POMMERMAN: MULTI-AGENT LEARNING USING ACTOR-CRITIC METHODS

Kris Walther (s172990), Mirza Hasanbasic (s172987), Søren Hjøllund Jensen (s123669) and Yevgen Zainchkovskyy (s062870)

Technical University of Denmark

ABSTRACT

In 2018, a global challenge was given to make the best Pommerman agent. In this paper, we present two attempts to beat this challenge through Deep Reinforcement Learning, using the Actor-Critic learning methods A2C and A3C. To achieve this, we have implemented imitation learning to study the model complexity and boost the training time, yielding a fairly aggressive agent. A thorough discussion of the final input and network architecture are also presented in the paper. We conclude that A2C converges faster than A3C for this challenge and achieve win rates that slightly outperform the rule-based agents innate to Pommerman.

1. INTRODUCTION

In 2015, the aptly named *Alpha Go* became the first computer program to beat humans in the Chinese game *Go* [1]. To achieve this feat, Silver et al. used the concept of reinforcement learning to teach a deep neural network super human intuition. Since 2015, many new approaches have been proposed for improving deep reinforcement learning [2–4]. From these proposals, one of the more popular techniques have been *Actor-Critic* networks.

In an Actor-Critic approach, the neural network has to estimate the value of being in a state (critic), as well as select the appropriate action (actor). In this method, both the actor and critic are updated at the same time, but the critic is used to correctly reward the actor for its actions. Actor-Critic methods can either be implemented using asynchronous updates (A3C) [5] or by collecting several instances in a mini-batch update (A2C). In general, A2C has proven to perform better than A3C in terms of faster convergence [6, 7].

Recent multi-agent learning competitions and environments [8–10] have complex scenarios where many actions are taken before a reward signal is available. This provides a challenge, as the models need not only to learn the basic behaviours, but also learn overall strategic elements. To overcome this problem, recent studies have implemented imitation learning [1, 10, 11] where the model usually first learns from humans players in order to learn the basic concepts of the game.

In this work, we seek to achieve results in the Pommerman challenge by implementing an Actor-Critic network. Furthermore, we will use imitation learning to improve the agent's learning rate. Comparisons between A2C and A3C will be made in order to establish which method yields the best performance when training the Pommerman agent.

The remainder of this article is organised as follows: Section 2 describes the problem and Pommerman environment. Section 3 covers the intermediate results and theoretical background used to construct our agent. Section 4 presents the results obtained from testing the final agent, as well as performance analysis on the A2C and A3C methods. Lastly, limitations, improvements and future considerations are discussed in section 5 before concluding the article in section 6. Our full code can be accessed at GitHub via [12].

2. PROBLEM FORMULATION

Pommerman is played on an 11×11 discrete board where agents interact with the environment at each discrete time step t. The board is randomly generated by the simulator with passages, destructible and indestructible walls, along with four agents starting in each corner (an example of a board can be seen in fig. 3a in a later section).

In the beginning of each time step, the agent receives an observation of the environment's state consisting of the features listed in table 1. Given a state s_t, the agent chooses one action a_t from the action space $A(s_t)$. The six possible actions are *Stop*, *Up*, *Left*, *Down*, *Right* and *Bomb*. When a destructible wall is destroyed, there is a chance to reveal one of three different power-ups: Increased ammo, increased blast range or the ability to kick bombs.

Feature	Dimensions
Board (walls, agents, etc.)	11×11 integers
Bomb Blast Strength	11×11 integers
Bomb Life	11×11 integers
Ammo	1 integer
Blast Strength	1 integer
Position	2 integers in $[0, 10]$
Can Kick	1 integer in [0,1]
Teammate	1 integer in [-1,3]
Enemies	3 integers in [-1,3]

Table 1: State parameters given at each time step.

The Pommerman environment also includes two agents: A *Random Agent* (RA), which randomly samples actions from $A(s_t)$, and a *Simple Agent* (SA), which is a well-performing rule-based agent. These agents will be used as a benchmark for the performance of our agent. The goal is to defeat these agents with a win rate greater than 50% in the following scenarios:

1. Agent versus three RAs.
2. Agent versus one SA and two RAs.
3. Agent versus three SAs.

The Pommerman environment enables three different game modes: Free-for-all (FFA), 2v2 team, and 2v2 team with radio communication. The prospect of cooperative learning will therefore also be discussed, as this is the lesser discovered frontier of deep reinforcement learning.

Given the large ever-changing state space and varying effects of actions by multiple agents, the environment is largely non-Markovian – proper methods will be discussed in the following section.

3. MATERIALS AND METHODS

Before any sort of training is carried out, a baseline is created for the relevant scenarios. This baseline measures the given agent's performance based on 10,000 games. The resulting baseline from a SA can be seen in table 2 and fig. 1.

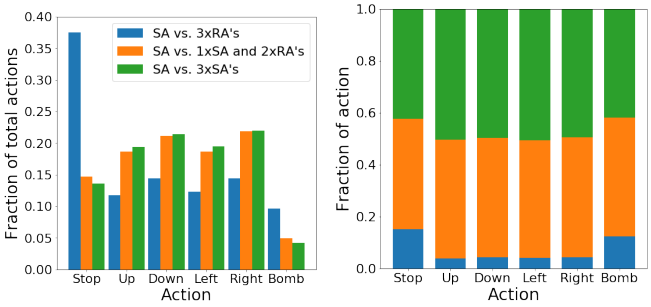
In the first scenario, one SA plays against three RA's. Here, the average game duration is 34.6 time steps. In these games, the SA dominates the RA's with 97.7% wins, 0.14% ties and 2.16% losses. In the second scenario we have one SA playing against one SA and two RA's, where the SA wins 42.3% of the games with an average duration of 250 time steps. The third benchmark measures four SA's playing against each other. Here we see an average game duration of 268 time steps. Out of these 10,000 games, the SA won 22.2 %, while 22.8% resulted in a tie. Interestingly, the main diagonal agents perform significantly better than off-diagonal agents, which suggests that either the map is not completely random or the SA favours some positions. The average SA win rate is then actually lower, at 19.3 %.

The action distribution of a SA in each game setting is seen in fig. 1a. It is apparent, that the *bomb* action overall is quite uncommon, while it is more frequent against RA's as these games tend to have shorter durations. Fig. 1b provides context to the action distribution; intuitively, the number of actions performed in each game mode are proportional to the average duration of the game.

3.1. Imitation learning

For the Pommerman environment, we implement imitation learning as it is often better to teach the agent the basic strategies before it learns by trial and error. We do this based on the actions of 10,000 games of the rule-based SA's rather than human games. SA's use Dijkstra's algorithm on each time step to collect power-ups, place bombs and avoid blasts [13].

In the SA code (line 108-112 of [14]), we see that the SA has a primitive memory of the last visited positions. We therefore know that in order to accurately imitate the SA, we must add memory via recurrent layers – further elaboration is presented in section 3.4. To implement imitation learning with recurrent neural networks, we log the game data as a sequence of 100 state and action pairs. The training problem for imitation learning now becomes a sequential classification problem with the actions encoded in a one-hot representation.



(a) (b)

Fig. 1: (a) Action distribution for each game setting independently. (b) Relative action distribution for each action across the game settings. Note the shared legend.

	Agent 0	Agent 1	Agent 2	Agent 3
		SA vs. three RA's		
Wins	97.7 %	0.6 %	0.9 %	0.8 %
Loses	2.3 %	99.4 %	99.1 %	99.2 %
Reward	0.953	-0.989	-0.983	-0.985
		SA vs. one SA and two RA's		
Wins	42.3 %	0.02 %	42.2 %	0.02 %
Loses	57.7 &	99.98 %	57.8 %	99.98 %
Reward	-0.154	-0.996	-0.152	-0.996
		SA vs. three SA's		
Wins	22.2 %	16.8 %	22.5 %	15.8 %
Loses	77.8 %	83.2 %	77.5 %	84.2 %
Reward	-0.555	-0.664	-0.551	-0.685

Table 2: Scores from 10,000 games with the three different game settings. Stats of Agent 0 – a SA – serves as benchmark for the performance of our agent.

3.2. Learning method strategy

The general network architecture with which we have chosen to engage the Pommerman challenge is based on the Actor-Critic method. This method is a hybrid of standard value-based and policy-based methods. The method trains two models for each agent; an actor and a critic. The critic model is trained to approximate the value function $V(s_t)$ which estimates the value of the current state. The actor model then handles the policy function $\pi(a_t|s_t)$, which decides the action to take. The

weights of the policy network θ are updated as

$$\Delta\theta = \nabla_\theta \log \pi(a_t|s_t)A(s_t, a_t), \quad (1)$$

while the weights of the value network θ_v are updated as

$$\Delta\theta_v = A(s_t, a_t)\nabla_{\theta_v} V(s_t)$$

Here $A(s_t, a_t)$ is an estimate of the advantage function

$$A(s_t, a_t) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}) - V(s_t), \quad (2)$$

where k can vary from state to state and is upper-bounded by t_{max}, while γ is the reward discount factor, and r_t is the reward received at time step t. However, the inventors of this Actor-Critic method [5] found that adding entropy to the policy update encourages more exploration. When adding entropy, eq. 1 takes the form

$$\Delta\theta = \nabla_\theta \log \pi(a_t|s_t)(R_t - V(s_t)) + \beta\nabla_{\theta'} H(\pi(s_t)) \quad (3)$$

where R_t is the total accumulated return from time step t, H is the entropy, and β is a hyperparameter that controls the strength of the entropy regularisation. Mnih et al. also suggested that the method can be further improved by using the *Generalised Advantage Estimation* (GAE) as done by Schulman et al. [15]:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V, \quad (4)$$

where $\lambda \in [0, 1]$ is a hyperparameter that controls a compromise between bias and variance, and $\delta_t^V = r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$ is the temporal difference residual. Thus we replace eq. 2 by eq. 4.

It has been observed that the skills learned by the agent are highly dependent on the discount factor, γ [16]. When this factor is low, the immediate actions taken by the agent are weighted strongly. Conversely, when this factor is high, the general strategy is weighted strongly. We therefore use an adaptive γ_j over the first 40.000 episodes, where j refers to the current episode:

$$\gamma_j = 0.5 + \frac{0.5}{1 + e^{-0.0003(j-20000)}}$$

The Actor-Critic methods we have chosen to apply are the Asynchronous Advantage Actor-Critic (A3C) and Advantage Actor-Critic (A2C). The difference lies in the synchronisation aspect: As illustrated in fig. 2a, the A3C method has n workers, which asynchronously updates the weights of the global network. For the A2C method in fig. 2b, the update of the global network will not happen until a coordinator has received input from all the workers.

3.3. Convolutional layers

Convolutional layers are often used to recognise patterns in images such as faces or objects. As objects on the game board are only represented using a single number, we do not need convolutional layers to recognise these. However, a recent experiment has attained promising results by using convolutional layers in the Pommerman challenge [17]. We speculate that the usage of convolutional layers allow the agent to find specific board patterns that are advantageous or disadvantageous. An example of such a pattern could be if an enemy walks into a hallway with no escape to either side.

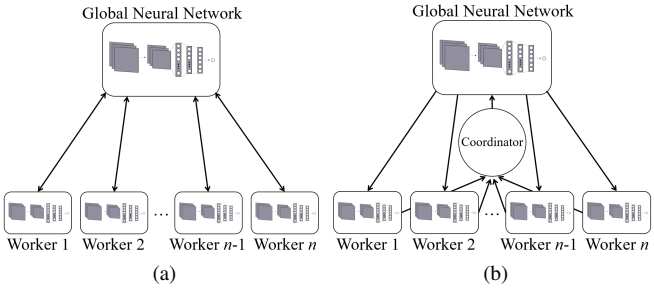


Fig. 2: (a) A3C method. (b) A2C method.

3.4. Adding memory

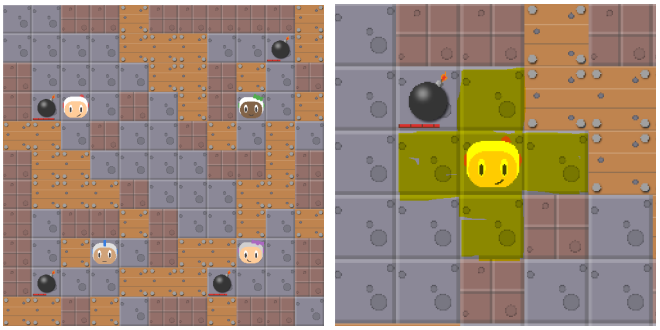
As the agent moves through the environment, some historic information is important: where was the agent before, and what action did it take to get here. To provide the agent with this information, we have added *Long Short Term Memory* (LSTM) layers to the actor, as well as provided the last six actions as input to the agent. Here, the previous actions are especially important during training, as the actions taken are sampled from a multinomial distribution, and can therefore not always be correctly reconstructed from the LSTM's memory.

3.5. Centering the view

One problem that arises when feeding the game data to the network is how the agent views the world. Therefore, having two different viewpoints may be advantageous. The first viewpoint used is the full game board, as illustrated in fig. 3a. Here, all walls and passageways are locked in place and only the agents and bombs can change place. The upside from using this point of view, is the consistency of the map. Another way to provide the map to the network is by centering it on the agent. To understand the benefits of using this map, we focus on the immediate surroundings and possible actions. The possible movement actions always depend on the environment in the four cells surrounding the agent. By centering the map on the agent, these cells are always in the same spot with respect to the agent, which allows the agent to construct a short term strategy. This point of view is illustrated in fig. 3b.

3.6. Estimating model complexity

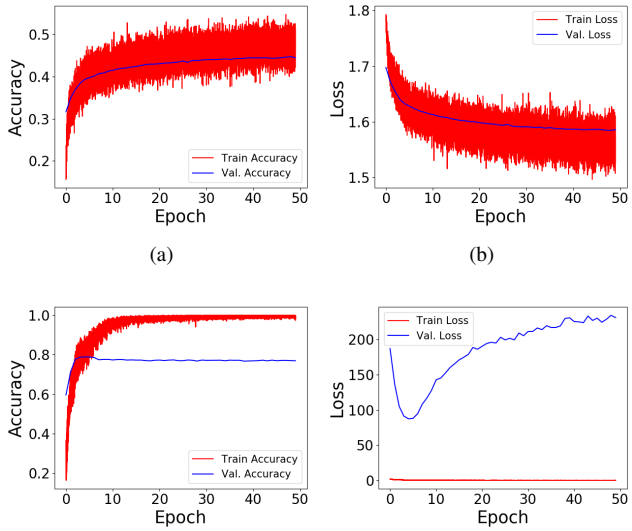
Finding a model complexity that is not too simple and not too complex is often difficult. To achieve this we use imitation learning on actions gathered from 10,000 SA games. If the



(a) (b)

Fig. 3: (a) Full game board. (b) 5x5 grid centered around the agent. Immediate action possibilities highlighted in yellow.

model is complex enough to learn how to act as a SA, we know that it is large enough to generalise a state-action mapping. In fig. 4 we see the result of attempting imitation learning on a too simple model and on the final model. Here we see that the accuracy for imitating the SA is increased when choosing the final model compared to the simple model. In fig. 4c we only



(a) (b)

(c) (d)

Fig. 4: (a) Training and validation accuracy for a simple model. (b) Training and validation loss for a simple model. (c) Training and validation accuracy for the final model. (d) Training and validation loss for the final model.

obtain an accuracy of 77% for the prediction of the SA's behaviour. Initially this seems very bad, but when we look at the implementation of the SA (line 112 of [14]), we see that the SA chooses a random action after eliminating bad and invalid actions. In the case where the SA chooses randomly between two actions, the model will learn to reduce the loss by predicting a medium value on both actions.

3.7. Final model architecture

The considerations described in sec. 3.2-3.6 as well as iterative improvements have resulted in the final model architecture as illustrated in fig. 5. In this model we provide two inputs. The first input provides the entire board from the state parameters *Board*, *Bomb Blast Strength* and *Bomb Life* from table 1. The board is padded with the value 1 as this represents indestructible walls, and the two other state parameters are padded with 0's. This input is passed through four convolutional layers and three fully connected layers, resulting in an encoded representation of the map state. This encoded representation is then fed directly to the critic and through two LSTM layers to the actor.

The second input consists of the flattened 5×5 grid around the agent for the same three state parameters, as well as the agent's *Ammo*, *Blast Strength* and its six previous actions. This input provides the immediate surroundings and is fed directly to the critic and LSTM layers.

Throughout the network we use the *Rectified Linear Unit* (ReLU) activation function. Batch normalisation is added in the A2C method, but cannot be used in the A3C method as inputs are always fed in a batch size of 1. Dropout was investigated for A3C as an alternative to batch normalisation, but was found not to be beneficial. The Adam algorithm is used for optimisation and for hyperparameters we use a learning rate of $\alpha = 10^{-5}$, a discount factor of $\gamma = 0.95$ (adaptive discount factor is used for imitation learning), while the hyperparameters of entropy regularisation and GAE mentioned in section 3.2 are $\beta = 10^{-2}$ and $\lambda = 0.9$ respectively.

4. EXPERIMENTS AND RESULTS

This section highlights the results obtained in this study. We first present a performance comparison of A2C and A3C training, which lays the foundation for which model we choose to train further. Hereafter, we present the final result after 150,000 epochs of training on the selected model.

4.1. A2C vs. A3C

The performance of the two Actor-Critic methods for training is seen in fig. 6. Here, the average reward is iteratively updated using:

$$\hat{r}_t = \hat{r}_{t-1} \cdot 0.999 + r_t \cdot 0.001,$$

where, $\hat{r}_t$ is the average reward at time step t, while r_t is the actual reward. From fig. 6, we see that A2C performs almost twice as well as A3C over the first 40.000 episodes.

4.2. Starting from imitation

When testing the A2C agent, we saw that it had learned to not place bombs in the hope of enemies killing themselves; a problem often encountered by other participants in the competition [17, 18]. To solve this problem and speed up training, we choose to start the model with the weights trained with imitation learning. To ensure that the critic has learned to reward the actor correctly, the critic is trained for 200,000 iterations while

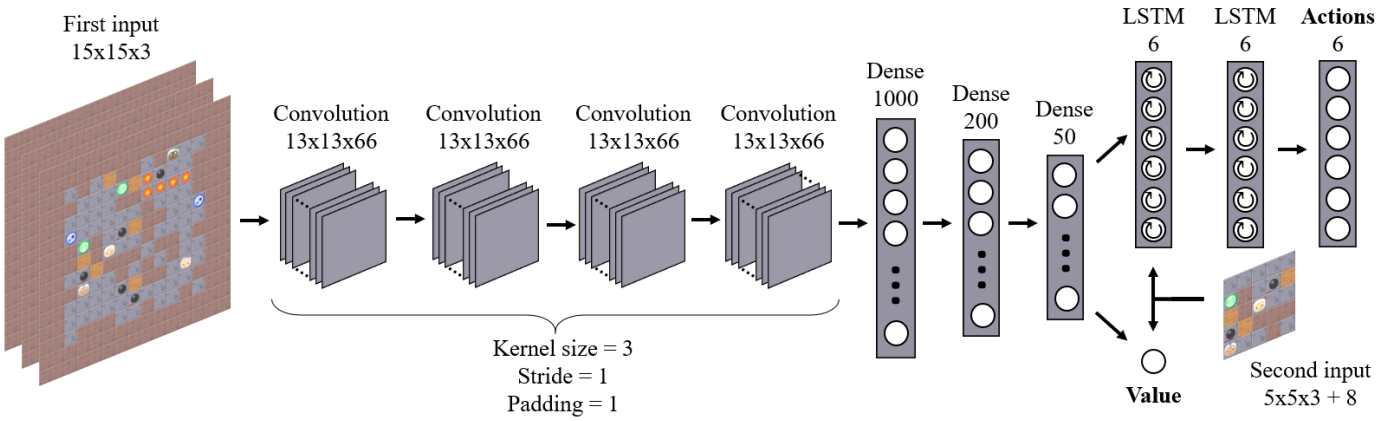


Fig. 5: Final model architecture for A2C and A3C networks with layer dimensions as specified and ReLU as activation function. The actor and the critic share the same network until they have passed through the fully connected layers. From here, the policy network passes through two LSTM layers before reaching the action output, while the value network passes straight to the value output.

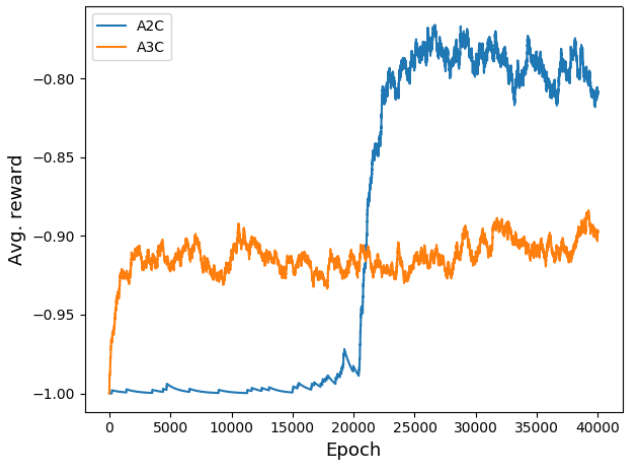


Fig. 6: Performance of A2C (blue) and A3C (yellow) wrt. average reward over 40.000 epochs.

the rest of the model weights are frozen. This training step is relatively inexpensive as it only requires updating a single dense layer.

In fig. 4c and 4d we see that the validation accuracy does not increase after the sixth epoch, and the validation loss increases hereafter. We therefore stop the model training after six epochs to avoid overfitting to the training dataset. Although the *bomb* action was underrepresented in the training data (fig. 1a), it is observed that the agent in fact does learn to place bombs at the correct positions.

4.3. Final results

The final training plot for A2C can be seen in fig. 7. Here, the model starts with a reward equal to that of a SA and increases for the first 50,000 epochs. Hereafter the agent learns some bad habits and loses performance until epoch 85,000 when the reward again increases. Unfortunately, we did not have time to perform more than 150,000 epochs, so we had to stop the training at this point. It was then observed that the agent had once again learned to not place any bombs and instead run away from the enemies. We therefore chose to use early stopping, and used the checkpoint saved at epoch 50,000 as the final model.

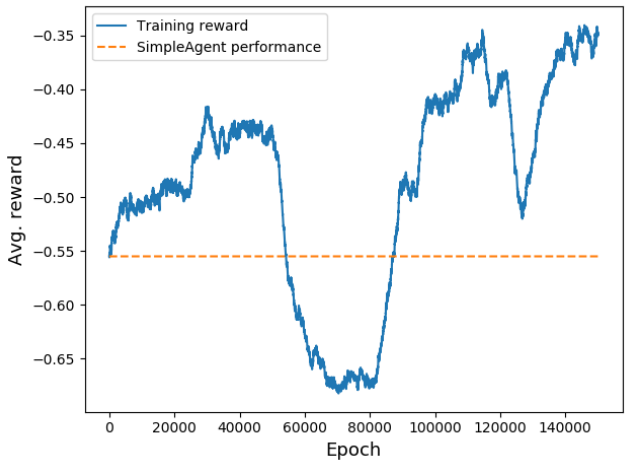


Fig. 7: Final reward during training over 150,000 Epochs

In fig. 8 we see the win rate of our agent trained by the final model for the three scenarios presented in section 2. Note that these win rates do not include ties, which means the performance of the rivalling agents are worse. Here, we see that we reach the goal of getting a 50 % winrate in scenario 1 and 2. Although we did not reach this benchmark in scenario 3, we still see a slightly better performance than that of a SA in the same position. Considering the average win rate of 19.3 % of four SA's found in the beginning of section 3, the performance is even better.

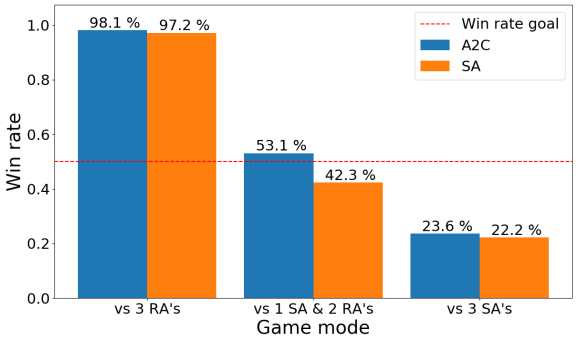


Fig. 8: Win rates of the three scenarios for our trained agent along with a SA for comparison. Each scenario is run for 10,000 games.

5. DISCUSSION

Reinforcement learning is a complicated and demanding topic, and so is the training of models within this field. The training time of our model is tremendously increased by the limitations on our hardware specifications, which has several consequences. First of all, it has impaired the important use of hyperparameter tuning, as the outcome of this cannot be studied across several ranges at sufficiently large epoch samples due to the long computation time. Second, it has not been possible to experiment with different inputs and outputs, which certainly has an impact on the performance of the model. E.g. the winners of the Pommerman competition [16] uses an action distribution of size 122 as output, while others use several state parameters as inputs [18]. In an attempt to remove these limitations, we tried utilising high performance cloud services such as AWS for the A3C model. However, we encountered difficulties using the `torch.multiprocessing` with AWS and abandoned this idea, ending up using our own PC's that could utilise a 8700k Intel CPU.

This limitation has been a problem, as other participants in the Pommerman challenge report, that when using advantage-Actor-Critic methods, critical results appear after millions of epochs of training [17, 19]. Training one million epochs alone would require 30 days of running time with our hardware. Other successful implementations include the policy gradient method called *Proximal Policy Optimisation* (known as PPO [20]), which reaches higher rewards at fewer training episodes [18], and other alterations to the advantage-Actor-Critic methods in the form of *Continual Match Based Training* [16] and *Terminal Prediction* [19]. Common to all the above mentioned implementations is, that similar network architectures to ours are used, typically involving convolutional layers. This indicates, that our problem of slow convergence does not lie here, but potentially in the implementation of the learning method.

In our model, a problem often arose, when using the weights of the imitated agent for further training. The results changed dramatically, implying hyper sensitivity to changes in specialised neurons. We attempted to combat this using a lower learning rate, batch normalisation in the A2C method and dropout in A3C. However, due to the large training time, we were not able to search the entire parameter space. Also, our final agent does not explore as much as desired. Tuning the hyperparameter β of the entropy term in eq. 3 might have countered this problem.

5.1. Towards cooperative learning

Finally, cooperative learning was not implemented, mainly due to the first point discussed in this section: Due to the computational limitations of our model, the focus has been to improve our trained agent for the specified scenarios.

New challenges arise when the team setting is played: Our model would now have to handle partial observability and the possibility for radio communication. These two aspects might be handled by changing the input and output of our model, but the greatest challenge would be the aspect of team learning. Action and state space would increase exponentially, depending on whether a homogeneous or heterogeneous team learning approach is used [21]. A crucial part is to teach the agents not to target each other as enemies. Proper reward shaping could prevent this, but as we have bitterly realised during this project, reward shaping alone rarely does the trick. Regarding the learning method, learning becomes difficult when observability is only partial. This requires decentralised policies which only depend on local observations of each agent. A centralised critic could then provide an indirect observation of entire state to each of the agents. This could be implemented using a multi-agent Actor-Critic method most suitable for our model, namely the *Decentralised Actor, Centralised Critic* as suggested by [22].

6. CONCLUSION

In this work, we have successfully trained an agent to play Pommerman using deep reinforcement learning. To achieve this, we have implemented both the A2C and A3C Actor-Critic methods and compared their convergence rate. Here, the best method was shown to be the A2C model which was then used to train the final model. We have also showcased successful imitation learning from the actions of the supplied agents. During training it was experienced that the agent became pacifistic and chose not to plant bombs. To avoid this, we stopped the training of the agent at a point where it was still aggressive and still outperformed the rule-based agent. Our final win rates were 98.1%, 53.1% and 23.6% in three increasingly difficult scenarios, respectively. These results were slightly better than the rule-based agent in all scenarios, although the desired win rate of 50% in the third scenario was not achieved. Due to limited hardware, we have not been able to do hyperparameter tuning, and it could therefore be possible to further improve our results by tweaking these. In further work, we therefore recommend exploring the possibility of utilising high performance cloud services with the Pommerman environment to speed up training of our model. However, this seems like a costly affair since A2C would only scale well on multi-GPU system clusters [16].

7. REFERENCES

[1] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al., "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484, 2016.

[2] Bahare Kiumarsi, Kyriakos G Vamvoudakis, Hamidreza Modares, and Frank L Lewis, "Optimal and autonomous control using reinforcement learning: A survey," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, pp. 2042–2062, 2018.

[3] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint arXiv:1708.05866*, 2017.

[4] Yuxi Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[5] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[6] "Policy gradient algorithms," https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#a2c, Accessed: 2019-01-01.

[7] "Baselines," https://blog.openai.com/baselines-acktr-a2c/, Accessed: 2019-01-01.

[8] "Capture the flag," https://deepmind.com/blog/capture-the-flag/, Accessed: 2018-12-30.

[9] Cinjon Resnick, Wes Eldridge, David Ha, Denny Britz, Jakob Foerster, Julian Togelius, Kyunghyun Cho, and Joan Bruna, "Pommerman: A multi-agent playground," *arXiv preprint arXiv:1809.07124*, 2018.

[10] "Pommerman pytorch," https://blog.openai.com/openai-five, Accessed: 2018-12-30.

[11] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al., "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning," *arXiv preprint arXiv:1807.01281*, 2018.

[12] "Project code," https://github.com/eugene/pommerman, Accessed: 2019-03-01.

[13] "Pommerman competition environment," https://github.com/MultiAgentLearning/playground, Accessed: 2018-12-18.

[14] "Simple agent code," https://github.com/MultiAgentLearning/playground/blob/master/pommerman/agents/simple_agent.py, Accessed: 2018-12-18.

[15] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[16] Peng Peng, Liang Pang, Yufeng Yuan, and Chao Gao, "Continual match based training in pommerman: Technical report," *arXiv preprint arXiv:1812.07297*, 2018.

[17] "Pommerman pytorch," https://github.com/rwightman/pytorch-pommerman-rl, Accessed: 2018-12-30.

[18] Dhruv Shah, Nihal Singh, and Chinmay Talegaonkar, "Multi-agent strategies for pommerman," 2018.

[19] Bilal Kartal, Pablo Hernandez-Leal, and Matthew E Taylor, "Using monte carlo tree search as a demonstrator within asynchronous deep rl," *arXiv preprint arXiv:1812.00045*, 2018.

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[21] Liviu Panait and Sean Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous agents and multi-agent systems*, vol. 11, no. 3, pp. 387–434, 2005.

[22] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, 2017, pp. 6379–6390.