EEMCS, Delft University of Technology

The Netherlands

Getting AI to Cooperate: Sharing a Critic in a Video Game

Author
J. J. H. Groenendijk

Supervisors:
Dr. F. A. Oliehoek
R. Loftin

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2023

Abstract

The popular video game "Overcooked" is a great example of a task requiring complex planning and cooperation with other players. This game is used as the inspiration for an environment for evaluating AI, called "Overcooked-AI". This paper implements a centralized critic into the Overcooked-AI environment's implementation of the PPO algorithm and compares the results with the decentralized critic approach when it comes to cooperation with human-like agents and computational efficiency.

The centralized critic approach gives similar results compared to the decentralized critic approach, both in self-play and when playing with human-like agents. This is probably due to the decentralized critic approach already having full access to the entire observation space, and no hyperparameter tuning being done due to a lack of time.

1 Introduction

The field of AI research has seen a lot of interest in research into the capabilities of AI in video games. Some reasons for this include that video games simulate difficult problems in repeatable, easily gradable scenarios which allow for rapid prototyping compared to real-world experiments. This makes them perfect for testing how well AI algorithms can be taught to perform certain tasks.

The popular video game "Overcooked" is a great example of such a task, since not only does it require complex planning, but it also requires cooperation with other players (which could be AI agents or humans). This research project will focus on the feasibility and efficiency of centralized critics for use with multi-agent reinforcement learning.

When using reinforcement learning in games with multiple players, it is best to have a shared method of evaluating how well each player is doing, rather than having each player learn on their own. This helps account for the fact that the actions of other players can affect the game and provides a more accurate way of measuring success. Centralized critics are a method of doing this by having a shared value function to see how well the agents are doing.

Currently, there is a popular framework for evaluating AI based on Overcooked, namely Overcooked-AI [1]. This framework has been used for evaluating many different cooperative AI strategies like training models specifically to team them up with humans [8], adding unit testing to see how the model responds to unpredictable actions [5], and obscuring information from the AI to see how it responds [9].

In this environment, currently, PPO (Proximal Policy Optimization) [11] is implemented. This is an On-Policy Optimization algorithm that allows for the optimization of a Markov Decision Process.

The goal of this research project is to implement centralized critics into this multi-agent reinforcement learning framework by implementing the MAPPO (Multi-Agent Proximal Policy Optimization) approach [12]. MAPPO is a good choice here because when we compare PPO and MAPPO, it will show the impact of the centralized critic well, and the central critic can be implemented with relative ease by using the RLlib library [6], which is a nice-to-have.

This research will answer the question of whether agents trained with a central critic can still work well with humans, or if they become too dependent on the other models the AI trains with. Specifically, it will answer the question: "How does MAPPO compare to PPO when it comes to efficiency and collaborating with human-like agents in the Overcooked-AI environment?"

In this paper, a basic overview of PPO is given. Then, the applied changes to the environment are shown, as well as the results of those changes. Afterward, the two trained models are compared with regard to computational efficiency (the training time of the algorithm), as well as flexibility when working together with other AI agents or even agents trained to behave like humans.

2 Background

To understand what is being done in this paper, it is good to understand what PPO[11] is and how it works. I will be giving a high-level overview of PPO for brevity's sake. PPO is a reinforcement learning network that trains a policy to perform a required task. To be precise, it optimizes a Markov Decision Process, which is a function that maps a state to a list of actions and their probabilities. To aid in this, it uses a second function, called the value function, which is used to evaluate the average reward of a given state. The step-by-step process of how (the clip version of) PPO is implemented is:

1. Initialize the policy and value function networks: Start by initializing the policy and value function networks with random weights. These networks will be trained to improve the agent's policy and estimate the state-value function.

2. Collect trajectories: Interact with the environment by running the current policy to collect trajectories. A trajectory consists of states, actions, rewards, and the next states.

3. Compute advantages: Calculate the advantages for each state-action pair in the collected trajectories. Advantages represent how much better or worse an action is compared to the prediction from the value function.

4. Compute policy ratio and surrogate loss: Calculate the ratio between the probabilities of choosing our current action in the new policy and the old policy for each state-action pair. Compute the surrogate loss, which is a combination of the policy ratio and the advantages. This helps ensure that the policy update is within a certain range.

5. Update the policy: Use the surrogate loss to update the policy network. This is done by optimizing the objective function using a gradient ascent algorithm, such as stochastic gradient descent (SGD) or Adam [4].

6. Update the value function: After updating the policy, update the value function network. Use the collected trajectories to train the value function network using a regression algorithm.

7. Perform multiple iterations: Repeat steps 2-6 for a specified number of iterations or until convergence. Each iteration involves collecting new trajectories, computing advantages, and updating the policy and value function.

The specific part of this algorithm that turns this algorithm into PPO is step 4. Here, the surrogate loss is calculated in such a way that does not let the algorithm make too drastic changes.

To go into detail: Take:

- p_{new} to be the proposed new policy
- p_{old} to be the current policy
- r to be the ratio of the chances of taking the current attempted action in the current policy and the new policy
- A to be the advantage, representing how much better the action is than the average action in the current state
- S to be the current batch of N state action pairs
- ϵ a hyperparameter, usually around 0.1 or 0.2 [2]

The surrogate loss is calculated to be:

$$L_{p_{old}}(p_{new}, S) = \sum_{s,a}^{S}(min(r \cdot A, clip(r, 1-\epsilon, 1+\epsilon) \cdot A))/N$$

This means that if the change is quite drastic (the ratio of probabilities r is further than ϵ away from 1), the loss is set to $1-\epsilon$ or $1-\epsilon$, and since this does not depend on our policy, the gradient of this loss with respect to our parameters is 0. There is one slight caveat to this, namely that the clipped value is not taken directly as the output, but it is actually the minimum of the clipped value and

the ratio multiplied by the advantage. This has the effect that if our advantage is negative and we are more likely to take this choice, the loss is not clipped, and if our advantage is positive and we are less likely to take this choice, the loss is not clipped.

The value function used in the overcooked-ai environment is a bit special because it's not a network on its own. The output of the policy is both an action and a value. One could see this both as a value function that takes in just a state (called a V function) because the input to the whole function is a state, or one could see this as a value function that takes in both the state as the input and a proxy value of the action (since the final action is computed from these inputs). This type of function is called a Q function.

In the MAPPO implementation, I explicitly pass in the action, the opponent's action, and the environment to the value function, so it is clearly a Q function. More info on this can be found in Section 3.

To obtain the advantages in the overcooked-ai environment, a method called Generalized Advantage Estimation [10] is used. This takes the rewards and values at every step, as well as two hyperparameters, γ and λ to calculate an estimate for the advantages.

3 Contribution and Methodology

To answer the research question, I will be amending the PPO algorithm with a centralized critic. A centralized critic is a way to give the value function more information, with the goal of speeding up the learning process by giving the critic insight into the strategy of the other agent. In other cases, this would also include the other agent's observation, but since there are no partial environments (a.k.a. every agent has all the information available to it at runtime), this is unnecessary.

After implementing this version of the PPO algorithm, the algorithm (now modified into MAPPO), will be used to train an agent with the same hyperparameters as the PPO agent. If I had more time, I would modify the hyperparameters following the advice of the authors of the original paper [12]. Using this agent trained with MAPPO in a self-play scenario, I will compare it against a PPO self-play trained agent in multiple ways.

1. Computational Efficiency: Do we need less training time to achieve the same results?
2. Self-Cooperation: How does it perform compared to PPO when paired with an agent using the same algorithm as itself?
3. Human-Like-Cooperation: how does it perform compared to PPO when paired with a human-like agent, created using Behavioural Cloning (BC)?

This research will grant us valuable insight into the value of a centralized critic. Especially since care will be taken to modify the environment as little as possible besides adding the centralized critic, allowing for accurate assessment of the impact of adding such a centralized critic, especially in this case where there are no partial observations, and the centralized critic only gets access to the opponent's action as the extra information.

Since centralized critics are a popular addition to the Machine Learning landscape, a lot of research has already been done using centralized critics. However, data on the effectiveness of centralized critics in a full observation environment, when compared with Decentralized critics, is scarce and will be valuable for the cooperative AI research landscape.

Some papers contrast Centralized and Decentralized critics, like [7], but they mostly focus on environments with partial observations because this is where multi-agent setups and centralized critics make the most impact. These other papers also do not look at the performance in terms of cooperation with humans or human-like agents, which my paper does.

4 Experimental Setup and Results

4.1 Experimental Setup

The results in this paper have been obtained by taking the latest version of the overcooked-AI GitHub repository[1], forking it[2], and implementing a centralized critic in a method similar to the example given by the Ray RLLib[3]. After this, I have trained the PPO/MAPPO models in a self-play scenario as given by the file human_aware_rl/ppo/run_experiments.sh. I have used the hyperparameters supplied by the original authors, and have not done any hyperparameter tuning myself due to lack of time. I have trained the Behaviour Cloning agents used for these results using the human_aware_rl/imitation/reproduce_bc.py.

After this, I obtained my results by moving my agents to the evaluate_centralized_critic/models directory and running the evaluate_centralized_critic/evaluate.py and the evaluate_centralized_critic/visualize.py scripts to analyze the results and obtain my graphs.

The main difference in my MAPPO implementation when compared to the original PPO implementation is the setup of the value function. In MAPPO, the value function uses information from both agents, not just from the agent that is being evaluated. The difference in setup is shown in figure 1.

[1] https://github.com/HumanCompatibleAI/overcooked_ai
[2] https://github.com/jellejurre/overcooked_ai
[3] https://github.com/ray-project/ray/blob/master/rllib/examples/centralized_critic.py

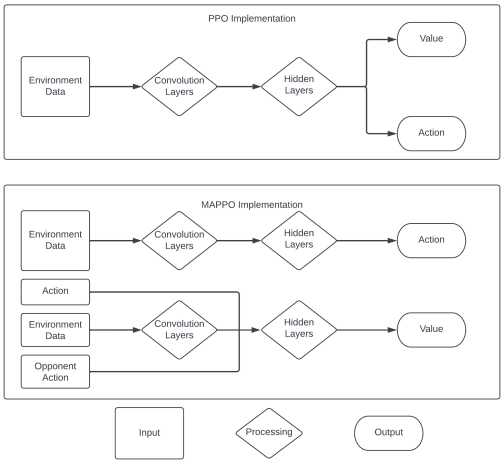


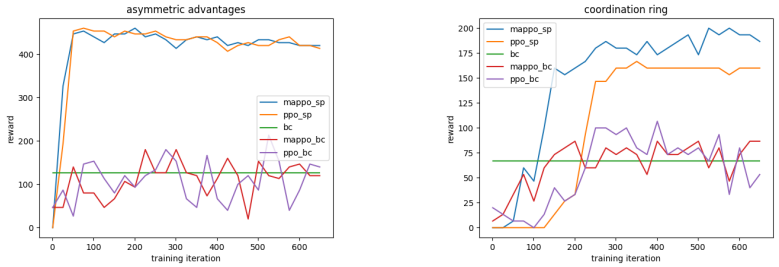
Figure 1: The difference in setups between the PPO and MAPPO setups of overcooked-AI

4.2 Results

I have trained MAPPO and PPO agents in a self-play scenario on four out of the five supplied maps (Counter Circuit wasn't included due to technical difficulties with this map). I have used the AgentEvaluator class to evaluate the agents against each other, every time using 3 games of 400 steps. The combinations I am using are:

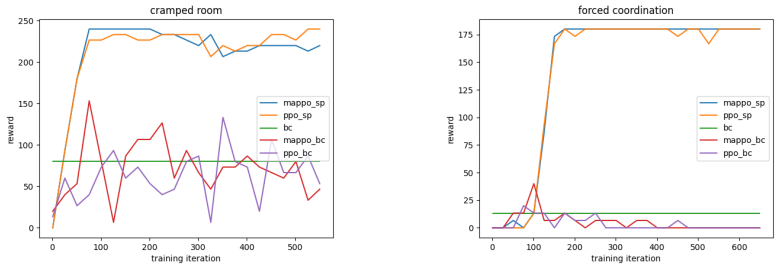
1. MAPPO self-play vs MAPPO self-play
2. PPO self-play vs PPO self-play
3. MAPPO self-play vs Behaviour Cloning
4. PPO self-play vs Behaviour Cloning
5. Behaviour Cloning vs Behaviour Cloning

On these maps, I have put agents from different training iterations against each other. Note that since BC doesn't have training iterations, I have used the same BC agent for every comparison.



(a) Asymmetric Advantages

(b) Coordination Ring

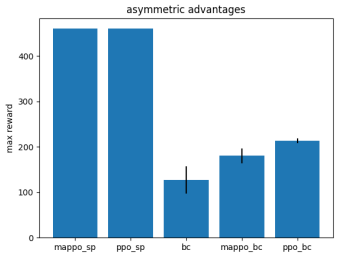


(c) Cramped Room

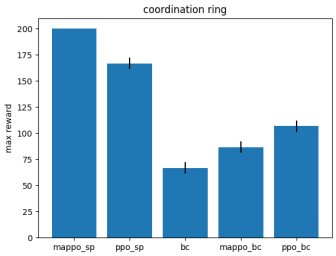
(d) Forced Coordination

Figure 2: The rewards of comparing different agents over training iterations

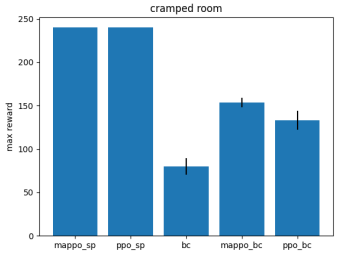
We can see in figure 2 that the results between MAPPO And PPO seem quite similar in their results. One interesting point is that in the coordination ring layout, which is the layout where the agents have to coordinate most in their movement due to the relative ease of blocking each other, the MAPPO self-play agent outperforms the PPO self-play agent, both when paired with itself as when paired with the behaviour cloning agent. However, when paired with itself, the gap remains during the entire training process, and when paired with the behaviour cloning agent, it about evens out after 200 training iterations.



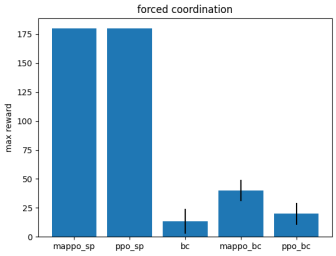
(a) Asymmetric Advantages



(b) Coordination Ring



(c) Cramped Room



(d) Forced Coordination

Figure 3: The max scores and their standard errors when comparing different agents

In figure 3 we can see the maximum mean value that was reached over all the training iterations, and the standard error over the three games which obtained that mean value. Here again, we can see that in most cases, MAPPO vs MAPPO and PPO vs PPO are very similar, with the only exception here being Coordination Ring's MAPPO vs MAPPO outperforming its PPO vs PPO score by about 35 points. Behaviour cloning, on the other hand, seems less stable, with MAPPO sometimes outperforming PPO, and vice versa.

I think the lower difference than what we would expect in most maps is due to the centralized critic being less of a benefit when both agents already have access to the full observation space since instead of both learning what the other agents are seeing and doing, they only gain access to what the other agent is doing compared to a decentralized critic approach. Another explanation for the lack of differences is the lack of hyperparameter tuning due to a lack of time.

5 Responsible Research

My work is reproducible by setting the "centralized_critic" flag to true in the config and then performing the steps to reproduce the agents from the original paper. It is written in a way that does not affect the original PPO agent, yet still allows for easy addition of more agents or interaction with an MAPPO agent.

The framework used uses the MIT license, and I am following this license.

For the behaviour cloning, the data used was obtained from the original paper by Caroll et al. [1], for which they completely anonymized the data. No human data was obtained for this paper alone.

6 Related work

[1] has made the environment this paper is based on. They research how well PPO and Behaviour cloning can work together with human test subjects. They do prune their data to throw out bad runs, but this is understandable as their limits on this are quite reasonable (worse performance than a single human). Their research shows models trained with a human-like model work together with other humans quite well compared to self-trained models. My paper looks into whether a centralized critic helps with this or not.

[12] looks into the performance of multiple multi-agent algorithms, and introduces MAPPO as an algorithm. They show MAPPO has good sample efficiency and performs well compared to other algorithms, although they do call their data an "empirical analysis", and say that their paper "does not directly analyze the theoretical underpinnings of PPO" (p. 10).

[7] compares Reinforcement Learning methods with centralized agents and critics and decentralized agents and critics, and they come to the conclusion that "policies trained by centralized critics avoid more-biased estimates usually produced by decentralized critics, but in return suffer more variance in the training process" (p. 6). It is nice to have such a direct comparison of centralized and decentralized critics and agents, although our results differ in how much the centralized critic matters, probably either due to their observation space being a partial space, or due to the lack of hyperparameter tuning on my end.

7 Discussion And Future Research

The original work trains agents both in self-play and Population Based Training (PBT) [3], however, I have based my work on the latest version of the repository to get access to their RLLib implementation (their old implementation uses TensorFlow's Stable Baselines) for faster computation and easier implementation of the centralized critic. This means that I do not have access to the PBT implementation and cannot run my tests with these agents. This is one area the future research could expand on.

If I had time to do hyperparameter tuning, I would follow the advice of Yu et al. [12] and increase the batch size, lower epoch count and tune the clipping parameter, which would hopefully lead to better final performance, and faster convergence, so future work could verify if my results still hold with more specifically tuned hyperparameters.

References

[1] Micah Carroll et al. *On the Utility of Learning about Humans for Human-AI Coordination*. 2020. arXiv: 1910.05789 [cs.LG].

[2] Wouter van Heeswijk. *Proximal Policy Optimization (PPO) explained*. Jan. 2023. URL: https://towardsdatascience.com/proximal-policy-optimization-ppo-explained-abed1952457b.

[3] Max Jaderberg et al. *Population Based Training of Neural Networks*. 2017. arXiv: 1711.09846 [cs.LG].

[4] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[5] Paul Knott et al. *Evaluating the Robustness of Collaborative Agents*. 2021. arXiv: 2101.05507 [cs.LG].

[6] Eric Liang et al. *RLlib: Abstractions for Distributed Reinforcement Learning*. 2018. arXiv: 1712.09381 [cs.AI].

[7] Xueguang Lyu et al. *Contrasting Centralized and Decentralized Critics in Multi-Agent Reinforcement Learning*. 2021. arXiv: 2102.04402 [cs.LG].

[8] Patrick Nalepka et al. "Interaction Flexibility in Artificial Agents Teaming with Humans". In: July 2021.

[9] João G. Ribeiro et al. *Assisting Unknown Teammates in Unknown Tasks: Ad Hoc Teamwork under Partial Observability*. 2022. arXiv: 2201.03538 [cs.AI].

[10] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2018. arXiv: 1506.02438 [cs.LG].

[11] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[12] Chao Yu et al. *The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games*. 2022. arXiv: 2103.01955 [cs.LG].