Proceedings of the Fifteenth AAAI Conference on Artificial
Intelligence and Interactive Digital Entertainment (AIIDE-19)

On Hard Exploration for Reinforcement Learning: A Case Study in Pommerman

Chao Gao,[2*] Bilal Kartal,[1] Pablo Hernandez-Leal,[1] Matthew E. Taylor[1]
[1]Borealis AI
[2]University of Alberta
Edmonton Canada
cgao3@ualberta.ca, {bilal.kartal, pablo.hernandez, matthew.taylor}@borealisai.com

Abstract

How to best explore in domains with sparse, delayed, and deceptive rewards is an important open problem for reinforcement learning (RL). This paper considers one such domain, the recently-proposed multi-agent benchmark of Pommerman. This domain is very challenging for RL — past work has shown that model-free RL algorithms fail to achieve significant learning without artificially reducing the environment's complexity. In this paper, we illuminate reasons behind this failure by providing a thorough analysis on the hardness of random exploration in Pommerman. While model-free random exploration is typically futile, we develop a model-based automatic reasoning module that can be used for safer exploration by pruning actions that will surely lead the agent to death. We empirically demonstrate that this module can significantly improve learning.

Efficient exploration is a long-standing problem in reinforcement learning (Haarnoja et al. 2017; Nachum, Norouzi, and Schuurmans 2017). Thrun (1992) divided exploration methods in two: undirected, which are closely related to a random walk with the advantage of not using specific knowledge (Szepesvári 2010; Still and Precup 2012); and directed, which need domain specific knowledge with the advantage of generally better exploration. A subcategory of directed exploration are model-based approaches, which usually take into account how often a state-action pair has been visited (Brafman and Tennenholtz 2002; Strehl and Littman 2008). However, count approaches suffer in non-tabular cases with large state spaces. In these challenging cases, the agent is usually equipped with a function approximator for effective generalization across familiar and unfamiliar states. An example that extends tabular count-based exploration to non-linear function approximation is pseudo-count (Bellemare et al. 2016), which can be used to evaluate states that have not been visited in the past.

However, these methods usually struggle in hard-exploration problems (Ecoffet et al. 2019) (e.g., Atari games such as Montezuma's Revenge and Pitfall) where many long sequences of actions are needed before obtaining obtaining non-zero external signals.

This paper considers the exploration problems that arise in a recently proposed benchmark for (multi-agent) reinforcement learning: Pommerman (Resnick et al. 2018a). The environment is based on the classic console game *Bomberman* that involves 4 bomber agents initially placed at the four corners of a board. The game can be played in Free-For-All (FFA) or Team modes. In FFA the winner is the last agent that survives; in Team mode each two diagonal agents form a team, and one team wins if it successfully destroys all members in the other team. In either mode, at every step, each agent issues an action simultaneously from 6 discrete candidate actions: moving `left`, `right`, `up`, `down`, placing a `bomb`, or `stop`. The `bomb` action is legal as long as the agent's `ammo` is greater than 0, and any illegal action (such as moving towards a wall) is superseded with `stop` by the environment. Figure 1 shows a snapshot of the game.

Pommerman is challenging for multi-agent learning, in particular, model-free reinforcement learning, predominantly because exploration is difficult due to: (1) delayed action effects: the only way to make a change to the environment (e.g., kill an agent) is by means of bomb placement, but the effect of such an action is only effective when the bomb explodes after 10 time steps; and (2) sparse and deceptive rewards: the former refers to the fact that the only non-zero reward is obtained at the end of an episode. The latter refers to the fact that quite often a winning reward is due to the opponents' involuntary suicide, which makes reinforcing an agent's action based on such a reward *deceptive*.

While techniques like Reward Shaping and Difference Rewards (Agogino and Tumer 2008; Devlin et al. 2014) have been used to deal with (2), for (1), efficient and safe exploration is crucial because failing to avoid the side-effect of the bomb actions results in irreversible events (Farquhar et al. 2017; Racanière et al. 2017), i.e., the agent is eliminated. Although it has been suggested that model-free RL does not learn well in Pommerman, particularly because the exploration with bomb action is highly correlated to losing (Resnick et al. 2018a; 2018b; Kartal, Hernandez-Leal, and Taylor 2019), a formal analysis about the exploration difficulty is lacking. In this paper we fill this gap by presenting an analysis on the exploration hardness in Pommer-

*Work performed as a part-time intern at Borealis AI.

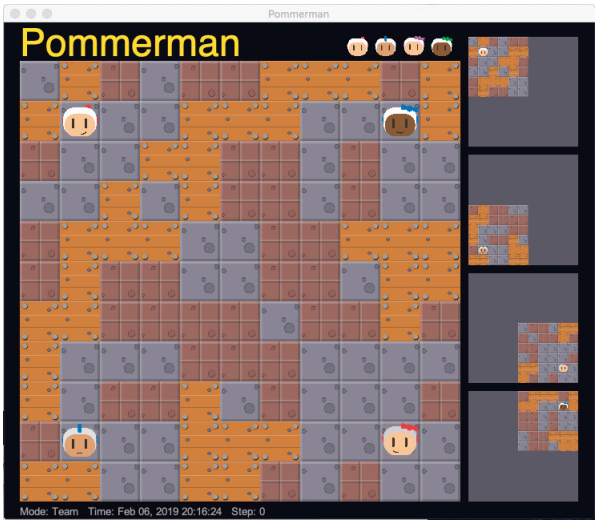


Figure 1: The game of Pommerman in Team mode, where each two-diagonal agents form a team. The right column shows the partial view of the board for each agent; the view is fully observable in FFA mode.

man: we show that suicides happen frequently during learning because of the nature of using model-blind random actions for exploration. Furthermore, the high rate of suicides has a direct effect on the samples needed to learn. We exemplify this in a case for which an *exponential* number of samples are needed to obtain a positive experience. This highlights that performing non-suicidal bomb placement could require complicated, long-term, and accurate planning. We then propose an efficient reasoning module that prunes unsafe actions. We experimentally demonstrate the usefulness of the module by comparing the learning performances with and without such a module, and further show the strength of obtained players by competing against `SimpleAgent`, the official baseline provided by Pommerman

Analysis of Random Exploration

Before presenting our analysis in Pommerman, we first describe a related classic problem of random walk in two-dimensional world, then we generalize to the case of random walk in a world with *obstacles*, and finally we describe the results for Pommerman.

Obstacle-free 2D grid world

We start with a simplified scenario where an agent sits on a 2d world and has five actions — `stop, left, right, up, down` — to navigate around. Assume the agent's original location is marked as $(0, 0)$ and follows an uniform random policy. The question of interest is: at time step $t \geq 0$, what is the probability that the agent will be at cell (i, j) for arbitrary (i, j)?

Each step the agent has five actions to take, thus after t steps, there are in total of 5^t "paths," because the probability of each path is the same, the probability that the agent be at position $p = (i, j)$ can be computed as $\frac{N(p,t)}{5^t}$, where

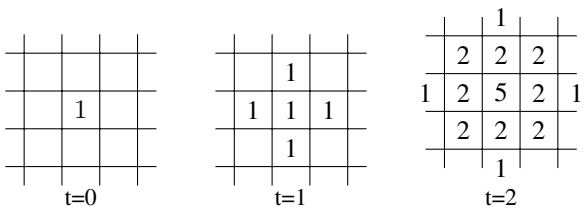


Figure 2: At step 0, the agent is located at the origin; at step 1, the agent could be at any position 1-step away; at step 2, the agent can at most reach a position that is two-steps away. The probability of being at each position can be computed by dividing the number of all feasible combinatorial trajectories, i.e., 5^t.

$N(p, t)$ is the number of paths that lead the agent to position p at time t. Therefore, the question is how to calculate $N(p, t)$ for any position $p = (i, j)$ and time step t.

After any tick the agent can at most move to a cell that is one step away. Therefore, to arrive at p at time step t, the agent must get to the neighboring positions within $t - 1$, as shown by the recursive relation:

$$N(p,t) = \begin{cases} \sum_{q \in \text{neighbor}(p)} & N(q, t-1) + N(p, t-1), \\ 1, & p = (0,0) \text{ and } t = 0 \end{cases} \tag{1}$$

where $\textit{neighbor}(p) = \{(i-1, j), (i, j-1), (i+1, j), (i, j+1)\}$, given $p = (i, j)$.

This recursion expresses that to be on position (i, j) at time t, in the previous time step $t - 1$, the agent must be at $(i-1, j), (i, j-1), (i+1, j), (i, j+1)$ or (i, j) from where one action can be taken by the agent, resulting in (i, j).

Since $N(p, t)$ in Equation 1 depends solely on the information in time $t - 1$, and solution to the base case ($t = 0$) is given, this indicates that $\forall p, t \geq 0, N(p, t)$ can be computed bottom-up using dynamic programming by iteratively increasing the value of t. As an illustration, Figure 2 shows the number of paths to each position for time steps $t = 0, 1, 2$, where unmarked positions are out of reach of the agent — they can also be filled with 0. Note also that each number in a sub-figure can be computed summing all the numbers of its neighbors and itself from the sub-figure on the left.

Generalization to arbitrary obstacles

Now we generalize to a scenario where: (i) the board is bounded and trying to move outside results in *stop*; and (ii) there are arbitrary static obstacles that the agent cannot pass through, i.e., the agent stays in the same position. This scenario is similar to a typical situation in Pommerman, where an agent is positioned on a board with limited size, has `ammo` of 0 (no bomb placement allowed) and can use the remaining five legal actions to explore the world.[1]

[1]Note that stripping the invalid `bomb` action when `ammo=0` does not violate the model-free principle, since from the egocentric perspective of view, `ammo` is an attribute of the agent rather than a specification from model of the environment.

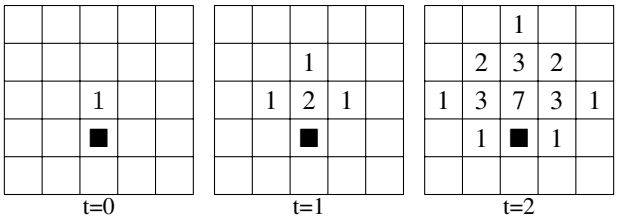


Figure 3: At step 0, the agent is located at the origin and there is one obstacle in the board represented by ■; step 1, the agent could be at any position 1-step away; at step 2, the agent can at most reach to a position that is two-steps away. The probability of being at each position can be computed by dividing the corresponding normalization constant 5^t.

Only a minor revision to Equation 1 is needed to represent the number of paths of being at each $p = (i, j)$ after t steps, given a board configuration $\mathcal{B}$, as:

$$N(p, t, \mathcal{B}) = \sum_{a \in A} N(prev(a, p, \mathcal{B}), t - 1, \mathcal{B}), \quad t > 0 \quad (2)$$

$$N(p, t, \mathcal{B}) = 1, p = (0, 0) \wedge t = 0 \quad (3)$$

where $A = \{\texttt{stop}, \texttt{left}, \texttt{right}, \texttt{up}, \texttt{down}\}$ is the action set and $prev(a, p, \mathcal{B})$ returns the previous position the agent must be if action a leads the agent to position $p = (i, j)$ on a board configuration $\mathcal{B}$.

Figure 3 shows an example for $t = 0, 1, 2$. In this case, there is one obstacle in the board (represented by ■).

Suicide study: Pommerman

In Pommerman, it has been noted that the action of bomb placing is highly correlated to losing (Resnick et al. 2018a), which is presumably the major impediment for achieving good results using model-free reinforcement learning. Now, we provide a formal analysis of the suicide problem that was suggested to be the reason to delay or prevent the agent from learning the *bombing skill* (Kartal, Hernandez-Leal, and Taylor 2019) when an agent follows a model-blind, random exploration policy.

In Pommerman, an agent can only be killed when it intersects with an exploding bomb's flames, then we say a *suicide* event happens if the death of the agent is caused by its own bomb. For the ease of exposition we consider the following simplified scenario: (i) the agent has `ammo=1` and has just placed a bomb (i.e., the agent now sits on the bomb with `ammo=0`); (ii) for the next time steps until the bomb explodes, the agent has 5 actions available at every time step; and (iii) other items on the board are static.

One difference between this case and the previous section is that the agent sits on an "obstacle" (i.e., a bomb), and once it moves off it cannot come back. This is in contrast to the previous assumption where the number of paths to any position is always related to its adjacent cells. Formally, the original position p of the agent represents a singleton point that if the agent is at one neighbor of p in time t, it cannot go back to p in any $t' \geq t + 1$. Because of this special point, we have the following observations:

Observation 1. *Assuming p is the origin and the bomb has not exploded yet, the number of paths of being at p after t time steps on a board $\mathcal{B}$ is given by:*

$$N(p, t, \mathcal{B}) = (|obstacle(p, \mathcal{B})| + 1)^t \quad with \quad \forall t \geq 0$$

where $obstacle(p, \mathcal{B})$ is the set of obstacle cells adjacent to p.

Proof. To stay at the origin, at each step, the agent must take an action either is `stop` or results in a bounce back, so that after t steps, there is in total $(|obstacle(p, \mathcal{B})| + 1)^t$ possible "paths." Note that $|obstacle(p, \mathcal{B})| \leq 4$. □

Due to the special treatment of the origin, the asymmetric impact needs to be propagated to its adjacent neighbours by the following initialization (noted as $\hat{N}$):

Observation 2. *For any passable neighbor $q \in neighbor(p)$, after t time steps in $\mathcal{B}$:*

$$\hat{N}(q, t, \mathcal{B}) = \hat{N}(p, t - 1, \mathcal{B})$$

where p is the origin and $\hat{N}(p, t - 1, \mathcal{B}) = (|obstacle(p, \mathcal{B})| + 1)^{t-1}$.

The other cells can be initialized $\hat{N}$ with 0, because they are out of direct influence of the origin.

Now, the remaining problem can be solved exactly by applying the scheme in the previous section, except that the initialization value has to be added to the recursion for $t \geq 0$ as follows:

$$N(p, t, \mathcal{B}) = \hat{N}(p, t, \mathcal{B}) + \sum_{a \in A} N(prev(a, p, \mathcal{B}), t - 1, \mathcal{B}).$$

Now, we use the previous analysis to exemplify a random, best, and worst cases that happen in Pommerman. For brevity, we call the number of paths to a cell c as the *N value of c*, or *N value of c* at time t if t has to be explicitly noted.

Example 1. *We take a random starting board configuration, depicted in Figure 1, and compute the N values ending at each position for $t = 0$ and $t = 3$ in Figure 4. This is a typical starting board in Pommerman, where every agent stays at its corner; they are disconnected with each other by randomly generated obstacles. One noticeable phenomenon is that the N values are concentrated around its origin.*

We also compute for $t = 9$ (time steps needed for a bomb to explode) before and after normalizing the number of paths into probabilities in Figure 5. Note that even when the configuration of the agents is slightly different their probabilities of ending up with suicide are $\approx 40\%$ $(0.39, 0.38, 0.46, 0.38$, counterclockwise starting from upper-left corner).

Indeed, the problem of suicide stems from acting randomly without considering constraints of the environment. In extreme cases, in each time step, the agent may have only one survival action, which means the safe path is always unique as t increases while the total number of feasible paths the agent can traverse grows exponentially. We illustrate this by the following corridor example.

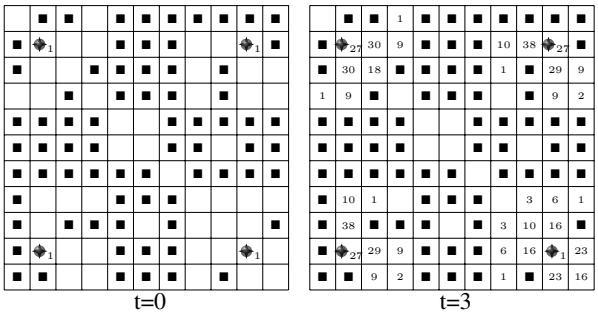


Figure 4: In Pommerman, after placing a bomb, each agent sits on the bomb; left and right figures are the number of paths of being at each cell at step 0 and 3, respectively. Probabilities can be obtained by dividing by 5^t.

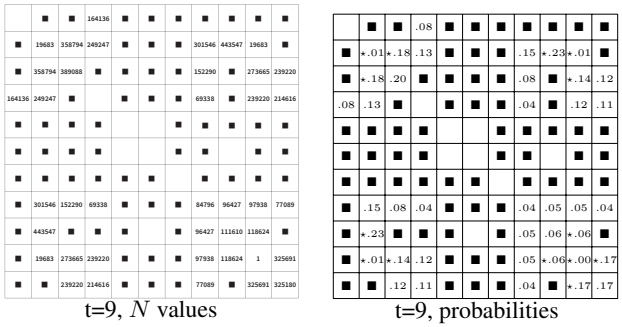


Figure 5: After 9 steps, the N value and probability of being at each cell for each agent. $\star$ indicates the cell is covered by flames.

Example 2. *Figure 6 shows a worst-case example: the agent is in a corridor formed by wood at two sides and places a bomb. If using model-blind exploration the chance of suicide is extremely high since among the* 5^9 *"paths,"* only one *of them is safe. In order to survive, it must precisely follow the right action at each time step. This also implies that for sub-problems of such in Pommerman, to acquire one positive behaviour example requires* exponential *number of samples.*

The previous example shows a worst-case scenario, now we present also the best-case.

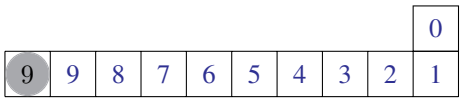


Figure 6: The corridor scenario: the agent places a bomb with `strength=10` on the leftmost cell. For each passage cell, the marked value means the minimum number of steps it is required to safely evade from impact of the bomb. After placing the bomb, in the next step the bomb has life of 9, thus in the remaining 9 ticks, the agent must exactly take the right action to evade.

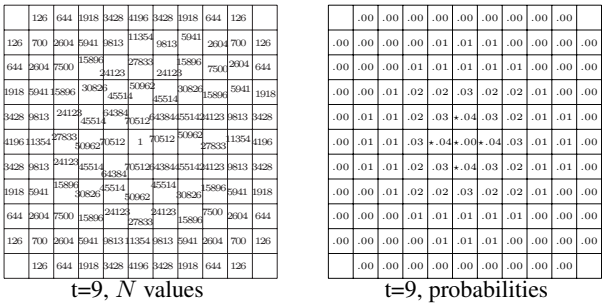


Figure 7: Best case for an agent to be after placing a bomb after 9 steps: empty board. The N value and probability of being at each cell for each agent; $\star$ indicates the cell is covered by flames.

Observation 3. *Suppose the agent starts by sitting on a bomb and the agent uniform-randomly walks by taking actions* `left`*,* `right`*,* `up`*,* `down`*, or* `stop`*. The best board configuration for the agent not committing suicide is when there are no obstacles (except the bomb at the origin) and the agent is at the center of the board.*

Figure 7 shows the computation results in detail for this best-case scenario where the suicide probability is 0.16. However, in Pommerman, such an ideal case does not exist at all — we have seen from Figures 3, 4, 5 and 6, usually the probability is higher. Specifically, for a whole game, if k bombs have to be placed, suppose the agent's suicide probability is x each time, then the suicide rate is thus $1-(1-x)^k$ which increases as k increases — this puts the agent in complicated scenario: placing more bombs is required for playing well while at the same time it also results more probable suicide. For example, suppose an agent has to survive $k = 8$ bombs to carve out a path to engage with enemies (not an uncommon scenario in Pommerman), then assuming $x = 0.4$, the probability of death after 8 bombs is ≈ 1. Furthermore, even the agent somehow managed to survive from these bombs, no reward is observed as the positive reward signal appears only when all opponents are dead.

Action Pruning

In this section we show that if the model-free demand is relaxed, an action pruning algorithm for safe exploration exists in Pommerman.

In the case illustrated in Figure 6, we see that the essence for survival is to try to navigate to a position that is free from the threat of future bombs (rightmost upper cell in Figure 6). Let us call the *safety value* of a passable cell the minimum number of steps to reach a safe region (that is out of the reach of any bombs). Assuming static bombs, we can prune unsafe actions by comparing the post-action location's *safety value* and its minimum bomb life value. Figure 6 shows the safety values for the corridor example.

This idea can be generalized to arbitrary bombs: given a board situation and the agent's position, each resulting cell after taking an action can be computed — the remaining problem is to compute its *safety value* as well as the min-

imum life value among the bombs covering this position, where we say a position is *covered by a bomb* if (after the explosion) the position is within reach of the flames.

We describe the complete algorithm for computing *safety values* in Algorithm 1 (`MinEvadeStep` function). The input to the algorithm are attributes of the agent as well as the board information that the agent can see. The output (calling `GetSafeActions`) is set of *safe* actions. Exact computation is impossible for partially observable environments, but can be approximated by treating unseen areas as walls.

Algorithm 1: Action Pruning

Input: $board, agent, A$
Output: $\hat{A}$
Function `MinEvadeStep`$(board, p, history)$**:**
  $u, l \leftarrow$ `FindMaxMinBombCovering`$(board, p)$
  `/* no bomb covering p */`
  if $u = -\infty$ **then**
    return 0
  else if $|history| \geq u$ **then**
    `/* bomb would have exploded upon arrival; flame_life=2 */`
    if $|history| > u + flame\_life$ **then return** 0
    else return ∞
  else if $|history| \geq l$ **then**
    if $|history| > l + flame\_life$ **then return** 0
    else return ∞
  end
  $num \leftarrow \infty$
  for $a \in \{left, right, up, down\}$ **do**
    $q \leftarrow$ `NextPosition`(p, a)
    if $q \notin history$ **then**
      $num \leftarrow \min(num, 1 +$ `MinEvadeStep`$(board, q, history \cup \{q\}))$
    end
  end
  return num
Function `GetSafeActions`$(board, agent, A)$**:**
  $\hat{A} \leftarrow \emptyset$
  Let p be agent's position
  for $a \in A = \{left, right, up, down, stop, bomb\}$ **do**
    $board, q \leftarrow$ `Next`$(board, p, a)$
    if $a = stop$ *or* $a = bomb$ **then** $H \leftarrow \{None, q\}$
    else $H \leftarrow \{q\}$
    $n \leftarrow$ `MinEvadeStep`$(board, q, H)$
    $m \leftarrow$ `FindMinBombCovering`$(board, p)$
    if $m > n$ **then** $\hat{A} \leftarrow \hat{A} \cup \{a\}$
  end
  return $\hat{A}$

The essence of Algorithm 1 is a recursive reasoning procedure that leverages the relationship between bomb's coverage and *safety value*: (1) a cell is safe if it is not covered by any bombs, and (2) a bomb-covering cell is also safe if the agent has enough time and an viable way to evade to a bomb-free region. In other words, if the board is fully observable and bombs and opponents are not moving, optimal moves will never be pruned. The immediate use of such pruning is embedding it in model-free reinforcement learning system where the goal is to learn a policy among the remaining "safe" actions, which we shall focus on in the remaining of the paper. Algorithm 1 can also be used in dynamic environments by looking-ahead tree-search (i.e., by expanding all opponent moves), though this will not be explored in this paper.

Algorithm 1 is computationally efficient, i.e., all utility functions are computable in $O(L)$, where L is width of the board, while the recursion depth of `MinEvadeStep` is less than the maximum bomb life (i.e., 10), thus the total complexity is bounded by $O(L^2)$ because of repeated subproblems; $L = 11$ in Pommerman.

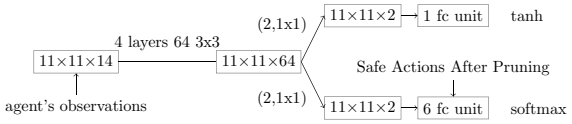


Figure 8: Architecture for learning.

Experiments

Here we show learning results both in modes, FFA and Team, comparing with/without action pruning.

Learning Algorithm

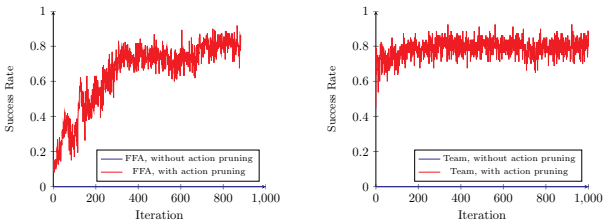
We use the PPO algorithm (Schulman et al. 2017) as the backend algorithm for our reinforcement learners, which minimizes the following error function on its $\mathcal{D}$, i.e., games played by the old neural net represented policy π_θ^{old}:

$$L(\theta; \mathcal{D}) = \sum_{(s_t, a_t, R_t) \in \mathcal{D}} \Bigg[- clip(\frac{\pi_\theta(a_t|s_t)}{\pi_\theta^{old}(a_t|s_t)}, 1-\epsilon, 1+\epsilon) A(s_t, a_t) + \frac{1}{4} \cdot \max \Big[(v_\theta(s_t) - R_t)^2, (v_\theta^{old}(s_t) + clip(v_\theta(s_t) - v_\theta^{old}(s_t), -\epsilon, \epsilon) - R_t)^2 \Big] \Bigg]. \quad (4)$$

Here, R_t is the return at time t , which can be of -1, 0 or 1 for lose, draw or win; clip range $\epsilon = 0.1$, $A(s_t, a_t)$ is the generalized advantage function (Schulman et al. 2015), where $\lambda = 0.95$, discount $\gamma = 0.9$, and $n\_step = 32$. Those parameters were set in accordance with OpenAI baseline implementation.

Architecture Similar to previous research (Resnick et al. 2018a), we extract 14 features planes from the agent's observation which is the input to our network. As shown in Figure 8, the architecture contains 4 convolution layers, followed by two policy and value heads, respectively. The input contains 14 features planes, each of shape 11×11. It then convolves using 4 layers of convolution, each has 64 3×3 kernels; the result thus has shape $11 \times 11 \times 64$. Then, each head convolves using 2 1×1 kernels. Finally, the output is squashed into action probability distribution and value estimation, respectively.

Reward Shaping Following previous work on Pommerman (Resnick et al. 2018a), to cope with the sparse reward problem, a dense reward function is added during



(a) FFA mode (b) Team mode

Figure 9: Success rate against static opponents with/without action pruning module. Consistent with our prior analysis, PPO failed to learn without pruning. x-axis is the iteration number. Each iteration contains 80 games played by 8 parallel workers. When learning without pruning, the algorithm ran for 5000 iterations, but never succeeded.

learning: (1) going to a cell not in a FIFO queue of size 30 gets 0.001 (to encourage the agent to move around and not camp); (2) picking up power-ups `kick`, `ammo`, `increase blast strength` gets 0.01; (3) killing a teammate gets `-0.5`; (4) killing an opponent gets 0.5; and (5) blasting a piece of wood gets 0.01.

Learn with Static Opponents

To show the effectiveness of our action pruning algorithm, we first learn against a team of two *static* opponents, i.e., agents who always take the `stop` action. This reduces the problem to a task where the learning agent's role is just to remove wood, get close to, and kill opponents.

Figure 9 shows the results in both FFA and Team modes. In FFA mode, at each game, one corner agent is selected as the learning agent, and it wins if and only if it can kill the three static agents in 800 steps. Similarly, in Team mode, two corner agents are randomly selected as neural net team players, and they win if in 800 steps both of the two opponents are destroyed. Note that, even though static opponents may appear simple, the learning agents need to overcome obstacles (i.e, blasting wood) by bomb placement in a randomly generated board.

From Figure 9, it is clear that without any pruning, the neural nets failed to learn at all even after a training period of 5 days — the success rate remains zero, consistent with previous findings (Resnick et al. 2018a). By contrast, after equipping the agent with the action pruning module, PPO successfully learns the tasks.

Evaluate Against Baseline Opponents

SimpleAgent is a search-based baseline provided by the competition which uses hand-crafted rules for enemy interaction, bomb placement, and search for navigation.[2] To show the relative strength of our trained player, we tested the neural net model of the final iteration against this baseline opponent. The win-percentage of our player is around 40% against 3 `SimpleAgents` in FFA (note that a player with similar strength of `SimpleAgent` shall win in about 25%), and around 70% against a `SimpleAgent` team in Team mode. These results indicate that our player outperforms `SimpleAgent` baseline, even though the neural nets were only trained against static opponents. Indeed, an early version of such a module has been used in a competition agent; by further training against a curriculum of opponents, the final player won the second-place in the learning category at NeurIPS 2018 team competition (Gao et al. 2019). The best learning agent `Navocado` employed Dijkstra-based shortest path search, used an ensemble of several neural net models, in execution (Peng et al. 2018). The strongest agent is purely based on heuristic search (Osogami and Takahashi 2019).

[2]https://github.com/multiagentlearning/playground

Table 1: Results against `SimpleAgent` in FFA and Team modes. In FFA, one neural net plays against three SimpleAgent; in Team, one neural net team plays against one SimpleAgent team. Final iteration neural net models on FFA and Team training are selected respectively for the test. Three independent trials were tried, where each trial contains 100 games.

Trial	`FFA`			`Team`		
	win	draw	lose	win	draw	lose
1	43	12	45	69	6	25
2	40	12	48	65	5	30
3	42	17	41	76	6	18

Conclusions

We have provided an analysis on the hardness of directionless exploration in Pommerman. Our analysis sheds light on the previous negative results of using model-free RL for this domain. Our analysis may also apply to other grid-world like environments for game difficulty analysis. We further presented a model-based reasoning module and empirically showed its merit in both single-agent FFA and multi-agent Team environments. With the reasoning module our neural nets can learn to solve the pure exploration task given the opponents are *static*, while without the reasoning module the learning agent did not improve its performance after days of training. We hope our analysis will bring more attention on using Pommemran as a challenging domain in safe reinforcement learning (Garcıa and Fernández 2015).

A number of works on analyzing the limitations of deep learning methods have emphasized on the need use *models* (Pearl 2018; Darwiche 2018), often drawn connection to Systems I and II theories on human mind (Evans and Stanovich 2013). Recent success (Silver et al. 2016; 2017; 2018) in complex game playing relies on effective integration of *model-based* solver and *model-free* learner (Geffner 2018). Given the exploration difficulty of Pommerman, the promising direction for high-level playing is to incorporate action-pruning (for dealing with the survival problem) and tree-search (for look-ahead opponent moves) into the multi-agent learning system.

References

Agogino, A. K., and Tumer, K. 2008. Analyzing and visualizing multiagent rewards in dynamic and stochastic domains. *Autonomous Agents and Multi-Agent Systems* 17(2):320–338.

Bellemare, M.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016. Unifying count-based exploration and intrinsic motivation. In *NIPS*, 1471–1479.

Brafman, R. I., and Tennenholtz, M. 2002. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3(Oct):213–231.

Darwiche, A. 2018. Human-level intelligence or animal-like abilities? *Commun. ACM* 61(10):56–67.

Devlin, S.; Yliniemi, L.; Kudenko, D.; and Tumer, K. 2014. Potential-based difference rewards for multiagent reinforcement learning. In *AAMAS*, 165–172.

Ecoffet, A.; Huizinga, J.; Lehman, J.; Stanley, K. O.; and Clune, J. 2019. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.

Evans, J. S. B., and Stanovich, K. E. 2013. Dual-process theories of higher cognition: Advancing the debate. *Perspectives on psychological science* 8(3):223–241.

Farquhar, G.; Rocktäschel, T.; Igl, M.; and Whiteson, S. 2017. TreeQN and ATreeC: Differentiable Tree-Structured Models for Deep Reinforcement Learning. *arXiv preprint arXiv:1710.11417*.

Gao, C.; Hernandez-Leal, P.; Kartal, B.; and Taylor, M. E. 2019. Skynet: A Top Deep RL Agent in the Inaugural Pommerman Team Competition. In *4th Multidisciplinary Conference on Reinforcement Learning and Decision Making*.

Garcıa, J., and Fernández, F. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16(1):1437–1480.

Geffner, H. 2018. Model-free, model-based, and general intelligence. In *International Joint Conference on Artificial Intelligence (IJCAI-18)*, 10–17.

Haarnoja, T.; Tang, H.; Abbeel, P.; and Levine, S. 2017. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning*, 1352–1361.

Kartal, B.; Hernandez-Leal, P.; and Taylor, M. E. 2019. Using Monte Carlo Tree Search as a Demonstrator within Asynchronous Deep RL. *AAAI-19 Workshop on Reinforcement Learning in Games*.

Nachum, O.; Norouzi, M.; and Schuurmans, D. 2017. Improving policy gradient by exploring under-appreciated rewards. *International Conference on Learning Representations*.

Osogami, T., and Takahashi, T. 2019. Real-time tree search with pessimistic scenarios. *arXiv preprint arXiv:1902.10870*.

Pearl, J. 2018. Theoretical impediments to machine learning with seven sparks from the causal revolution. *arXiv preprint arXiv:1801.04016*.

Peng, P.; Pang, L.; Yuan, Y.; and Gao, C. 2018. Continual match based training in Pommerman: Technical report. *arXiv preprint arXiv:1812.07297*.

Racanière, S.; Weber, T.; Reichert, D.; Buesing, L.; Guez, A.; Rezende, D. J.; Badia, A. P.; Vinyals, O.; Heess, N.; Li, Y.; et al. 2017. Imagination-augmented agents for deep reinforcement learning. In *Advances in neural information processing systems*, 5690–5701.

Resnick, C.; Eldridge, W.; Ha, D.; Britz, D.; Foerster, J.; Togelius, J.; Cho, K.; and Bruna, J. 2018a. Pommerman: A multi-agent playground. *AIIDE Multi-Agent Workshop*.

Resnick, C.; Raileanu, R.; Kapoor, S.; Peysakhovich, A.; Cho, K.; and Bruna, J. 2018b. Backplay:" man muss immer umkehren". *arXiv preprint arXiv:1807.06919*.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; and Abbeel, P. 2015. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587):484.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *Nature* 550(7676):354.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science* 362(6419):1140–1144.

Still, S., and Precup, D. 2012. An information-theoretic approach to curiosity-driven reinforcement learning. *Theory in Biosciences* 131(3):139–148.

Strehl, A. L., and Littman, M. L. 2008. An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences* 74(8):1309–1331.

Szepesvári, C. 2010. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning* 4(1):1–103.

Thrun, S. B. 1992. Efficient exploration in reinforcement learning. Technical report, Technical Report CMU-CS-92-102, School of Computer Science, Carnegie Mellon.