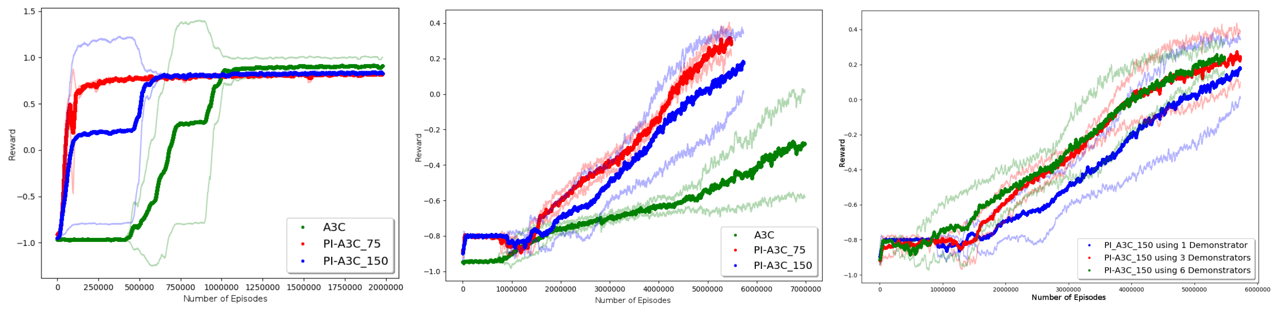

(a) Learning against a Static Agent (b) Learning against a Rule-based Agent (c) Varying the number of Demonstrators

Figure 3: a) Against Static agent, all variants have been trained for 12 hours. The PI-A3C framework using MCTS demonstrator with 75 and 150 rollouts learns significantly faster compared to the standard A3C. b) Against Rule-based opponent, all variants have been trained for 3 days. Against this more skilled opponent, PI-A3C provides significant speed up in learning performance, and finds better best response policies. c) Against Rule-based opponent, employing different number $(n = 1, 3, 6)$ of Demonstrators, i.e. increasing from 1 to 3, slightly improved the results, however, there is almost no variation from 3 to 6 demonstrators. For both (a) and (b), increasing the expertise level of MCTS through doubling the number of rollouts (from 75 to 150) does not yield improvement, and can even hurt performance. Our hypothesis is that slower planning decreases the number of demonstrator actions too much for the model-free RL workers to learn to imitate for safe exploration.

Table 1: Vanilla MCTS-based demonstrator evaluation: average episodic rewards of 200 games. Note that mean rewards are affected by too many tie games (rewarded -1 by the Pommerman simulator), e.g. against Static opponent, MCTS wins 88 games, loses 4 games by suicide, and ties 108 games, which results in average reward of -0.12.

Vanilla-MCTS vs.	Static	Rule-based
75 Rollouts	-0.12	-0.23
150 Rollouts	-0.08	-0.20

asynchronous updates to the global neural network. As Figure 3 shows, the relatively weaker demonstrator (75 rollouts) enabled faster learning than the one with 150 rollouts against both opponents. We also measured the Vanilla-MCTS performance against both opponents as reported in Table 1. The results show that average rewards increase when given more rollouts against both opponents. Against static opponents, Vanilla-MCTS performance is worse than both A3C and PI-A3C methods. This occurs because even when vanilla-MCTS does not commit suicides often, it fails to blast the opponent, resulting in too many tie games (episodic reward of -1). In contrast, PI-A3C makes use of the demonstrator for safer exploration with higher-quality atomic actions and learns to explore faster than A3C. Against a more challenging Rule-based opponent, pure MCTS (mean episodic reward of -0.23 using 75 rollouts) is slightly better than pure A3C (mean episodic reward of -0.27), however PI-A3C with either 75 or 150 rollouts combines both approaches well and perform the best, see Figure 3(b).

We hypothesize that the faster demonstrator (MCTS with only 75 rollouts) makes more updates to the global neural network, warming up other purely model-free workers for *safer* exploration earlier in contrast to the slower demonstrator. This is reasonable as the model-free RL workers constitute all but one of the CPU workers in these experiments, therefore the earlier the model-free workers can start safer exploration, the better the learning progress is likely to be.[3]

On the trade-off of using multiple demonstrators Our proposed method is built on top of an asynchronous distributed framework that uses several CPU workers: $k \geq 1$ act as a demonstrator and the rest of them explore in model-free fashion. We conducted one experiment to better understand how increasing the number of demonstrators, each of which provides additional *Planner Imitation* losses asynchronously, affects the learning performance. The trade-off is that more demonstrators imply fewer model-free workers to optimize the main task, but also a higher number of actions to imitate. We present the results in Figure 3(c) where we experimented 3 and 6 demonstrators, with identical resources and with 150 rollouts each. Results show that increasing to 3 improves the performance while extending to 6 demonstrators does not provide any marginal improvement. We also observe that the 3 demonstrator version using 150 rollouts presented in Figure 3(c) has a relatively similar performance with the 1 demonstrator version using 75 rollouts (see Figure 3(b)), which is aligned with our hypothesis that providing more demonstrator guidance early during learning is more valuable than fewer higher quality demonstrations.

Demonstrator biasing with policy network Uniform random rollouts employed by MCTS yield unbiased estimates, however it requires a high number of rollouts to reduce the variance. One way to improve search efficiency has

[3]Even though we used MCTS with fixed number of rollouts, this could be set dynamically, for example, by exploiting the reward sparsity or variance specific to the problem domain, e.g., using higher number of rollouts when close to bombs or opponents.

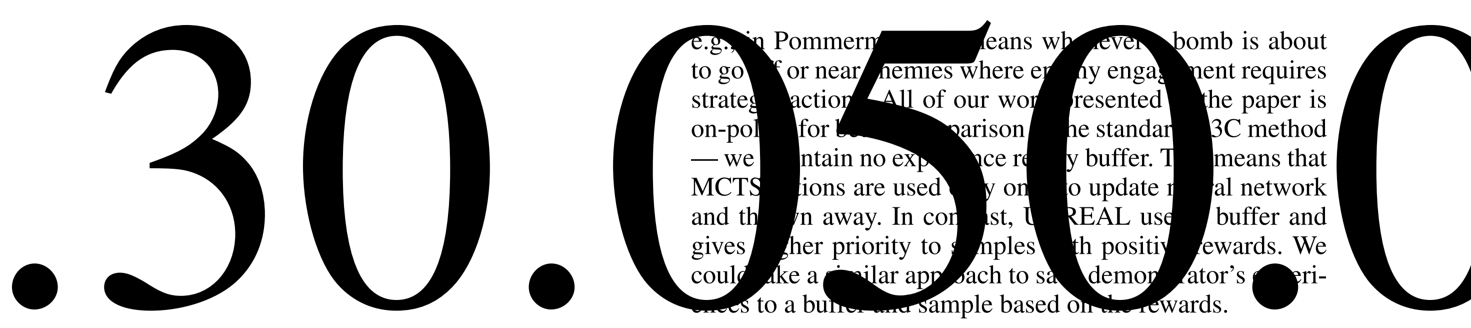

Figure 4: Learning against Rule-based opponent: using the policy head during MCTS rollout phase within the demonstrator provides improvement in learning speed, but it has higher variance compared to the default random policy.

been through different biasing strategies, such as prioritizing actions globally based on their evaluation scores (**?**), using heuristically computed move urgency values (**?**), or concurrently learning a rollout policy (**?**). In a similar vein with these methods, we let the MCTS-based demonstrator to access the policy head during rollouts, named as PI-A3C-NN (Planner Imitation - A3C with Neural Network). Our results suggest that employing a biased rollout policy improves in the average learning performance, however it has higher variance, as depicted in Figure 4.

Conclusions

In this work, we present a framework that uses a non-expert simulated demonstrator within a distributed asynchronous deep RL method to succeed in hard-exploration domains. Our experiments use the recently proposed Pommerman domain, a very challenging benchmark for pure model-free RL methods. In Pommerman, one main challenge is the high probability of suicide while exploring (yielding a negative reward), which occurs due to the delayed bomb explosion. However, the agent cannot succeed without learning how to stay safe after bomb placement. The methods and ideas proposed in this paper address this hard-exploration challenge. The main idea of our proposed framework is to use MCTS as a shallow demonstrator (small number of rollouts). In our framework, model-free workers learn to safely explore and acquire this skill by imitating a shallow search-based non-expert demonstrator. We performed different experiments varying the quality and the number of demonstrators. The results show that our proposed method shows significant improvement in learning efficiency across different opponents. There are several directions to extend our work. Soemers et al. (**?**) employed Breadth-First Search to initialize state-action value estimates by identifying near terminal states before starting MCTS based lookahead search. This enhancement can be tested within our framework. Also, MCTS-minimax hybrids method (**?**) can be employed as the demonstrator, which can provide better tactical play through minimax search. Another avenue is to investigate how to formulate the problem so that *ad-hoc* invocation of an MCTS-based simulator can be employed, i.e., action guidance can be employed only when the model-free RL agent needs one, e.g., in Pommerman this means whenever a bomb is about to go off or near enemies where enemy engagement requires strategic actions. All of our work presented in the paper is on-policy for better comparison to the standard A3C method — we maintain no experience replay buffer. This means that MCTS actions are used only once to update neural network and thrown away. In contrast, UNREAL uses a buffer and gives higher priority to samples with positive rewards. We could take a similar approach to save demonstrator's experiences to a buffer and sample based on the rewards.

Appendix

Neural Network Architecture: We use a NN with 4 convolutional layers that have 32 filters and 3×3 kernels, with stride and padding of 1, followed with a dense layer with 128 units, followed actor-critic heads. Architecture is not tuned. **State Representation:** Similar to (**?**), we maintain 28 feature maps that are constructed from the agent observation. These channels maintain location of walls, wood, power-ups, agents, bombs, and flames. Agents have different properties such as bomb kick, bomb blast radius, and number of bombs. We maintain 3 feature maps for these abilities per agent, in total 12 is used to support up to 4 agents. We also maintain a feature map for the remaining lifetime of flames. All the feature channels can be readily extracted from agent observation except the opponents' properties and the flames' remaining lifetime, which are tracked efficiently from sequential observations. **Hyperparameter Tuning:** We did not perform a through hyperparameter tuning due to long training times. We used a $\gamma = 0.999$ for discount factor. For A3C loss, the default weights are employed, i.e., $\lambda_v = 0.5$, $\lambda_\pi = 1.0$, and $\lambda_H = 0.01$. For the *Planner Imitation* task, $\lambda_{PI} = 1$ is used for the MCTS worker. We employed the Adam optimizer with a learning rate of 0.0001. We found that for the Adam optimizer, $\epsilon = 1 \times 10^{-5}$ provides a more stable learning curve (less catastrophic forgetting). We used a weight decay of 1×10^{-5} within the Adam optimizer for L2 regularization. **MCTS Implementation Details:** After rollouts, demonstrator action is selected by the max visit count, a.k.a. *Robust Child* (**?**). Default UCB1 exploration constant of $\sqrt{2}$ is used. The max search tree depth from any given state is set to 25 given the low search budget. The value head by the NN is not used by vanilla MCTS,

Velit dolores quod fugit impedit modi iusto expedita, obcaecati pariatur maiores illo dolores sit omnis praesentium officia, assumenda ipsam optio esse nihil quas dicta laboriosam sint explicabo officiis, nam sint tempore fugiat non quaerat quia dolorum atque reiciendis animi, optio numquam earum illum architecto aliquid repellat quam fuga.Voluptas iste id, quia beatae ipsa ut magnam tenetur voluptatem tempora quas corporis vitae, provident eius vero deleniti eum maxime error optio explicabo, voluptates animi obcaecati accusantium saepe.Ad optio eligendi libero ipsum esse nostrum fuga vero temporibus maxime, at nulla odio exercitationem ad beatae.Asperiores nihil odit blanditiis ducimus harum perspiciatis similique veniam laborum, suscipit autem illum tempora laboriosam quae recusandae ullam dolor quo neque earum?Architecto officia facere exercitationem nostrum sequi illo nemo delectus sit, porro voluptate dicta aspernatur, culpa vero amet autem libero reprehenderit, vel iste eius magnam quisquam recusandae perspiciatis temporibus quas voluptatem possimus.