Overcooked Experiments (Project 3)

Steven Nord
snord3@gatech.edu
Hash:
4a74490e6888c9a4cd7f09c79856947bf001c399

Abstract— **Throughout this report, different techniques will be explored for Multi-Agent Reinforcement Learning (MARL) challenges. The techniques will be applied to a single algorithm (Deep Q-Networks) to allow for analogous analysis. Experimentations will be explored with the DQN algorithm (used in Project 2) using the popular game Overcooked.**

I. Introduction

Overcooked is a multi-agent environment with a discrete state-space and action-space. Deep Q-Networks (DQN) (used in Project 2) is the lone algorithm discussed throughout this report. Since there are multiple agents interacting within the environment, it presents challenges to the algorithm out of the box. As a result, various techniques will be applied to demonstrate how single-agent algorithms can be modified to work in multi-agent settings.

Since DQN is primarily a single-agent algorithm, a baseline is established initially by training a single agent to solve the environment. The hypothesis is that a single agent would be able to complete some of the layouts, but would struggle on others. Also, collaboration between the two agents would need to be established to achieve success beyond introducing another agent.

II. Overcooked AI

There are two agents/chefs that interact with the environment and the objective is to serve as many soups as possible in 400 timesteps. An agent is presented with a 96-dimension state-space. Some of the attributes in the state-space range from the agents' position in the kitchen, whether an agent is holding any ingredients, and ingredients currently in a pot.

The actions for each chef are as follows: move up, down, right, left, do nothing, or interact with the environment. The result of interacting with the environment varies based on the position and orientation of the chef in the kitchen and the object near them. Table 1 shows the possible events that could result if the chef's and respective object's positions align. Nothing happens to the agent or the environment if these do not align.

Table 1

Events	Initial Rewards	Phase I Rewards	Custom Rewards
Onion Pickup	0	0	1
Onion Drop	0	0	1
Dish Pickup	0	3	1
Dish Drop	0	0	1
Place Onion in Pot	0	3	5
Starting Stove	0	0	10
Starting Stove Early	0	0	-20
Soup Pickup	0	5	10
Soup Drop	0	0	-10
Soup Delivery	20	20	200

The tasks needed to successfully create a soup include picking up and placing three onions in a pot, then start the stove. Chefs are only able to carry one onion at a time, so getting three onions in a pot requires several sequential actions to occur. The stove can be started at any point the pot has an onion inside, but soups with less than three ingredients do not count as complete and are not rewarded as a delivered soup. Once the stove is started, it takes 20 timesteps to cook, regardless of whether or not the soup is full of the correct ingredients.

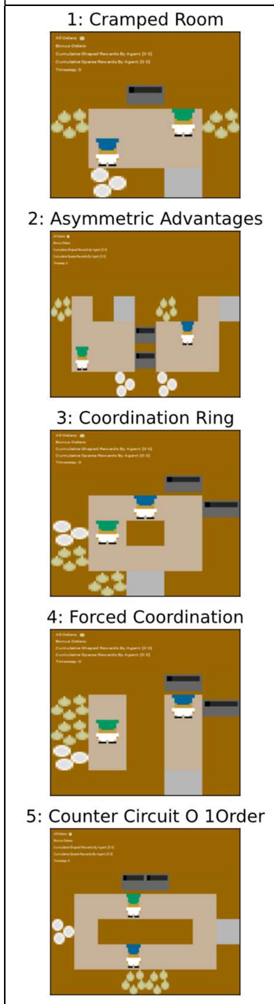


Figure 1: Layouts

The delivery portion of the process requires agents to walk over, pick up a dish, and then walk it over to the stove. When the soup is complete, the agents need to plate the soup (e.g., Soup Pickup in Table 1) and walk it to the serving counter to deliver.

It does not matter which chef does which task as the challenge of delivering completed soups is a collaborative effort.

Not only are agents tasked with making soups as quickly as possible, but they also must do so in several different kitchen designs. Figure 1 on the previous page shows the different layouts the agents must master as a team.

Episodes are deemed a successful operation if the chefs can prepare and deliver seven successful soups in 400 timesteps.

III. Deep Q Learing Algorithm (DQN)

DQN is primarily a single-agent algorithm, but techniques are implemented throughout experimentation to adapt it to multi-agent settings. A major advantage that DQN has is that it is easy to implement and can be built upon from Project 2.

There are several obstacles to overcome when implementing multi-agent environments. One of the main disadvantages that arises for DQN is that it assumes a stationary environment. If an agent is only aware of its own action selection, then the environment is responding to multiple inputs the individual agents are unaware of. This will be discussed in more detail later.

The pseudocode for the DQN algorithm can be seen in more detail below. This algorithm serves as a reference point that will be tweaked and discussed in later sections.

```
DQN Algorithm
Input:
  $\gamma$, ε_decay_rate, ε_min, n_hidden_nodes, $\alpha$, $\tau$
Algorithm parameters:
  memory replay $D$ with capacity (N)
  action-value function $Q$ with random weights
for $\text{episode}_i$ in number of episodes (M):
  for t in T:
    Choose $\text{action}_t$ for $\text{state}_t$ using ε-greedy policy
    Execute $\text{action}_t$ and observe $\text{reward}_t$ and $\text{state}_{t+1}$
    $\varepsilon = \max(\varepsilon * \varepsilon\_decay\_rate, \varepsilon\_min)$
    Store transaction $(s_t, a_t, r_t, s_{t+1})$ in $D$
    Randomly sample minibatch from $D$
    If $s_{t+1}$ is None: (meaning at terminal state)
      $y_t = r_t$
    else:
      $y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a_{t+1})$
    Calculate loss ($L$) between $Q(s_t, a_t)$ and $y_t$
    Update Q with SGD by minimizing $L$
```

IV. Experiments

Experiments will focus more on the impact of various adaptions made to DQN and less on hyperparameter tuning. Adaptions include the following:

A. Reward Shaping Techniques
B. Single Agent (Baseline)
C. Decentralized Training
D. Centralized Training

Experiments are done in an iterative process. First, reward shaping is improved to help agents learn subtasks for the ultimate goal of making soups. Then, the agents' trained behavior is observed from the modified reward structure. Finally, the process repeats.

The following figures show the rolling average (solid lines) and the spread of one standard deviation (corresponding colorful bands) of episode results to help make plots clearer. The rolling window is set to last 25 episodes.

A. Reward Shaping Techniques

Table 1 shows the of incremental progression of the reward structure provided to help the chefs learn the necessary actions to make soups.

The initial rewards are much too sparse for the agents to learn. Each agent is randomly performing actions, but receiving no feedback during the cooking process. This results in the agents never learning to make any soups with this reward structure.

Phase I reward structure allows for agents to learn how to operate within layouts 1 and 2, but struggles with the more advanced layouts. This is due to the collaborative nature required for the remaining layouts. "4. Forced Coordination" layout specifically requires the two agents take on distinct roles and neither agent can complete a soup on their own.

The Custom reward structure is designed mainly to facilitate this behavior. The agent on the left (green chef) needs to be incentivized to place onions and dishes (onion and dish drop in Table 1) along the middle counters in order for the agent on the right (blue chef) to be able to place the onions in the pot, pick up soups, and deliver soups.

The Custom reward structure alone introduces an unwanted behavior. The pickup and drop off rewards encourage an undesirable loop since the rewards are received for actions that theoretically negate one another.

This can be countered with adding a penalty for actions that negate the previous action. For example, if an agent drops an onion and in the very next timestep picks up the onion, then the previous rewards are negated/removed.

Figure 2 shows the rising trend in looping actions from the Custom reward shaping (blue line) when Negate Penalty is not also used. The orange line shows how the Negate Penalty offsets this incentive of infinite loops. This leads to more Starting Stove events in Figure 3 which will ultimately lead to a higher probability of delivering more soups.

Figure 2:
Negated Action Taken

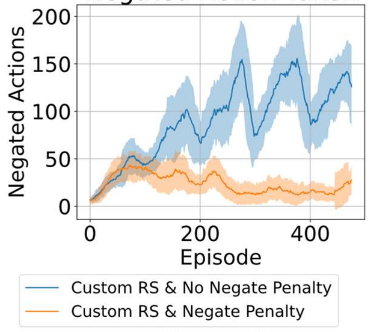
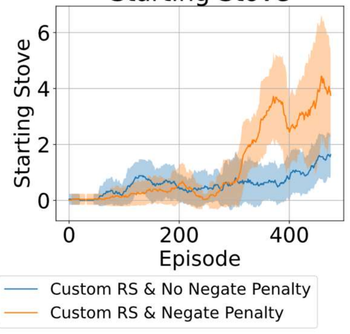
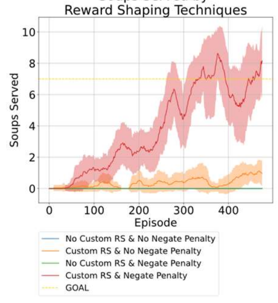


Figure 3:
Starting Stove



Keep in mind, Starting Stove is not the same as Starting Stove Early; the latter does not include the appropriate amount of ingredients. Starting Stove Early is a penalty introduced to discourage the agents from starting a stove before three onions are placed in the pot. This leads to faster learning as agents do not have to wait for the delivery of an incomplete soup to get no reward, instead they will receive immediate notification that this action is not helpful.

Figure 4:
Soups Served by
Reward Shaping Techniques



Results from the reward shaping techniques are shown in Figure 4. The graph shows no learning taking place when the Custom Reward Shaping is not used at all (blue and green lines). Nevertheless, only after Negated Penalty is implemented along with Custom Reward Shaping are the agents finally equipped with the proper incentives to learn subgoals (red line). As mentioned previously, without the Negate Penalty (orange line) the agents are prioritizing looping actions over the ultimate goal (making complete soups). The negate penalty is a form of potential-based shaping.

B. Single Agent (Baseline)

Once reward shaping is deemed to be optimal for learning the proper subgoals, the initial experiment is training a single agent to make as many soups as possible. One agent is equipped with DQN for learning, while the other is guided to a corner of the kitchen, out of the way, and instructed to remain there throughout the episode. The idea behind this approach is it demonstrates a baseline for what could be achieved in each layout. From there, the expectation is that more soups should be able to be produced with collaboration from another chef.

Additionally, this helps establish a good set of hyperparameters to use. For instance, a key parameter to note is epsilon decay rate as this plays the biggest role in solving Overcooked. The ultimate value used is 0.999975. This parameter is significantly higher than it is set to in Lunar Lander (0.99 in Project 2).

The rationale for this is that the agents have to explore much more. When agents are accounting for several subtasks, there are many combinations of how tasks/moves can playout and must be explored.

Figure 5:
Soups Served by Layouts

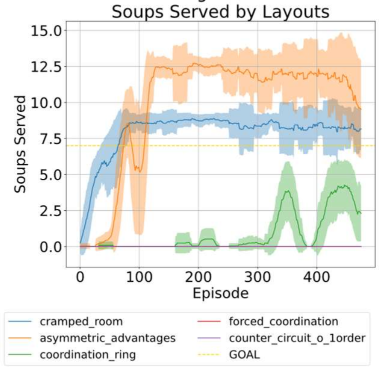


Figure 5 shows the results of training a single agent. Layouts 1 and 2 are achievable with a lone chef. This aligns with expectations as the blue chef has full access to all required tools (onions, dishes, stove, and delivery counter) in both layouts and the green chef simply just remains in its starting position.

There is no surprise layout 4 is impossible since the required tools are not all accessible by either agent. As for the remaining layouts (3 and 5), they both present too many obstacles, especially with an agent standing in the way.

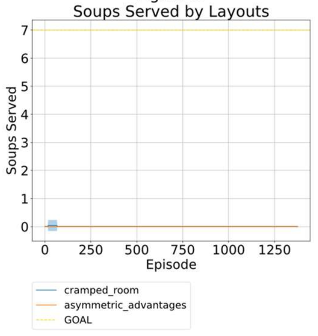
C. Decentralized Training (IDQN)

Decentralized Training, aka Independent DQN (IDQN), is when each agent is learning the environment without any knowledge of what actions the other agents are taking at any timestep. IDQN utilizes the DQN algorithm shown in the previous grey box, but with modifications to adapt for multiple agents. First, each agent's transactions are stored in their own replay buffers. Second, ε-greedy actions are selected independently of the other agent's action and then fed into the environment.

The environment that is deterministic in nature, now appears to be rather stochastic. For instance, if the blue chef drops an onion on the counter and the green chef starts the stove, then neither agent is expecting the environment to respond with an onion on the counter and the stove to be on.

Figure 6 depicts the difficulty this issue presents in training each agent. The plot shows some learning taking place because even producing a single soup in any given episode is statistically implausible using random actions. Maintaining learning is challenging given the stochastic tendencies introduced.

Figure 6:
Soups Served by Layouts



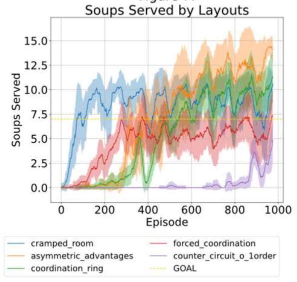
Allowing the training to go on longer may be enough to allow each agent to settle in on their own optimal actions, but given limited resources this approach is not investigated in more details at this time.

D. Centralized Training

Centralized training is a single DQN where states are provided as input to the DQN's neural network and the output is a combined action array (one for each agent). Rewards relayed back to each agent are a straightforward summation of rewards from the environment. The centralized learning allows for each agent to be aware of all actions being taken by the other agents so the environment holds its deterministic characteristic.

The learning to this approach can be seen in Figure 7. It is clear centralizing the information within the training fosters efficient learning to occur, especially when compared with decentralized training (Figure 6).

Figure 7:
Soups Served by Layouts



V. Trained Chefs

Since centralized training produces the optimal results out of the experiments it will be used to produce the policies for execution. Within the multi-agent realm there are centralized and decentralized techniques for execution, just as there is for training.

Figure 8:
Soups Served by Layouts

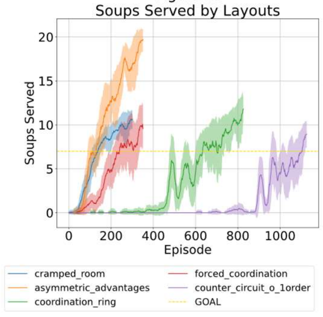
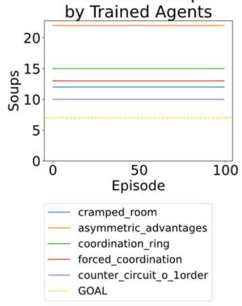


Figure 9:
Number of Soups
by Trained Agents



Centralized execution is when a single agent has control of multiple platforms. The agent provides instructions to the platforms and the platforms are deployed without any form of artificial intelligence. Decentralized execution is giving each agent the intelligence to make informed decisions about how to operate within the environment given the state they are in.

Figure 10

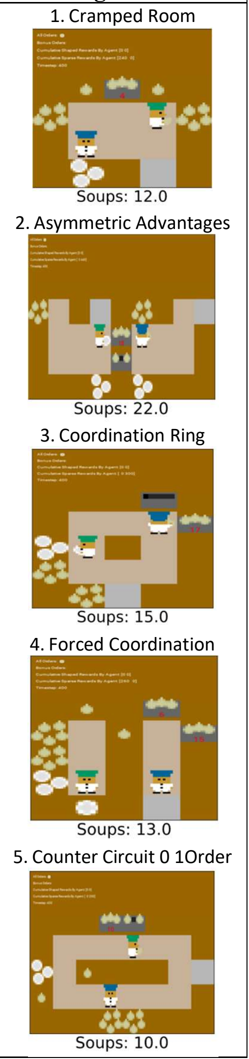


Figure 8 shows the training of the final agents. Given the difficulty of getting the algorithm to fully converge and limited training time resources, training is stopped when policies reach a predefined threshold for each layout. The convergence issues will be discussed in more detail in the Challenges and Future Exploration section.

Cramped Room and Asymmetric Advantages layout require the least amount of collaboration. This is due to each agent having access to all the required tools to completely create and deliver their own soups without having to navigate around the other chef as much as the other layouts. As a result, they are consistently one of the fastest layouts to train on (blue and orange line). On the other hand, Forced Coordination, Counter Circuit, and to some degree Coordination Ring take longer to train given the amount of collaboration that needs to be learned.

In this setting, both produce the same results since the states are fully observable, meaning agents are fully aware of the states of the other agents. As mentioned earlier, the state-space that the agents learn from (either centralized or decentralized) are aware of the position of the other agent. In other situations, decentralized execution may be the only viable option if technology restrictions prohibit communication between the agent and the platforms for utilizing centralized execution.

Figure 9 is produced from a decentralized approach so each agent is deployed into the environment with their own policy. The policy is generated from the centralized training so the policy takes into account other agents' optimal action, but the agents will not truly be aware if other agents are actually following their policies during execution. The agents' epsilon is set to zero so there is no randomness in the results. The plot shows agents successfully collaborating to master each of the layouts. Figure 10 provides a visual depiction of the final timestep for each layout for the trained agents.

VI. Challenges and Future Exploration

It is a challenge to select an algorithm. DQN has the advantage of being relatively straight-forward when modifying it to multi-agent. This allows the agents to quickly show positive results.

A downside to multi-agent DQN is that it quickly becomes impractical once more agents or actions are incorporated. The reason for this is that output of the neural network grows exponentially with the number of agents (n) and the size of the action space (a):

$$size\ of\ output = a^n$$

For this situation, the output is a reasonable size at 36 possibilities (6^2), but if just one more chef is introduced into the equation, then the output size increases to 216 (6^3).

From there, it is a challenge to get it to converge. This may have been due to the Exploration-Exploitation dilemma that plagues reinforcement learning. Several epsilon decay strategies were attempted with little impact. Also, DQN is sensitive to hyperparameter tuning, which may be the culprit. Since this report is less about hyperparameter tuning and focuses more on multi-agent techniques, it is not prioritized.

For future consideration, other options could be:

1. Multi-Agent Deep Deterministic Policy Gradient (MADDPG): The paper references continuous action space, but there have been modifications to this to adapt to discrete action spaces (e.g., Gumbel-Softmax) [1].
2. Counterfactual Multi-Agent (COMA): This approach also uses centralized learning and decentralized execution, but varies slightly from the approach described in this paper. COMA looks to enhance the credit assignment of action's reward by an individual agent by comparing the combined global reward if that action is replaced with a default action [2]. The inspiration is to gain a better sense of the impact (i.e., of how much impact that action contributes to the global reward).

VII. Conclusion

Through the experimentation with Overcooked, a base understanding is presented for how to adjust DQN to multi-agent settings. There are limitations to this approach, but it is demonstrated that it can establish cooperations amongst a number of agents and outperform a single agent.

Certainly, the list does not end with the additional options provided in the future exploration section. Reinforcement Learning in general is a growing field and multi-agent research has only scratched the surface of its potential.

References

[1] Abbeel, P., Harb, J., Lowe, R., Mordatch, I., Tamar, A., and Wu, Y. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". USA, 2017

[2] Afouras, T., Farquhar, G., Foerster, J., Nardelli, N., and Whiteson, S. "Counterfactual Multi-Agent Policy Gradients". Oxford, UK