A Closer Look at Invalid Action Masking in Policy Gradient Algorithms

Shengyi Huang and **Santiago Ontañón** *
College of Computing & Informatics, Drexel University
Philadelphia, PA 19104
{sh3397,so367}@drexel.edu

Abstract

In recent years, Deep Reinforcement Learning (DRL) algorithms have achieved state-of-the-art performance in many challenging strategy games. Because these games have complicated rules, an action sampled from the full discrete action distribution predicted by the learned policy is likely to be invalid according to the game rules (e.g., walking into a wall). The usual approach to deal with this problem in policy gradient algorithms is to "mask out" invalid actions and just sample from the set of valid actions. The implications of this process, however, remain under-investigated. In this paper, we 1) show theoretical justification for such a practice, 2) empirically demonstrate its importance as the space of invalid actions grows, and 3) provide further insights by evaluating different action masking regimes, such as removing masking after an agent has been trained using masking.

Introduction

Deep Reinforcement Learning (DRL) algorithms have yielded state-of-the-art game playing agents in challenging domains such as Real-time Strategy (RTS) games (Vinyals et al. 2017; Vinyals et al. 2019) and Multiplayer Online Battle Arena (MOBA) games (Berner et al. 2019; Ye et al. 2020). Because these games have complicated rules, the valid discrete action spaces of different states usually have different sizes. That is, one state might have 5 valid actions and another state might have 7 valid actions. To formulate these games as a standard reinforcement learning problem with a singular action set, previous work combines these discrete action spaces to a *full discrete action space* that contains available actions of all states (Vinyals et al. 2017; Berner et al. 2019; Ye et al. 2020). Although such a full discrete action space makes it easier to apply DRL algorithms, one issue is that an action sampled from this full discrete action space could be invalid for some game states, and this action will have to be discarded.

To make matters worse, some games have extremely large full discrete action spaces and an action sampled will typically be invalid. As an example, the full discrete action space of Dota 2 has a size of 1,837,080 (Berner et al. 2019),

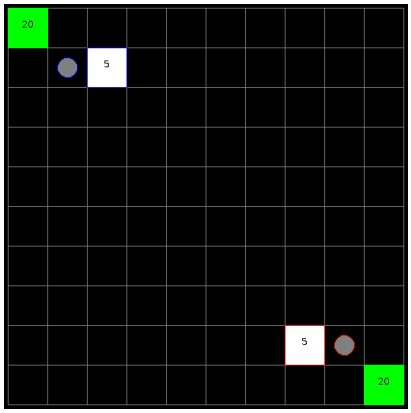


Figure 1: A screenshot of μRTS. Square units are "bases" (light grey, that can produce workers), "barracks" (dark grey, that can produce military units), and "resources mines" (green, from where workers can extract resources to produce more units), the circular units are "workers" (small, dark grey) and military units (large, yellow or light blue), and on the right is the 10×10 map we used to train agents to harvest resources. The agents could control units at the top left, and the units in the bottom left will remain stationary.

and an action sampled might be to buy an item, which can be *valid* in some game states but will become *invalid* when there is not enough gold. To avoid repeatedly sampling invalid actions in full discrete action spaces, recent work applies policy gradient algorithms in conjunction with a technique known as invalid action masking, which "masks out" invalid actions and then just samples from those actions that are valid (Vinyals et al. 2017; Berner et al. 2019; Ye et al. 2020). To the best of our knowledge, however, the theoretical foundations of invalid action masking have not been studied and its empirical effect is under-investigated. In this paper, we take a closer look at invalid action masking in the context of games, pointing out the gradient produced by invalid action masking corresponds to a valid policy gradient. More interestingly, we show that in fact, invalid action masking can be seen as applying a *state-dependent differentiable function* during the calculation of the action probability distribution, to produce a behavior policy. Next, we de-

*Currently at Google

sign experiments to compare the performance of *invalid action masking* versus *invalid action penalty*, which is a common approach that gives negative rewards for invalid actions so that the agent learns to maximize reward by not executing any invalid actions. We empirically show that, when the space of invalid actions grows, invalid action masking scales well and the agent solves our desired task while invalid action penalty struggles to explore even the very first reward. Then, we design experiments to answer two questions: (1) What happens if we remove the invalid action mask once the agent was trained with the mask? (2) What is the agent's performance when we implement the invalid action masking naively by sampling the action from the masked action probability distribution but updating the policy gradient using the unmasked action probability distribution? Finally, we made our source code available at GitHub for the purpose of reproducibility[1].

Background

In this paper, we use policy gradient methods to train agents. Let us consider the Reinforcement Learning problem in a Markov Decision Process (MDP) denoted as $(S, A, P, \rho_0, r, \gamma, T)$, where S is the state space, A is the discrete action space, $P : S \times A \times S \to [0, 1]$ is the state transition probability, $\rho_0 : S \to [0, 1]$ is the initial state distribution, $r : S \times A \to \mathbb{R}$ is the reward function, γ is the discount factor, and T is the maximum episode length. A stochastic policy $\pi_\theta : S \times A \to [0, 1]$, parameterized by a parameter vector θ, assigns a probability value to an action given a state. The goal is to maximize the expected discounted return of the policy:

$$J = \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \gamma^t r_t \right]$$

where τ is the trajectory $(s_0, a_0, r_0, \ldots, s_{T-1}, a_{T-1}, r_{T-1})$, $s_0 \sim \rho_0, s_t \sim P(\cdot|s_{t-1}, a_{t-1}), a_t \sim \pi_\theta(\cdot|s_t), r_t = r(s_t, a_t)$

The core idea behind policy gradient algorithms is to obtain the *policy gradient* $\nabla_\theta J$ of the expected discounted return with respect to the policy parameter θ. Doing gradient ascent $\theta = \theta + \nabla_\theta J$ therefore maximizes the expected discounted reward. Earlier work proposes the following policy gradient estimate to the objective J (Sutton and Barto 2018):

$$\nabla_\theta J = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t \right], G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

Invalid Action Masking

Invalid action masking is a common technique implemented to avoid repeatedly generating invalid actions in large discrete action spaces (Vinyals et al. 2017; Berner et al. 2019; Ye et al. 2020). To the best of our knowledge, there is no literature providing detailed descriptions of the implementation of invalid action masking. Existing work (Vinyals et al. 2017; Berner et al. 2019) seems to treat invalid action masking as an auxiliary detail, usually describing it using only a few sentences. Additionally, there is no literature providing theoretical justification to explain why it works with policy gradient algorithms. In this section, we examine how invalid action masking is implemented and prove it indeed corresponds to valid policy gradient updates (Sutton et al. 2000). More interestingly, we show it works by applying a *state-dependent differentiable function* during the calculation of action probability distribution.

First, let us see how a discrete action is typically generated through policy gradient algorithms. Most policy gradient algorithms employ a neural network to represent the policy, which usually outputs unnormalized scores (logits) and then converts them into an action probability distribution using a softmax operation or equivalent, which is the framework we will assume in the rest of the paper. For illustration purposes, consider an MDP with the action set $A = \{a_0, a_1, a_2, a_3\}$ and $S = \{s_0, s_1\}$, where the MDP reaches the terminal state s_1 immediately after an action is taken in the initial state s_0 and the reward is always +1. Further, consider a policy π_θ parameterized by $\theta = [l_0, l_1, l_2, l_3] = [1.0, 1.0, 1.0, 1.0]$ that, for the sake of this example, directly produces θ as the output logits. Then in s_0 we have:

$$\begin{aligned} \pi_\theta(\cdot|s_0) &= [\pi_\theta(a_0|s_0), \pi_\theta(a_1|s_0), \pi_\theta(a_2|s_0), \pi_\theta(a_3|s_0)] \\ &= \text{softmax}([l_0, l_1, l_2, l_3]) \\ &= [0.25, 0.25, 0.25, 0.25], \\ &\text{where } \pi_\theta(a_i|s_0) = \frac{\exp(l_i)}{\sum_j \exp(l_j)} \end{aligned} \tag{1}$$

At this point, regular policy gradient algorithms will sample an action from $\pi_\theta(\cdot|s_0)$. Suppose a_0 is sampled from $\pi_\theta(\cdot|s_0)$, and the policy gradient can be calculated as follows:

$$\begin{aligned} g_{\text{policy}} &= \mathbb{E}_\tau \left[\nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) G_t \right] \\ &= \nabla_\theta \log \pi_\theta(a_0|s_0) G_0 \\ &= [0.75, -0.25, -0.25, -0.25] \end{aligned}$$

$$(\nabla_\theta \log \text{softmax}(\theta)_j)_i = \begin{cases} (1 - \frac{\exp(l_j)}{\sum_j \exp(l_j)}) & \text{if } i = j \\ \frac{-\exp(l_j)}{\sum_j \exp(l_j)} & \text{otherwise} \end{cases}$$

Now suppose a_2 is invalid for state s_0, and the only valid actions are a_0, a_1, a_3. Invalid action masking helps to avoid sampling invalid actions by "masking out" the logits corresponding to the invalid actions. This is usually accomplished by replacing the logits of the actions to be masked by a large negative number M (e.g. $M = -1 \times 10^8$). Let us use $mask : \mathbb{R} \to \mathbb{R}$ to denote this masking process, and we can calculate the re-normalized probability distribution $\pi'_\theta(\cdot|s_0)$ as the following:

$$\pi'_\theta(\cdot|s_0) = \text{softmax}(mask([l_0, l_1, l_2, l_3])) \tag{2}$$

$$= \text{softmax}([l_0, l_1, M, l_3]) \tag{3}$$

$$= [\pi'_\theta(a_0|s_0), \pi'_\theta(a_1|s_0), \epsilon, \pi'_\theta(a_3|s_0)] \tag{4}$$

$$= [0.33, 0.33, 0.0000, 0.33]$$

where ϵ is the resulting probability of the masked invalid action, which should be a small number. If M is chosen to be

[1] https://github.com/vwxyzjn/invalid-action-masking

sufficiently negative, the probability of choosing the masked invalid action a_2 will be virtually zero. After finishing the episode, the policy is updated according to the following gradient, which we refer to as the *invalid action policy gradient*.

$$g_{\text{invalid action policy}} = \mathbb{E}_\tau \left[\nabla_\theta \sum_{t=0}^{T-1} \log \pi'_\theta(a_t|s_t) G_t \right] \quad (5)$$

$$= \nabla_\theta \log \pi'_\theta(a_0|s_0) G_0 \quad (6)$$

$$= [0.67, -0.33, 0.0000, -0.33]$$

This example highlights that invalid action masking appears to do more than just "renormalizing the probability distribution"; it in fact makes the gradient corresponding to the logits of the invalid action to zero.

Masking Still Produces a Valid Policy Gradient

The action selection process is affected by a process that seems external to π_θ that calculates the mask. It is therefore natural to wonder how does the policy gradient theorem (Sutton et al. 2000) apply. As a matter of fact, our analysis shows that the process of invalid action masking can be considered as a state-dependent differentiable function applied for the calculation of π'_θ, and therefore $g_{\text{invalid action policy}}$ can be considered as a policy gradient update for π'_θ.

Proposition 1. *$g_{\text{invalid action policy}}$ is the policy gradient of policy π'_θ.*

Proof. Let $s \in S$ to be arbitrary and consider the process of invalid action masking as a differentiable function $mask$ to be applied to the logits $l(s)$ outputted by policy π_θ given state s. Then we have:

$$\pi'_\theta(\cdot|s) = \text{softmax}(mask(l(s)))$$

$$mask(l(s))_i = \begin{cases} l_i & \text{if } a_i \text{ is valid in } s \\ M & \text{otherwise} \end{cases}$$

Clearly, $mask$ is either an identity function or a constant function for elements in the logits. Since these two kinds of functions are differentiable, π'_θ is differentiable to its parameters θ. That is, $\frac{\partial \pi'_\theta(a|s)}{\partial \theta}$ exists for all $a \in A, s \in S$, which satisfies the assumption of policy gradient theorem (Sutton et al. 2000). Hence, $g_{\text{invalid action policy}}$ is the policy gradient of policy π'_θ. □

Note that $mask$ is *not* a piece-wise linear function. If we plot $mask$, it is either an identity function or a constant function, depending on the state s, going from $-\infty$ to $+\infty$. We therefore call $mask$ a state-dependent differentiable function. That is, given a vector x and two states s, s' with different number of invalid actions available in these states, $mask(s, x) \neq mask(s', x)$.

Experimental Setup

In the remainder of this paper, we provide a series of empirical results showing the practical implications of invalid action masking.

Table 1: Observation features and action components.

Observation Features	Planes	Description
Hit Points	5	0, 1, 2, 3, ≥ 4
Resources	5	0, 1, 2, 3, ≥ 4
Owner	3	player 1, -, player 2
Unit Types	8	-, resource, base, barrack, worker, light, heavy, ranged
Current Action	6	-, move, harvest, return, produce, attack
Action Components	**Range**	**Description**
Source Unit	$[0, h \times w - 1]$	the location of the unit selected to perform an action
Action Type	$[0, 5]$	NOOP, move, harvest, return, produce, attack
Move Parameter	$[0, 3]$	north, east, south, west
Harvest Parameter	$[0, 3]$	north, east, south, west
Return Parameter	$[0, 3]$	north, east, south, west
Produce Direction Parameter	$[0, 3]$	north, east, south, west
Produce Type Parameter	$[0, 6]$	resource, base, barrack, worker, light, heavy, ranged
Attack Target Unit	$[0, h \times w - 1]$	the location of unit that will be attacked

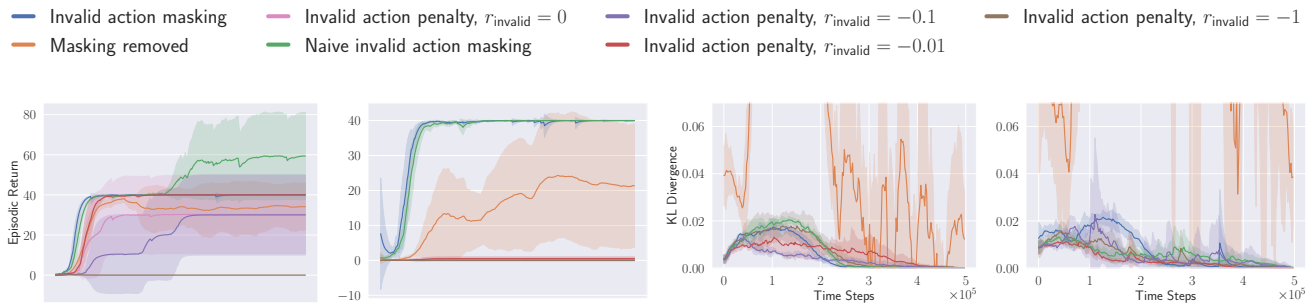
Evaluation Environment

We use μRTS[2] as our testbed, which is a minimalistic RTS game maintaining the core features that make RTS games challenging from an AI point of view: simultaneous and durative actions, large branching factors, and real-time decision-making. A screenshot of the game can be found in Figure 1. It is the perfect testbed for our experiments because the action space in μRTS grows combinatorially and so does the number of invalid actions that could be generated by the DRL agent. We now present the technical details of the environment for our experiments.

- **Observation Space.** Given a map of size $h \times w$, the observation is a tensor of shape (h, w, n_f), where n_f is a number of feature planes that have binary values. The observation space used in this paper uses 27 feature planes as shown in Table 1, similar to previous work in μRTS (Stanescu et al. 2016; Yang and Ontañón 2018; Huang and Ontañón 2019). A feature plane can be thought of as a concatenation of multiple one-hot encoded features. As an example, if there is a worker with hit points equal to 1, not carrying any resources, the owner being Player 1, and currently not executing any actions, then the one-hot encoding features will look like the following:

$$[0, 1, 0, 0, 0], [1, 0, 0, 0, 0], [1, 0, 0],$$
$$[0, 0, 0, 0, 1, 0, 0, 0], [1, 0, 0, 0, 0, 0]$$

[2] https://github.com/santiontanon/microrts



(a) 4×4 Map (b) 10×10 Map (c) 4×4 Map (d) 10×10 Map

Figure 2: The first two figures show the episodic return over the time steps, and the remaining two show the Kullback–Leibler (KL) divergence between the target and current policy of PPO over the time steps. The shaded area represents one standard deviation of the data over 4 random seeds. Curves are exponentially smoothed with a weight of 0.9 for readability.

The 27 values of each feature plane for the position in the map of such worker will thus be the concatenation of the arrays above.

- **Action Space.** Given a map of size $h \times w$, the action is an 8-dimensional vector of discrete values as specified in Table 1. The action space is designed similar to the action space formulation by Hausknecht, et al., (Hausknecht and Stone 2016). The first component of the action vector represents the unit in the map to issue actions to, the second is the action type, and the rest of the components represent the different parameters different action types can take. Depending on which action type is selected, the game engine will use the corresponding parameters to execute the action.
- **Rewards.** We are evaluating our agents on the simple task of harvesting resources as fast as they can for Player 1 who controls units at the top left of the map. A $+1$ reward is given when a worker harvests a resource, and another $+1$ is received once the worker returns the resource to a base.
- **Termination Condition.** We set the maximum game length to be 200 time steps, but the game could be terminated earlier if all the resources in the map are harvested first.

Notice that the space of invalid actions becomes significantly larger in larger maps. This is because the range of the first and last discrete values in the action space, corresponding to *Source Unit* and *Attack Target Unit* selection, grows linearly with the size of the map. To illustrate, in our experiments, there are usually only two units that can be selected as the *Source Unit* (the base and the worker). Although it is possible to produce more units or buildings to be selected, the production behavior has no contribution to reward and therefore is generally not learned by the agent. Note the range of *Source Unit* is $4 \times 4 = 16$ and $24 \times 24 = 576$, in maps of size 4×4 and 24×24, respectively. Selecting a valid *Source Unit* at random has a probability of $2/16 = 0.125$ in the 4×4 map and $2/576 = 0.0034$ in the 24×24 map. With such action space, we can examine the scalability of invalid action masking.

Training Algorithm

We use Proximal Policy Optimization (Schulman et al. 2017) as the DRL algorithm to train our agents.

Strategies to Handle Invalid Actions

To examine the empirical importance of invalid action masking, we compare the following four strategies to handle invalid actions.

1. **Invalid action penalty.** Every time the agent issues an invalid action, the game environment adds a non-positive reward $r_{\text{invalid}} \leq 0$ to the reward produced by the current time step. This technique is standard in previous work (Dietterich 2000). We experiment with $r_{\text{invalid}} \in \{0, -0.01, -0.1, -1\}$, respectively, to study the effect of the different scales on the negative reward.
2. **Invalid action masking.** At each time step t, the agent receives a mask on the *Source Unit* and *Attack Target Unit* features such that only valid units can be selected and targeted. Note that in our experiments, invalid actions still could be sampled because the agent could still select incorrect parameters for the current action type. We didn't provide a feature-complete invalid action mask for simplicity, as the mask on *Source Unit* and *Attack Target Unit* already significantly reduce the action space.
3. **Naive invalid action masking.** At each time step t, the agent receives the same mask on the *Source Unit* and *Attack Target Unit* as described for invalid action masking. The action shall still be sampled according to the re-normalized probability calculated in Equation 4, which ensures no invalid actions could be sampled, but the gradient is updated according to the probability calculated in Equation 1. We call this implementation *naive invalid action masking* because its gradient does not replace the gradient of the logits corresponds to invalid actions with zero.
4. **Masking removed.** At each time step t, the agent receives the same mask on the *Source Unit* and *Attack Target Unit* as described for invalid action masking, and trains in the same way as the agent trained under invalid action mask-

Table 2: Results averaged over 4 random seeds. The symbol "-" means "not applicable". Higher is better for r_{episode} and lower is better for a_{null}, a_{busy}, a_{owner}, t_{solve}, and t_{first}.

Strategies	Map size	r_{invalid}	r_{episode}	a_{null}	a_{busy}	a_{owner}	t_{solve}	t_{first}
Invalid action penalty	4×4	-1.00	0.00	0.00	0.00	0.00	-	0.53%
		-0.10	30.00	0.02	0.00	0.00	50.94%	0.52%
		-0.01	**40.00**	0.02	0.00	0.00	14.32%	0.51%
		0.00	30.25	2.17	0.22	2.70	36.00%	0.60%
	10×10	-1.00	0.00	0.00	0.00	0.00	-	3.43%
		-0.10	0.00	0.00	0.00	0.00	-	2.18%
		-0.01	0.50	0.00	0.00	0.00	-	1.57%
		0.00	0.25	90.10	0.00	102.95	-	3.41%
	16×16	-1.00	0.25	0.00	0.00	0.00	-	0.44%
		-0.10	0.75	0.00	0.00	0.00	-	0.44%
		-0.01	1.00	0.02	0.00	0.00	-	0.44%
		0.00	1.00	184.68	0.00	2.53	-	0.40%
	24×24	-1.00	0.00	49.72	0.00	0.02	-	1.40%
		-0.10	0.25	0.00	0.00	0.00	-	1.40%
		-0.01	0.50	0.00	0.00	0.00	-	1.92%
		0.00	0.50	197.68	0.00	0.90	-	1.83%
Invalid action masking	04x04	-	**40.00**	-	-	-	8.67%	**0.07%**
	10x10	-	**40.00**	-	-	-	11.13%	**0.05%**
	16x16	-	**40.00**	-	-	-	11.47%	**0.08%**
	24x24	-	**40.00**	-	-	-	18.38%	**0.07%**
Masking removed	04x04	-	33.53	63.57	0.00	17.57	76.42%	-
	10x10	-	25.93	128.76	0.00	7.75	94.15%	-
	16x16	-	17.32	165.12	0.00	0.52	-	-
	24x24	-	17.37	150.06	0.00	0.94	-	-
Naive invalid action masking	4×4	-	**59.61**	-	-	-	11.74%	**0.07%**
	10×10	-	**40.00**	-	-	-	13.97%	**0.05%**
	16×16	-	**40.00**	-	-	-	30.59%	**0.10%**
	24×24	-	**38.50**	-	-	-	49.14%	**0.07%**

ing. However, we then evaluate the agent without providing the mask. In other words, in this scenario, we evaluate what happens if we train with a mask, but then perform without it.

We evaluate the agent's performance in maps of sizes 4×4, 10×10, 16×16, and 24×24. All maps have one base and one worker for each player, and each worker is located near the resources.

Evaluation Metrics

We used the following metrics to measure the performance of the agents in our experiments: r_{episode} is the average episodic return over the last 10 episodes. a_{null} is the average number of actions that select a *Source Unit* that is not valid over the last 10 episodes. $a_{\text{busy}}$is the average number of actions that select a *Source Unit* that is already busy executing other actions over the last 10 episodes. a_{owner} is the average number of actions that select a *Source Unit* that does not belong to Player 1 over the last 10 episodes. t_{solve} is the percentage of total training time steps that it takes for the agents' moving average episodic return of the last 10 episodes to exceed 40. t_{first} is the percentage of the total training time step that it takes for the agent to receive the first positive reward.

Evaluation Results

We report the results in Figure 2 and in Table 2. Here we present a list of important observations:

Invalid action masking scales well. Invalid action masking is shown to scale well as the number of invalid actions increases; t_{solve} is roughly 12% and very similar across different map sizes. In addition, the t_{first} for invalid action masking is not only the lowest across all experiments (only taking about $0.05 - 0.08\%$ of the total time steps), but also consistent against different map sizes. This would mean the agent was able to find the first reward very quickly regardless of the map sizes.

Invalid action penalty does not scale. Invalid action penalty is able to achieve good results in 4×4 maps, but it does not scale to larger maps. As the space of invalid action gets larger, sometimes it struggles to even find the very first reward. E.g. in the 10×10 map, agents trained with invalid action penalty with $r_{\text{invalid}} = -0.01$ spent 3.43% of the entire training time just discovering the first reward, while agents trained with invalid action masking take roughly 0.06% of the time in all maps. In addition, the hyper-parameter r_{invalid} can be difficult to tune. Although having a negative r_{invalid} did encourage the agents not to execute any invalid actions (e.g. a_{null}, a_{busy}, a_{owner} are usually very close to zero for these agents), setting $r_{\text{invalid}} = -1$

seems to have an adverse effect of discouraging exploration by the agent, therefore achieving consistently the worst performance across maps.

KL divergence explodes for naive invalid action masking. According to Table 2, the r_{episode} of naive invalid action masking is the best across almost all maps. In the 4×4 map, the agent trained with naive invalid action masking even learns to travel to the other side of the map to harvest additional resources. However, naive invalid action masking has two main issues: 1) As shown in the second row of Figure 2, the average Kullback–Leibler (KL) divergence (Kullback and Leibler 1951) between the target and current policy of PPO for naive invalid action masking is significantly higher than that of any other experiments. Since the policy changes so drastically between policy updates, the performance of naive invalid action masking might suffer when dealing with more challenging tasks. 2) As shown in Table 2, the t_{solve} of naive invalid action masking is more volatile and sensitive to the map sizes. In the 24×24 map, for example, the agents trained with naive invalid action masking take 49.14% of the entire training time to converge. In comparison, agents trained with invalid action masking exhibit a consistent $t_{\text{solve}} \approx 12\%$ in all maps.

Masking removed still behaves to some extent. As shown in Figures 2b, masking removed is still able to perform well to a certain degree. As the map size gets larger, its performance degrades and starts to execute more invalid actions by, most prominently, selecting an invalid *Source Unit*. Nevertheless, its performance is significantly better than that of the agents trained with invalid action penalty even though they are evaluated without the use of invalid action masking. This shows that the agents trained with invalid action masking can, to some extent, still produce useful behavior when the invalid action masking can no longer be provided.

Conclusions

In this paper, we examined the technique of invalid action masking, which is a technique commonly implemented in policy gradient algorithms to avoid executing invalid actions. Our work shows that: 1) the gradient produced by invalid action masking is a valid policy gradient, 2) it works by applying a *state-dependent differentiable function* during the calculation of action probability distribution, 3) invalid action masking empirically scales well as the space of invalid action gets larger; in comparison, the common technique of giving a negative reward when an invalid action is issued fails to scale, sometimes struggling to find even the first reward in our environment, 4) the agent trained with invalid action masking was still able to produce useful behaviors with masking removed.

Given the clear effectiveness of invalid action masking demonstrated in this paper, we believe the community would benefit from wider adoption of this technique in practice. Invalid action masking empowers the agents to learn more efficiently, and we ultimately hope that it will accelerate research in applying DRL to games with large and complex discrete action spaces.

References

[Berner et al. 2019] Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; Józefowicz, R.; Gray, S.; Olsson, C.; Pachocki, J. W.; Petrov, M.; de Oliveira Pinto, H. P.; Raiman, J.; Salimans, T.; Schlatter, J.; Schneider, J.; Sidor, S.; Sutskever, I.; Tang, J.; Wolski, F.; and Zhang, S. 2019. Dota 2 with large scale deep reinforcement learning. *ArXiv preprint* abs/1912.06680.

[Dietterich 2000] Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research* 13:227–303.

[Hausknecht and Stone 2016] Hausknecht, M. J., and Stone, P. 2016. Deep reinforcement learning in parameterized action space. In Bengio, Y., and LeCun, Y., eds., *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

[Huang and Ontañón 2019] Huang, S., and Ontañón, S. 2019. Comparing observation and action representations for deep reinforcement learning in μrts.

[Kullback and Leibler 1951] Kullback, S., and Leibler, R. A. 1951. On information and sufficiency. *The annals of mathematical statistics* 22(1):79–86.

[Schulman et al. 2017] Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *ArXiv preprint* abs/1707.06347.

[Stanescu et al. 2016] Stanescu, M.; Barriga, N. A.; Hess, A.; and Buro, M. 2016. Evaluating real-time strategy game states using convolutional neural networks.

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

[Sutton et al. 2000] Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, 1057–1063.

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

[Vinyals et al. 2019] Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature* 575(7782):350–354.

[Yang and Ontañón 2018] Yang, Z., and Ontañón, S. 2018. Learning map-independent evaluation functions for real-time strategy games. *2018 IEEE Conference on Computational Intelligence and Games (CIG)* 1–7.

[Ye et al. 2020] Ye, D.; Liu, Z.; Sun, M.; Shi, B.; Zhao, P.; Wu, H.; Yu, H.; Yang, S.; Wu, X.; Guo, Q.; Chen, Q.; Yin, Y.; Zhang, H.; Shi, T.; Wang, L.; Fu, Q.; Yang, W.; and Huang, L. 2020. Mastering complex control in MOBA games with deep reinforcement learning. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, 6672–6679. AAAI Press.