environments and comparing the learning process between different action-spaces.

Our main tool for evaluation are learning curves, which show how well agents perform at different stages of training. These show the speed of learning (how fast curve rises), show the final performance (how high curve gets) and if the agent learns at all (if the curve rises). We will use four different games (Doom, Atari, Starcraft II and Obstacle Tower), along with a toy-environment. Source code for the experiments is available at https://github.com/Miffyli/rl-action-space-shaping.

A. Reinforcement learning agent

We use the *Proximal Policy Optimization* (PPO) [38] algorithm for training the agents. We employ the high-quality implementations from stable-baselines [39] and rllib [40], which support various action-spaces. We opt for PPO rather than other recent state-of-the-art algorithms [25], [34] for its simplicity, previous results in the environments used in this work and maturity of their implementations. We do not expect final insights to change between different learning algorithms, as action-space transformations are not part of the algorithm design, but part of the environment. An exception to this are the `Continous` actions, which have multiple ways to implement them, and come with additional parameters to tune [25], [41].

Unless otherwise noted, we will use the following hyperparameters: eight parallel environments, from each of which we collect 256 before running four epochs of updates on the gathered data. Entropy coefficient/weight is set to 0.01, and PPO clipping to 0.2. Network is optimized with Adam [42] with learning rate $2.5 \cdot 10^{-4}$.

B. Get-To-Goal experiments

For rapid experimentation with different action spaces and their effects on the learning performance, we implement a simple reach-the-goal environment. The environment consists of a bounded 2D area, a player and a goal. The game starts with a player and a goal at random locations and ends when the player either reaches the goal (reward 1) or when environment times out (reward 0). Agent receives a 2D vector pointing towards the goal, as well as their current heading as a $(\cos(\phi), \sin(\phi))$ tuple, where $\phi \in [0, 2\pi]$ is the relative rotation angle. We use this environment to test DC by using discrete and continuous variants of the action space.

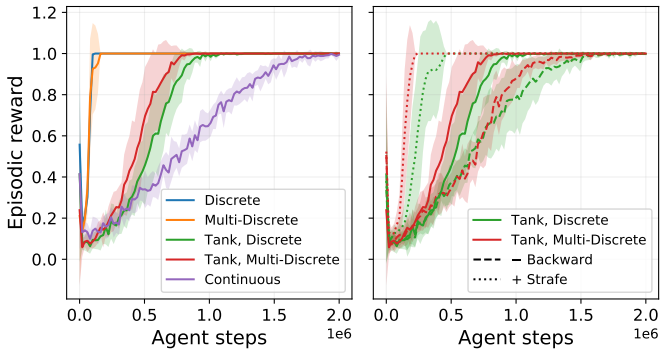

Fig. 1. Learning curves in Get-To-Goal environment, with various action spaces (left), and various buttons available in tank-like controls (right). Averaged over ten repetitions, with shaded region representing the standard deviation.

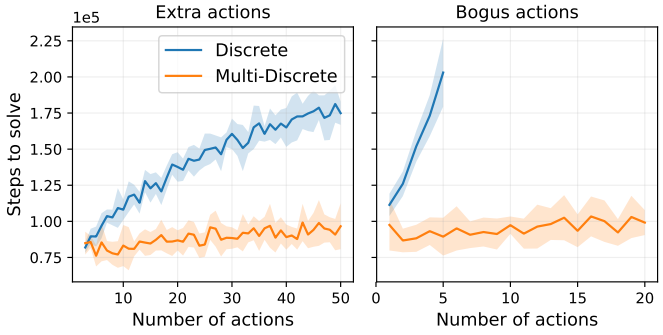

Fig. 2. Results of an increasing number of actions in Get-To-Goal environment. With **Extra actions**, each action moves the player to a different direction, while in **Bogus actions** the extra actions do nothing. Averaged over ten repetitions.

- **Multi-Discrete**: Player can move on two axes with the four buttons *Up*, *Down*, *Left* and *Right* (`MultiDiscrete`).
- **Discrete**: A flattened version of above, where only *one* of the buttons may be pressed at a time, *i.e.* no diagonal movement allowed (`Discrete`).
- **Continuous**: Player specifies the exact direction of the next move with a continuous value, with 0 representing straight up, 90 straight right and 180 straight down (`Continuous`).
- **Tank, Discrete/Multi-Discrete**: Player has a heading ϕ, and it can choose to increase/decrease it (turn left/right), and/or to move forward/backward towards the heading (`Discrete` and `MultiDiscrete` versions).

To study RA and CMD, we train agents with different number of actions available in `Discrete` and `MultiDiscrete` spaces. Each action moves the player to a different direction, equally spaced on a unit circle (**Extra actions**). We also run experiments with additional no-op actions, which do not do anything, to simulate *e.g.* the `USE` action in Doom, which only works in specific scenarios (**Bogus actions**). We also test the effect of "backward" and "strafe" actions, which are often removed in FPS games, by enabling/disabling them in tank-like action spaces. All experiments are run with stable-baselines [39].

Figure 1 (left) shows the results with different action-spaces. Learning with tank-like controls is slower than with the non-tank controls and learning with `Continuous` spaces is the slowest. It should be noted that with rllib [40] we observed similar results, except `Continuous` learned faster than tank-like controls. This indicates that policies for `Continuous` actions are sensitive to the implementation, with the discrete options being robust to this.

Figure 1 (right) shows results with tank-like controls, with and without backward and strafe actions. In both action-spaces, agents learn slower the more actions they have available. This demonstrates how removing actions can be detrimental to the performance (RA). It is evident, that agents learn faster on `MultiDiscrete` spaces than on the `Discrete` alternatives. Figure 2 shows how `MultiDiscrete` spaces are more robust to additional actions. These results demonstrate that there are situtaions where RL agents can profit from `MultiDiscrete` compared to `Discrete` spaces (CMD).

C. Atari experiments

With Atari games, we test RA and CMD transformations. Atari games have been a standard benchmark for DRL algorithms [6], [34], [38], where their action space is defined as a `Discrete` action space. By default, the action space consists of only the necessary actions to play the game (**minimal**). We compare these minimal actions against the **full** action space, and a **multi-discrete** action space, where joystick and fire-button are additional buttons with 9 and 2 options respectively.

We use six games, selected for varying number of actions in the minimal action space. All games have 18 actions in the full space, while *Space Invaders* and *Q*bert* have six, *MsPacman* and *Enduro* have nine and *Breakout* has four actions for minimal space. *Gravitar* uses all 18 actions, and thus do not have minimal spaces. We use the "v4" versions of the environments (e.g. `GravitarNoFrameskip-v4`), which are easier to learn. We use the PPO hyper-parameters from stable-baselines rl-zoo [43].

Figure 3 shows the resulting learning curves. On average, there is no clear difference between the different action-spaces over all games, with *MsPacman* being an exception. Here the **multi-discrete** agent achieved almost one-quarter higher score than other action spaces. Interestingly, in *Enduro*, both action spaces using all buttons out-perform the **minimal** space, despite the fact that full space does not offer any new actions for agent to use. With these results, removing actions (RM) can limit the performance, but overall does not change results. Same applies to converting multi-discrete into discrete (CMD), although in one of the games it did obtain higher performance.

D. Doom experiments

With Doom (1993), we test all of the transformations (DC, RA and CMD). We use the environment provided by

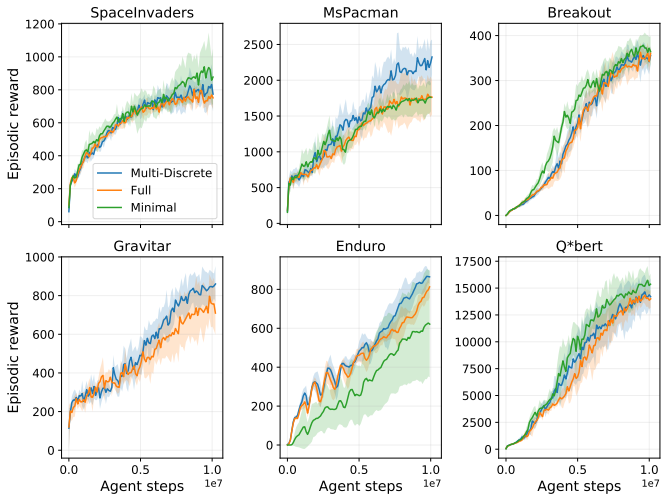

Fig. 3. Results with six Atari games, using **minimal**, **full** and **multi-discrete** actions. Averaged over five training seeds. **Minimal** action-set only includes actions that are available in that specific game. **Multi-discrete** uses all possible actions, but by separating joystick and fire-button into two different discrete variables. *Gravitar* does not have a **minimal** action space.

ViZDoom interface [8], with three different scenarios: *Get-To-Goal*, *Health gathering supreme* (HGS) and *Deathmatch*. The first is an implementation of the toy task described earlier, in the form of a first-person shooter scenario, where the agent is tasked to reach a goal object in a room. The player receives a reward of $+1$ if it reaches the goal, 0 otherwise, including at the timeout-termination after 2100 frames (one minute of game-time). *HGS* and *Deathmatch* are Doom scenarios, where the player has to gather medkits in a maze to survive (HGS) and fight against randomly spawning enemies (Deathmatch). We modify the Deathmatch scenario to give a $+1$ reward per kill. We also make enemies weaker, so they die from one shot. Otherwise, the scenario is too difficult to learn by a PPO agent in the given time. We test four sets of buttons (RA):

- **Bare-Minimum**. The only allowed buttons are moving forward, turning left and attack (Deathmatch). These are the bare-minimum number of buttons to complete tasks.
- **Minimal**. Same as **Bare-Minimum**, but with an option to turn right. This was a common configuration in *e.g.* the MineRL competition [10].
- **Backward**. Same as **Minimal**, but with the additional option of moving backward.
- **Strafe**. Same as **Backward**, but with the additional options of moving left and right. This corresponds to the original movement controls of Doom.

For each set of buttons, we test five different action spaces: The original `MultiDiscrete` space, where pressing each button down is its own discrete choice, three levels of discretization (CMD) and continuous mouse control (DC). Discretization is done by creating all possible combinations of pressed down buttons, with varying levels of buttons pressed down ($n = 1$, $n = 2$ or all). For continuous actions, **mouse** actions correspond to a `MultiDiscrete` action space, where turning left and right has been replaced with a mouse control (a scalar, representing how much we should turn left or right). This action space is not combined with the bare-minimum button-set.

Observations consist of grayscale (*GetToGoal* and *HGS*) or RGB (*Deathmatch*) image of size 80×60, along with any game-variables enabled in the scenario. Each action is repeated for four frames. All experiments except **mouse** are run using stable-baselines, as only rllib supports mixed action-spaces. Other results are same between rllib and stable-baselines.

Figure 4 shows the results. **Multi-discrete** action space performs as well as discretized versions. Using discrete actions with only one button down is the least reliable out of discrete spaces, as it is not able to reliably solve the easiest environment (CMD). Using continuous actions prevents learning in all but the simplest environment (DC). Increasing number of available actions improves the results in more difficult scenarios (RA).

E. Obstacle Tower experiments

Obstacle Tower [44] is a 3D platformer game with randomly generated levels, designed for reinforcement learning research. Its original action space is defined as a `MultiDiscrete` space, with options to move forward/backward and left-/right, turn left/right and jump. We use this environment to test CMD and RA transformations, by disabling strafing, moving backward or forcing moving forward. Similar to Doom experiments, `Discrete` space is obtained by creating all possible combinations of the `MultiDiscrete` actions. Observations are processed using the "retro" setting of the environment (84×84 RGB images). We use 32 concurrent environments, set entropy coefficient to 0.001 and collect 128 steps per environment per update. These parameters were selected based on the previous experiments with Obstacle Tower environment.

Figure 5 shows the results. There is no significant difference between `Discrete` and `MultiDiscrete` spaces (CMD). The only major difference is with **backward** action: all experiments that allow moving backward have slower learning than the rest. This supports the intuition to remove unnecessary actions, and especially when the action can negate other actions during random exploration (RA).

F. StarCraft II experiments

StarCraft II is a complex multi-player real-time strategy game and a challenging domain for RL algorithms. Particularly interesting for our work is the vast size of the action space. For environment interaction we use the StarCraft II learning environment (SC2LE) [21] that provides a parametric action space, with base actions and action parameters. Base actions describe an intended action such as *move screen* or *build barracks*. Action parameters such as *screen coordinates* or *unit id* modify these base actions. There are hundreds of base actions and depending on the base action and screen resolution up to millions or even billions of possible parameter combinations.

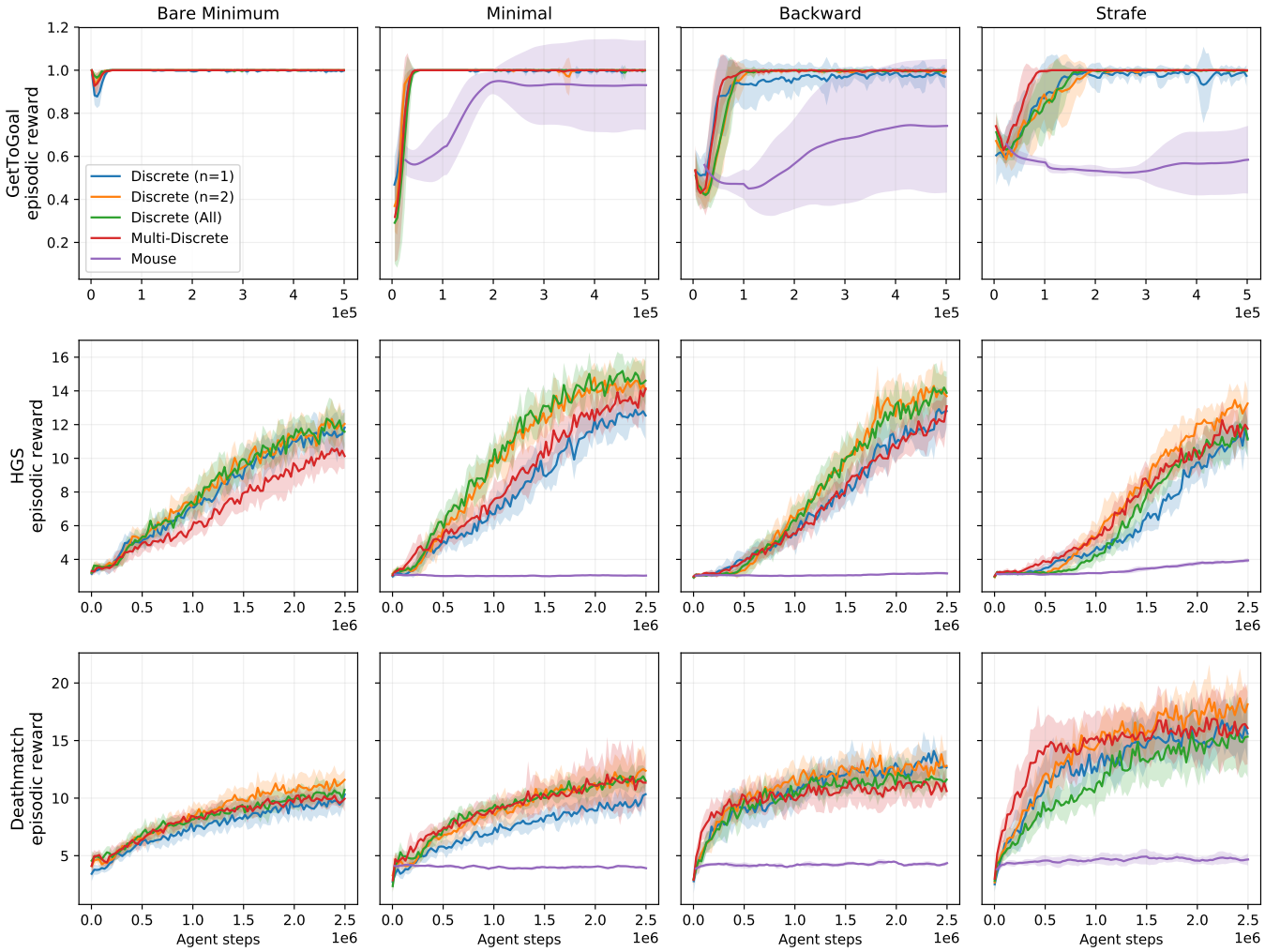

Fig. 4. Results with the ViZDoom environment, with three environments (rows) and four sets of buttons (columns). Number of available actions increases from left to right. Curves are averaged over ten repetitions.

We use four of the seven provided mini-games in our experiments:

- *CollectMinerlShards (CMS)* is a simple navigation task, where the player controls two Marines and must collect randomly scattered mineral shards as quickly as possible.
- In *DefeatRoaches (DR)*, the player controls a small army of Marines and must defeat a small Roach army.
- The goal of *CollecMineralsAndGas (CMAG)* is to build up a working economy and collect as much minerals and vespene gas as possible.
- *BuildMarines (BM)* is targeted at establishing a unit production with the goal of building as many Marines as possible.

To study RA on these mini-games, we test the effects of **Masked** and **Minimal** transformations. We also evaluate auto regressive policies (**AR**), which do not transform the action space but are of interest when dealing with large parametric action spaces:

- **Masked.** In StarCraft II, at any given time, only a subset of all actions is available. To prevent agents from selecting unavailable actions and to ease the learning, base action policies are often masked by setting the probabilities of unavailable actions to zero [21], [32]. We run experiments with agents that are either equipped with action masking or not. Selecting an unavailable action results in a no-op action.
- **Minimal.** Not all actions are required to play the selected mini-games optimally. We evaluate the impact of removing unnecessary actions from the action space. For each mini-game, we define a *minimal* set of base actions required for optimal control.
- **AR.** The choice of optimal action parameters depends on the choices of base action and other action parameters. This is usually addressed by introducing an order of dependency and condition action parameter policies accordingly in an *auto-regressive* manner [2], [21], [32]. To study the effect of *auto-regressive* policies, we run experiments where we condition action parameter policies on the sampled base action. Following [32], we embed sampled base actions into a 16-dimensional continuous

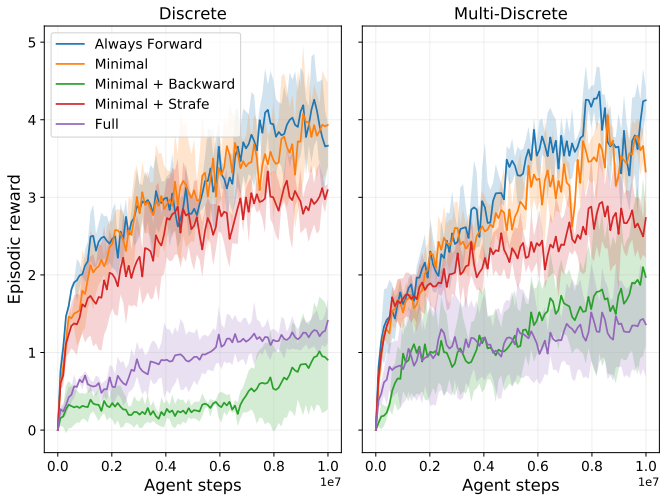

Fig. 5. Results in the Obstacle Tower environment, various button sets and both `Discrete` and `MultiDiscrete` action spaces. **Always Forward** includes actions for turning and jumping. **Minimal** lets agent to choose when to move forward. Curves are averaged over three seeds.

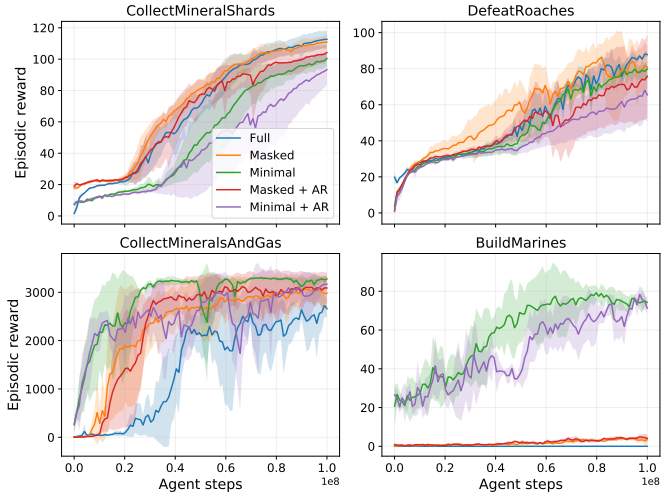

Fig. 6. Results on StarCraft II mini games. **Full** includes all possible actions, **Masked** uses action-masking to remove unavailable actions and **Minimal** extends on this by removing all but necessary actions. We include experiments with autoregressive policies (**AR**), as they are commonly used with Starcraft II environment.

space and feed them as additional input to each of the action parameter policy heads.

For training, we employed IMPALA [36], a distributed off-policy actor critic method. We chose IMPALA over PPO for its scalability to large number of actors and since its strong performance on the SC2LE mini-games [32]. We use ControlAgent architecture and hyper-parameters described in [32], except with fixed learning rate 10^{-4} and entropy coefficient 10^{-3}.

The results are shown in Figure 6. Masking unavailable actions (**Masked**) turned out to be crucial for learning on *BM* and significantly improved performance on *CMAG*. For *CMS* and *DR* we did not see any improvement with masked polices. It is evident that agents trained on *CMS* and *DR* did not profit from minimal action spaces (**Minimal**). In contrast, on *CMAG* and *BM*, minimal action spaces improved the performance even for random policies at the beginning of training. Agents trained on *CMAG* showed much quicker training progress and required less samples to achieve the final performance. On *BM*, we see how RA can lead to significant improvements in the final performance of RL agents. With auto-regressive policies (**AR**), we did not observe significant improvement. We see that, within the limited number of samples, the agents learned only simple, sub-optimal behaviour, where they choose between few actions with distinct parameters. We believe that auto-regressive policies can be beneficial for learning better policies with larger set of actions.

IV. Discussion and Conclusions

Overall, our results support the use of the action space transformations listed in Section II-C, with the exception being converting `MultiDiscrete` to `Discrete` spaces (CMD). Removing actions (RA) can lower the overall performance (Doom), but it can be an important step to make environments learnable by RL agents (SC2, Obstacle Tower). Continuous actions are harder to learn than discrete actions (Get-To-Goal) and can also prevent learning all-together (ViZDoom). Discretizing them (DC) improves performance notably.

As for the `MultiDiscrete` spaces, we did not find notable difference between results with the `MultiDiscrete` and `Discrete` variants. Experiments on the Get-To-Goal task show how `MultiDiscrete` spaces scale well with an increasing number of actions, while `Discrete` do not. In all other environments (ViZDoom, Obstacle Tower, Atari) we observe no significant difference between the two.

In this work, we have formalised the concept of *action space shaping* and summarized its application in the previous RL research. We found three major transformations used throughout such work: removing actions, discretizing continuous actions and discretizing multi-discrete actions. We evaluated these transformations and studied their implications on five environments, which range from simple navigation tasks up to complex 3D first-person shooters and real-time strategy games.

Answering the question presented in introduction, "do these transformations help RL training", removing actions and discretizing continuous actions can be crucial for the learning process. Converting multi-discrete to discrete action spaces has no clear positive effect and can suffer from poor scaling in cases with large action spaces. Our guide for shaping an action space for a new environment is thus as follows:

Start by removing all but the necessary actions and discretizing all continuous actions. Avoid turning multi-discrete actions into a single discrete action and limit the number of choices per discrete action. If the agent is able to learn, start adding removed actions for improved performance, if necessary.

In the future, we would like to extend this work with a more pin-pointed approach on what exactly makes the learning easier, both in theory (*e.g.* exploration vs. number of actions)

and in practice (*e.g.* what kind of actions in games are bad for reinforcement learning). A simpler extension would be to repeat these experiments with more complex games like Minecraft, that have a large variety of mechanics and actions. Specifically, continuous actions serve more attention, along with combinations of different action-spaces. Finally, this work is but a steppingstone in the path towards automated action space shaping. We now know we can ease the learning process significantly with heuristics and manual engineering, and next we would like to see this process automated, *e.g.* as part of the reinforcement learning process.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] 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.

[3] OpenAI et al., "Dota 2 with large scale deep reinforcement learning," *arXiv:1807.01281*, 2019.

[4] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, *et al.*, "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning," *arXiv:1807.01281*, 2018.

[5] 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.

[6] 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.

[7] D. Ye, Z. Liu, M. Sun, B. Shi, P. Zhao, H. Wu, H. Yu, S. Yang, X. Wu, Q. Guo, *et al.*, "Mastering complex control in moba games with deep reinforcement learning," in *AAAI*, 2020.

[8] 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.

[9] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, "The malmo platform for artificial intelligence experimentation.," in *IJCAI*, 2016.

[10] S. Milani, N. Topin, B. Houghton, W. H. Guss, S. P. Mohanty, O. Vinyals, and N. S. Kuno, "The minerl competition on sample-efficient reinforcement learning using human priors: A retrospective," *arXiv:2003.05012*, 2020.

[11] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, 1999.

[12] J. Harmer, L. Gisslén, J. del Val, H. Holst, J. Bergdahl, T. Olsson, K. Sjöö, and M. Nordin, "Imitation learning with concurrent actions in 3d games," in *CIG*, 2018.

[13] W. H. Guss, C. Codel, K. Hofmann, B. Houghton, N. Kuno, S. Milani, S. Mohanty, D. P. Liebana, R. Salakhutdinov, N. Topin, *et al.*, "The minerl competition on sample efficient reinforcement learning using human priors," *arXiv:1904.10079*, 2019.

[14] V. Firoiu, W. F. Whitney, and J. B. Tenenbaum, "Beating the world's best at super smash bros. with deep reinforcement learning," *arXiv:1702.06230*, 2017.

[15] Y. Gao, H. Xu, J. Lin, F. Yu, S. Levine, and T. Darrell, "Reinforcement learning from imperfect demonstrations," in *ICML*, 2018.

[16] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in *Proceedings of the 1993 Connectionist Models Summer School*, 1993.

[17] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, and S. Mannor, "Learn what not to learn: Action elimination with deep reinforcement learning," in *NIPS*, 2018.

[18] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," *arXiv:1512.07679*, 2015.

[19] A. Tavakoli, F. Pardo, and P. Kormushev, "Action branching architectures for deep reinforcement learning," in *AAAI*, 2018.

[20] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *ICML*, 2016.

[21] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv:1708.04782*, 2017.

[22] O. Delalleau, M. Peter, E. Alonso, and A. Logut, "Discrete and continuous action representation for practical rl in video games," in *AAAI Workshop on Reinforcement Learning in Games*, 2019.

[23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[24] A. Kanervisto and V. Hautamäki, "Torille: Learning environment for hand-to-hand combat," in *CoG*, 2019.

[25] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018.

[26] P.-W. Chou, D. Maturana, and S. Scherer, "Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution," in *ICML*, 2017.

[27] A. Skrynnik, A. Staroverov, E. Aitygulov, K. Aksenov, V. Davydov, and A. I. Panov, "Hierarchical deep q-network from imperfect demonstrations in minecraft," *arXiv:1912.08664*, 2019.

[28] C. Scheller, Y. Schraner, and M. Vogel, "Sample efficient reinforcement learning through learning from demonstrations in minecraft," *arXiv:2003.06066*, 2020.

[29] A. Nichol, "Competing in the obstacle tower challenge." https://blog.aqnichol.com/2019/07/24/competing-in-the-obstacle-tower-challenge, 2019.

[30] A. Dosovitskiy and V. Koltun, "Learning to act by predicting the future," in *ICLR*, 2017.

[31] Y. Wu and Y. Tian, "Training agent for first-person shooter game with actor-critic curriculum learning," in *ICLR*, 2017.

[32] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, *et al.*, "Deep reinforcement learning with relational inductive biases," in *ICLR*, 2019.

[33] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, *et al.*, "Alphastar: Mastering the real-time strategy game starcraft ii." https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

[34] 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 *ICML*, 2016.

[35] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, *et al.*, "Deepmind lab," *arXiv:1612.03801*, 2016.

[36] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *ICML*, 2018.

[37] Y.-J. Li, H.-Y. Chang, Y.-J. Lin, P.-W. Wu, and Y.-C. Wang, "Deep reinforcement learning for playing 2.5d fighting games," in *ICIP*, 2018.

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

[39] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines." https://github.com/hill-a/stable-baselines, 2018.

[40] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning," in *ICML*, 2018.

[41] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *ICML*, 2016.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2014.

[43] A. Raffin, "Rl baselines zoo." https://github.com/araffin/rl-baselines-zoo, 2018.

[44] A. Juliani, A. Khalifa, V.-P. Berges, J. Harper, E. Teng, H. Henry, A. Crespi, J. Togelius, and D. Lange, "Obstacle tower: A generalization challenge in vision, control, and planning," in *AAAI Workshop on Games and Simulations for Artificial Intelligence*, 2019.