Playing Overcooked-AI with deep reinforcement learning

Leonardo Monti
University of Bologna
leonardo.monti3@studio.unibo.it

Abstract

This project is about the implementation of agents for playing the benchmark video game Overcooked-AI using deep reinforcement learning. The game requires two agents to act cooperatively and to coordinate in order to score points. The training of the agents is done with the state of the art algorithm Proximal Policy Optimization adapted to the multi agent context. We train agents capable of solving different layouts of the game with satisfactory performance. The drive of this study is to investigate how the numerous hyperparameters and low level choices of the algorithm affect the training process and the ability of the agents to generalize to several environments, including unseen ones.

1 Introduction

In the domain of multi agent reinforcement learning the ability of the agents to coordinate and cooperate are of fundamental importance to solve tasks like playing Overcooked-AI [2]. In the game the agents collect onions, put them to cook, collect dishes and serve the cooked onion soups. Each step happens in a fixed position within a kitchen. The layout of the kitchen includes obstacles that cannot be traversed by the agents. There exist five layouts of different sizes and designs. The layouts are designed to require a specific cooperative behavior from the agent. For example cramped room requires the agents not to stand in each other's way and forced coordination requires the agents to pass objects between each other.

In this project we implement from the ground up an instance of Multi Agent Proximal Policy Optimization (MAPPO) [4]. The implementation relies on a centralized computing infrastructure, so the experience collection and the policy optimization are performed alternatively on the same node. We use the implementation to train the agents to play the game and put to test the implementation choices. We do a series of experiments to test the agents' coordination to solve each layout separately. Then we test the agents' generalization to learn more than one layout at the same time. We also test the generalization to solve a layout that was not included in the training.

2 Training Algorithm

Training algorithm: The Proximal Policy Optimization algorithm (PPO) [3] consists of learning a policy, or actor, and a value function, or critic. The actor learns to select the actions, the critic evaluates the states so the actor can be informed on how better or worse than average is an action in a certain state. For this project the version PPO clip was selected. It prevents catastrophically large changes to the policy by clipping the updates.

The policy is trained maximizing the following objective:

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

$$L(s,a,\theta_{old},\theta) = \min(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}\delta_{ADV}^{\pi_{\theta_{old}}},\ clip(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}, 1-\varepsilon, 1+\varepsilon)\delta_{ADV}^{\pi_{\theta_{old}}}) + \alpha\mathcal{H} \tag{1}$$

Where

- θ_{old} are the parameters of the policy used to collect experience
- θ are the parameters of the policy at this point in training
- $\mathcal{H}$ is the entropy of the policy, maximized to boost exploration at the beginning
- $\delta_{ADV}^{\pi_{\theta_{old}}}$ is the advantage function: it increases or reduces the probability of an action that is better or worse than the average outcome of a state

The value function is trained minimizing the following loss function:

$$L_V = MSE(V^{pred}(s), V(s)) \tag{2}$$

3 Implementation choices

3.1 Neural network structure

The available options among feed forward neural networks are convolutional neural networks (CNN) and multilayer perceptrons (MLP). CNNs offer the capability of spatial reasoning: they naturally work with the spatial relationships between the layout features, like the position of a soup with respect to an agent. Unfortunately the need to generalize to different layout sizes means that we would have to transform the inputs to a common size: convolution itself is translational invariant but the head of the networks need to be of a fixed size. Padding smaller layout would make the training computationally inefficient and cropping the larger layouts would likely make them nearly impossible to be solved. The Overcooked environment offers a function to featurize the state into a compact representation of constant size. This possibility lead to the choice of using MLPs for the policy and the value function.
For the training of the neural networks we used the TensorFlow library [1].

3.2 Centralization of the value function and parameter sharing

The need of cooperation, the shared rewards and the full observability of the environment lead naturally to the decision of using one centralized critic for both the agents. For the policy the most general implementation would be to train one policy network for each agent, but this would be a waste since all agents play the same role and cannot predict in which of the two possible starting position it will play. So the policy networks share their parameters: we train the same parameters to play as both the agents.

This approach follows the Centralized Training Decentralized Execution (CTDE) framework because after training each agent is independent of the other. The key concept is that the same policy is applied twice on separate agents, instead of having one policy that decides the actions of both agents as would happen in team learning. Each network is optimized independently using the Adam optimizer.

We could further shrink the amount of trained parameters by using a shared backbone for the policy network and the value network. We chose not to do this because it would diverge from the CTDE framework and would overload the MLP network.

3.3 Strategies for generalization

The generalization aspect of this project is learning different layouts. This choice a consequence of sharing the parameters of the agents. This architecture would complicate changing the behavior of one of the agents during training.

3.3.1 Random sampling of layouts

In order to achieve layout generality the desired number of original Overcooked-AI environments were encapsulated into a general environment. The general environment samples with uniform probability the layout for each episode.

3.3.2 Curriculum learning

In order to obtain convergence faster than in a multi layout environment we trained the agents on one layout at a time alternatively for different numbers of episodes. This provides to the agents a simpler task at a time and achieve convergence faster because gradients are more stable. The effect of this technique is that it provides a better starting point for the full sampled environments training.

3.4 Standardization of inputs and network initialization

Although it is a standard practice of training neural networks, the inputs were not standardized. This is because the standardization of the input would need to be done with rolling estimates of average and standard deviation. This would alter the distribution of the observations during training and destroy convergence. Moreover, the linear layer of the networks can learn to do normalization if it is advantageous because it is a linear operation.

The layers of both networks were initialized with the Orthogonal initializer for weights and constant to 0 for biases. The constant 0 biases in the policy are needed to center the distribution of actions at the beginning, in order not to introduce a bias toward a certain action. This process makes normalization of the input observations less relevant because it does not make the assumption of the data being standardized. On the other hand the default initialization of the weights (Glorot) aims provide the best starting point possible for standardized data to the gradient descent algorithm.

3.5 Hyperparameter search

The implementation exposes several hyperparameters and the performance is highly sensible to them. In order to not exceed computation time limits an exhaustive search on all the hyperparameter space was not possible. Hyperparameters were optimized individually, without testing all the combinations. Each set of hyperparameters was evaluated for the speed of convergence and average score on the single layer "cramped room". Table 1 reports the value of the most relevant hyperparameters.

Table 1: Hyperparameter values

Name	**Value**
policy_hidden_sizes	(256, 128)
critic_hidden_sizes	(128, 64)
activation	leaky_relu
lr_policy	3e-4
lr_critic	1e-3
max_grad_norm	0.5
use_lr_decay	False
discount_gamma (γ)	0.99
advantage_lambda (λ)	0.95
horizon	400
ppo_buffer_size	4000
batch_size	256
n_epochs per batch	7
ppo_eps (ϵ)	0.3
entropy_factor (β)	0.05

Worth of noticing are the small sizes and shallowness of the networks.

4 Experiments and results

The results were measured by running the agents 20 times for each configuration.

4.1 Single layout

Table 2 shows that the baseline single layout were mastered in a reasonable time. Only the layout "counter_circuit" was completely unsolved.

Table 2: Scores on single layers

Layout	Avg score	Std dev	N episodes
cramped_room	127	17.06	1000
asymmetric_advantages	165.00	21.79	1000
coordination_ring	62	32.80	1000
forced_coordination	72	18.33	1000
counter_circuit	0.00	0	1000

4.2 Unseen layouts

Table 3 shows that the agents, after being trained on only one layer, were not able at all to play on a layout they had not seen before. This is the first sign that the learned policies are not general with respect to the layer but rather a memorization of a set of actions.

Table 3: Scores on unseen layouts

Trained on	Evaluated on	Avg score	N episodes
cramped_room	asymmetric_advantages	0	1000
asymmetric_advantages	cramped_room	0	1000
coordination_ring	forced_coordination	0	1000
counter_circuit	coordination_ring	0	1000

4.3 Multiple layouts

Table 4 shows that despite the efforts the algorithm has a hard time to generalize to multiple layout. Further tests with minor hyperparameter tuning lead to equally low performances. In particular it often happens that the agent solves one of the layout, this makes the gradients from that layout stronger which result in overfitting.

Table 4: Scores on multiple layouts

Layout 1	Layout 2	Avg Score 1	Avg score 2	N episodes
coordination_ring	counter_circuit	3	0	1000
coordination_ring	forced_coordination	112	0	1000

4.4 Curriculum learning

Table 5 shows the history of a training done with curriculum learning on two layouts. The number of episodes to train for each layer was decided to be close to the number of episode that is needed for each layer to start receiving consistently some reward. This was finally sufficient for the agent to achieve a satisfactory score on both the layouts.

5 Conclusion

This project demonstrates that a pair of agents trained with MAPPO can master one level of Overcooked-AI at a time. On the other hand the learned policies are not general, since they are extremely specific to the layout used for training. This suggests that the policies do not learn a deep understanding of the game but rather a sequence of actions that are good for the layout.

When put to test to the larger problem of learning two set of layouts at the same time, the agents did not learn an effective policy within the 1000 episodes that they were exposed to for training. This is most likely due to the hyperparameters optimized on the single layout problem, and to the restricted number of episodes. It is possible that sampling the environments with probability inversely proportional to their score would make the agent learn reliably both the layouts.

Breaking down the problem and presenting one part at a time to the agents simplified the learning task enough to achieve good scores after being trained on 1208 episodes. Training on the layouts

Table 5: History of training with curriculum learning

Layout 1	Layout 2	Avg score 1	Avg score 2	N episodes
cramped_room	-	NA	NA	333
asymmetric_advantages	-	NA	NA	250
cramped_room	-	NA	NA	125
asymmetric_advantages	-	5	154	250
cramped_room	asymmetric_advantages	127	119	250

separately was not enough because the agents would forget how to play the previous layout, but it provided a good starting point for achieving good result when finely trained further.

The key finding of this project is that even a simplified benchmark like Overcooked-AI requires a substantial amount of computing effort to be mastered. While this implementation meets it primary requirements, the results suggest that more hyperparameter exploration and more, finer training could lead to better performances.

CNNs, at least for the policy network, are promising to improve generalization because of the possibility of spatial reasoning. To overcome the difference of size input one could train a fully convolutional encoder that would reduce the spatial dimension and only output features. Such network would be substantially larger than the ones used for this project and would need more computation to be trained.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems*, 32, 2019.

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

[4] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games, 2022.