78c8011da20acc250de1a9e75b10a2f0cdff8155

Project 3: Overcooked AI

Trung Pham
Tpham328@gatech.edu
78c8011da20acc250de1a9e75b10a2f0cdff8155 (Git hash)

***Abstract*—In this paper, we present and discuss the application of Value Decomposition Network (VDN) within the dual-agent in Overcooked environment (modeled after a popular game. The simulation involved two agents/chefs collaborate to maximize the number of soups cooked and delivered after a certain time with objective to enhance cooperative strategies between agents to optimizing soup delivery across five distinct kitchen layouts. With VDN, each agent was equipped with its own Q-network to facilitating structured learning and decision-making process.**

The results demonstrates that VDN using with reward shaping could effectively solve the problems presented in first 3 layouts, showing consistent and substantial improvement in the number of soups delivered. However, the poor performance in the more complex fourth and fifth layouts indicated the need for further improvement in the learning models. The study concludes that while VDN is an effective solution to develop cooperative behaviors in structured multi-agent settings, its application requiring higher level of agent collaboration. Further work could explore the integration of state stacking techniques to enhance agents' awareness leading to improved performance.

I. Introduction

Overcooked is a two agents simulated environment, where two agents are tasked with collaboratively preparing and delivering onion soups in a restaurant kitchen. The inherent challenge lies not only in the execution of cooking task but also in strategic collaboration to optimize the process within a number of timesteps. This offers unique challenges that test both individual and cooperative agent strategies.

The primary goal of this problem is to develop a robust multi-agent system capable of maximizing soup deliveries within 400 timesteps per episode, with each successful delivery yields a reward of 20 points. The task is complicated by the need of precise coordination to manage resources efficiently. Some kitchen layout required collaboration between two agents like one can only cook and one can only prepare onion. Furthermore, the project stipulates the use of a single algorithmic approach and an uniform set of hyperparameters across five layout, emphasizing the need of an adaptable solution.

In response to this challenge, we employs Value-Decomposition Networks (VDN), utilizing separate but interconnected networks for each agent to facilitate learning through shared experiences and rewards. This approach also leverage a joint replay buffer as well as incorporating strategy like reward shaping, mirrored buffer, and Q-target network.

The paper also discussed two metrics collected by the environment and potentially its usage to improve the learning model as well as other improvement for future research like state stacking.

II. Background

In Overcooked environment, both agents have access to full observations. Each observation are provided as 96-element vector corresponding to different feature (position, distance to ingredients, etc.) The action space is discrete with six possible actions: up, down, left, right, stay and interact. Each layout is unique have wall and certain constraints, with fixed position of ingredients. It takes 20 timesteps to cook a soup and the rewards to successfully delivered a soup is 20. Each episode is 400 timesteps and the goal is to achieve consistently 7 soup delivered per episode.

III. Methodology

Based on suggestion of the problem, the model used in project 2 was implemented on Overcooked environment. The learner used in project 2 is a Dual Deep Q-Learning Network (DDQN). In this model, a separate Q-target network is maintained and used to evaluate the Q value target, and therefore is updated at a slower pace after certain interval. Using DDQN for Overcooked required intensive hyperparameter tuning and its only solved the first two layouts. It also displayed unstableness and high variance results (oscillating result).

For this reason, we opted to implement Value-Decomposition Networks (VDN), an approach in the realm of multi-agent reinforcement learning. VDN is predicated on the principle that joint action-value function, representing the combined actions of all agents can be decomposed into the sum of individual action-values for each agent. This decomposition allow for scalable and efficient learning by simplifying the typically complex multi-agent learning process, where action space exponentially increases the number of agents and possible actions.

Each agent's network in the VDN setup consists of multiple layers. Same way as set up in DDQN:

Input layer: accepts a stacked state representation, incorporating multiple past observations to capture temporal dependencies and environmental dynamics.

Hidden layers: comprises several dense layers that allow the network to learn complex patterns in the data.

Output layer: produces a vector of action values corresponding to each possible action the agent can take at given state.

Algorithm 21 Value decomposition networks (VDN)

Initialize n utility networks with random parameters $\theta_1, \ldots, \theta_n$
Initialize n target networks with parameters $\bar{\theta}_1 = \theta_1, \ldots, \bar{\theta}_n = \theta_n$
Initialize a shared replay buffer D
for time step $t = 0, 1, 2, \ldots$ **do**
  Collect current observations $o_1^t, \ldots, o_n^t$
  for agent $i = 1, \ldots, n$ **do**
    With probability ϵ: choose random action a_i^t
    Otherwise: choose $a_i^t \in \arg\max_{a_i} Q(h_i^t, a_i; \theta_i)$
  Apply actions; collect shared reward r^t and next observations $o_1^{t+1}, \ldots, o_n^{t+1}$
  Store transition (h^t, a^t, r^t, h^{t+1}) in shared replay buffer D
  Sample mini-batch of B transitions (h^k, a^k, r^k, h^{k+1}) from D
  if s^{k+1} is terminal **then**
    Targets $y^k \leftarrow r^k$
  else
    Targets $y^k \leftarrow r^k + \gamma \sum_{i \in I} \max_{a_i' \in A_i} Q(h_i^{k+1}, a_i'; \bar{\theta}_i)$
  Loss $\mathcal{L}(\theta) \leftarrow \frac{1}{B} \sum_{k=1}^{B} \left(y^k - \sum_{i \in I} Q(h_i^k, a_i^k; \theta_i) \right)^2$
  Update parameters θ by minimizing the loss $\mathcal{L}(\theta)$
  In a set interval, update target network parameters $\bar{\theta}_i$ for each agent i

Figure 1: VDN Algorithm

We follow the algorithm in figure 1 for the VDN learner.

Each agents have their own policy network. They shared a replay buffer that contains the following fields: (agent0_state, agent1_state, agent0_action, agent1_action, combined reward, agent0_next_state, agent1_next_state, terminal)

Hyperparameters, including learning rate, discount factor and the size of replay buffer, etc. are tuned based on preliminary experiments to balance between exploration and exploitation ensuring robust learning across all kitchen layouts without overfitting. Here we chose gamma = 0.9, learning rate = 0.0003, buffer have size of 100,000 records, sample mini batch size = 1,024. We also have epsilon decay rate of 0.00001 until epsilon reaches 0.15 or 15% probability of the time, the agent explore instead of prioritizing rewards.

After intense test, we realize epsilon is very important with Overcooked learners, because both agents need to explore enough to understand collaboration and its relation with reward.

We also implement reward shaping strategy with key task that lead to the successful delivery of soup like placing onion in the pot, pick up dish, pick up soup, we gave a smaller reward of 1, 3 and 5 respectively. The shaped rewards for each agent is added together with the shared reward from delivered soup to become a combine reward that is stored in the shared buffer. Reward shaping is among the biggest improvement on our model because it guided the agent toward common goal.

In our project 2, we used hard update on the deep Q network, this is proven not appropriate in Overcooked environment. We opted to use soft update of the Q-network for both agent with tau equal 0.001. This improved our model moderately.

Last, at the start of an episode, each agent is placed randomly between 2 starting position, due to the fact that they can learn from each other, we implement a mirrored experience in shared replay buffer. Because both agent can perform same task/actions, they can shared the experiences. This double our experience and improve our training speed significantly. It made our model converge to a solution faster and more consistently. It also solved the problem that an agent is randomly placed in a non-preferred starting position for him.

The VDN algorithm is trained with 1,000 episodes for layout 1 and 2; 1,500 episodes for layout 3. It is trained with over 3,000 for layout 4 and 5 but still not converged.

IV. Result of Lunar Lander solution using DDQN

Our model successfully converge to solution for layout 1, 2 and 3. Layout 4 and 5 required intensively training of over 3,000 episodes and no convergence is found as well as consistency achieved.

Both layout 1 and 2 are solved by the learner around 500 to 1,000 training episode. Layout 3 are sovled around 1,400 episodes. Layout 4 and 5 showed no converged solution so far after 3,000 training episodes.

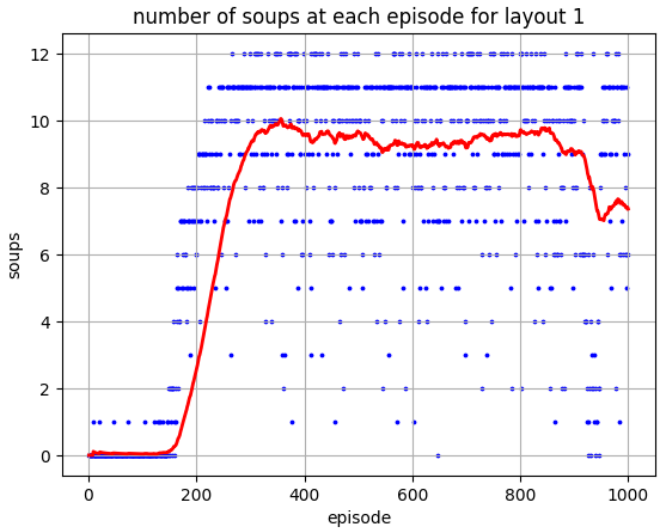


Figure 2: layout 1 training result.

Layout 1 passed the goal very quickly around 300 training episodes, the performance improvement slow down and start going down, probably overfitting.

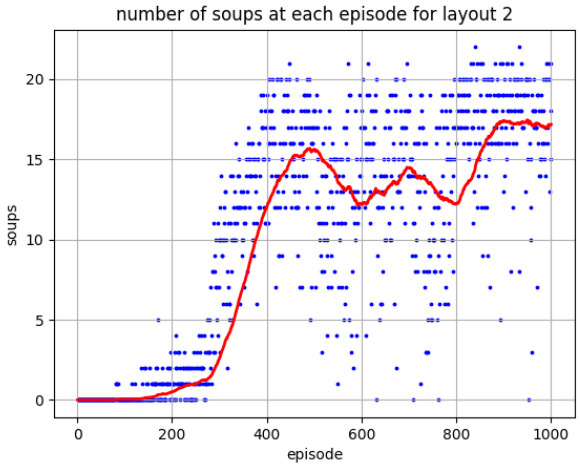


Figure3: Layout 2 result

For layout 2, the model start converging to solution and achieve the goal of averaging 7 soups after 350 episodes, fastest among the layouts. The model consistency continued to rise given more training episodes. This makes sense because in layout 2, both agents have access to everything and not require much if any collaboration.

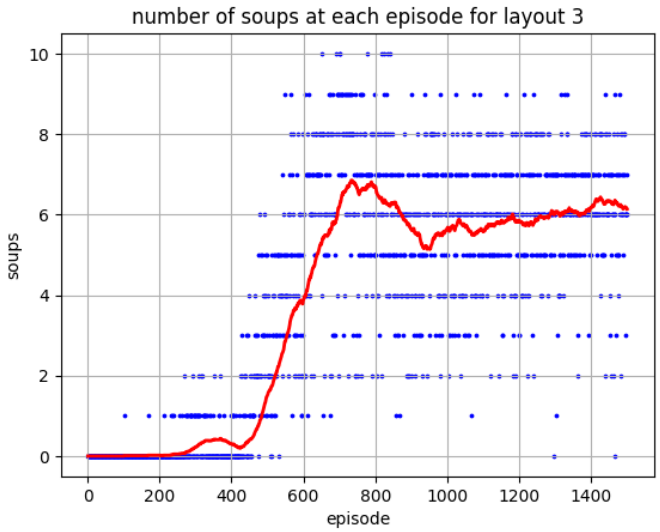


Figure 4: Layout 3 result

Layout 3 is a more complex kitchen layout with collaboration require between two agents so they do not block each other. Solving layout 3 require more training episodes, instead the model is still improving if we can train for more episodes. Layout 3 shows very similar pattern like layout 1 and 2 in term of model performance. Number of soups delivered increase quickly the slow down. This showed that once the agent explored a way to make soup and increase reward, they start making soups consistently after.

There are no graph for layout 4 and 5 because the agents in our model have not find a way to make soup yet. We could train the model for more but training layout 4 and 5 took really long time, the decay rates need to be slower for layout 4 and 5. However, doing so will make solution for layout 1, 2 and 3 less stable because the agents will explore more instead of prioritizing rewards.

We played 100 games with each trained agent and the results are good. This reflected that overfitting is less of an issue for layout 1-3.

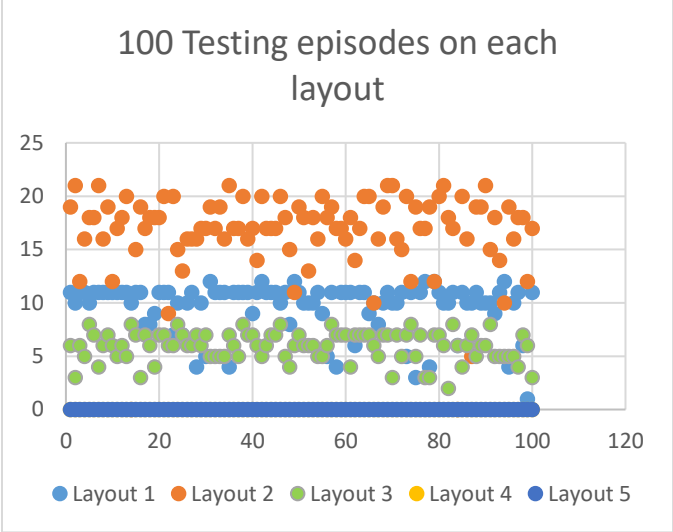


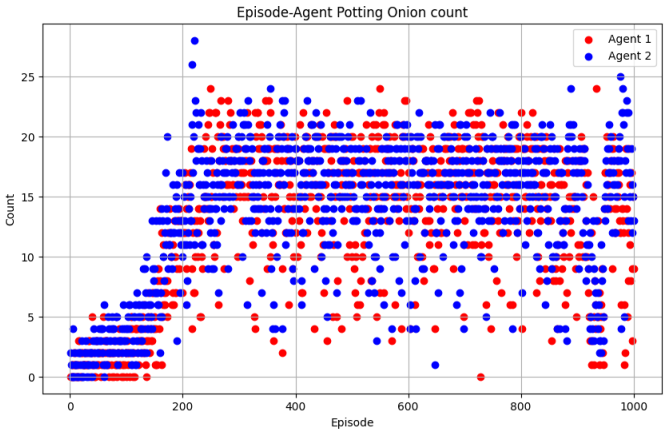
Figure 5: 100 testing episode results.

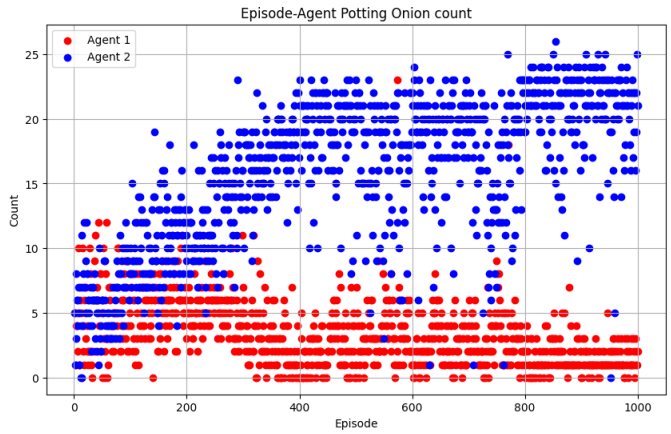
As we can see, VDN with reward shaping is very effective to solve Overcooked AI problem. It require a lot of computational power but the result is very good if we have enough time to train the model.

V. METRICS

In addition, we studied two metrics returned from the environment to help improve our learning model. They are the numbers of onion being put into pot, and the number of dish being pick up. Put onion and pick up dish are also two of the reward shaping that we used to guide the agent toward the common goal.

Over the course of training, we stored the onion numbers being put in each episodes from each layout.





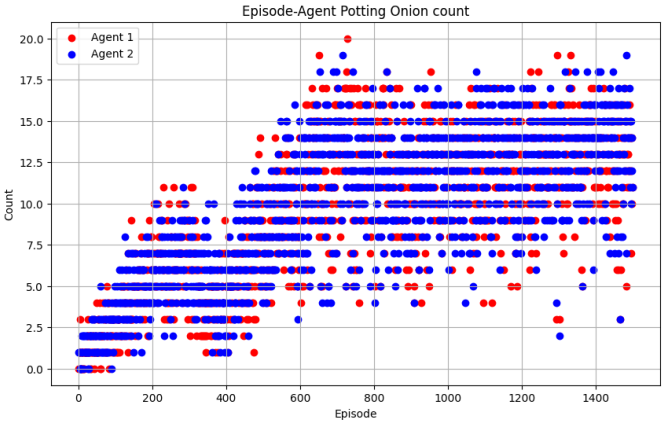
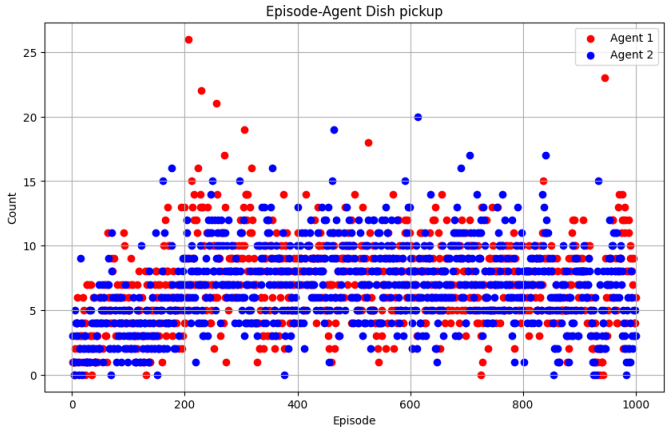


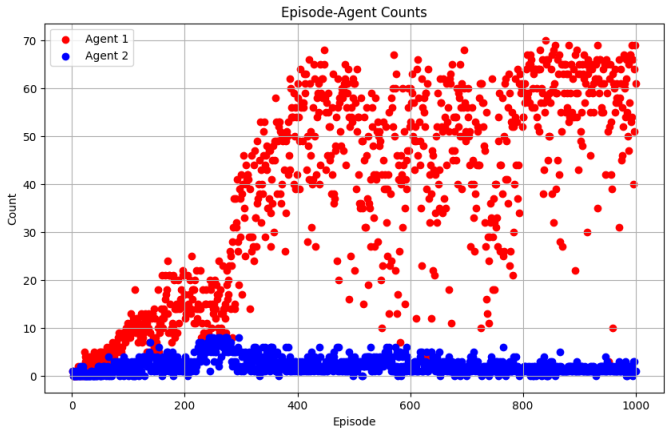
Figure 6: Potting onion of layout 1-3

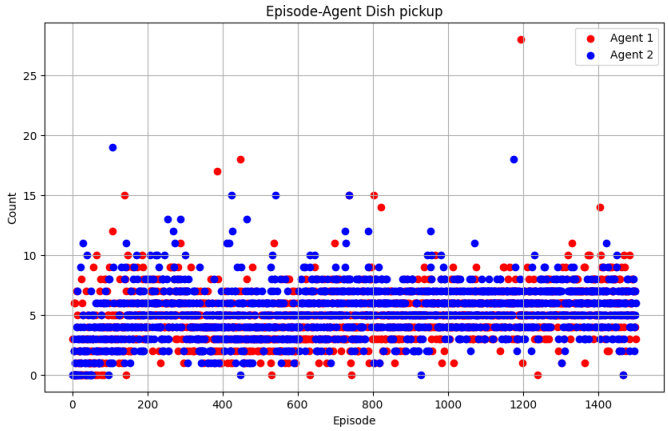
We can see that both agents are consistently picking up onion in layout 1 and 3. Layout 2 looks like agent 1 did not put onion as much as agent 2. Since putting onion yield a reward of 1. There is an incentive to put onion into pot. Over the course of training episodes, the numbers of onion being put increases, showing that the agent does learning to be incentive to put onion. This is the reason we used putting onion as part of the reward shaping, since the agents need to put onion before they can cook soup.

A problem with reward shaping is reward hacking, when the agent instead of making soup or fulfill the goal just put onion to get small immediate reward. This is a problem that can be fixed by decreasing the shaping reward for putting onion after a fixed interval or gradually decrease after the agent successfully learned this step or achieve a number of onion being put.

Another metric we are interested in is the disk pick up. The agent needs to pick up the disk before serving soup. Because this is subsequent step after put onion, we give them incentive reward of 3.







Similar to previous metric, one of agents in layout 2 do not performed well. We can clearly see that in opposite to put onion, agent 1 now pick up disk more often, it seem each agent in layout 2 have found a way to prioritize their tasked, one agent put onion and one agent pick up disk. This is actually good because they have shorter distance to travel and therefore collaborate in term of helping each other increase productivity. As a result, the average soup delivered in layout 2 is higher than both layout 1 and 3.

Same as putting onion, once the agent learned of the step to make soup, we can decrease the incentive of doing that task.

VI. Drawback and improvement

Our model could not solve the layout 4 and 5 consistently. Upon testing, we realized that the model did not explore enough and leading to unlearn when the buffer is full and being replaced with bad experience before the agent can explore or aware of all the steps leading to the goal. A solution to this is having a higher epsilon or lower the decay rate and train the model more at the cost of computation. This is however not good for layout 1-3 where the kitchen map is simpler. No solution is actually good for all.

In addition, we also look into an improvement technique called state stacking, which involve stacking multiple consecutive observation together called a stack. A stack contains information of history and future so the agent can be aware of task and steps. A single observation might not provide complete information about the environment dynamics.

State stacking helps enhance temporal awareness, especially in Overcooked environment, where past actions or events significantly influence future states. By observing multiple frames together, the agent can infer of directions and trend that cannot be seen from single frame alone. The goal of delivering soup is a sequence of tasks, therefore, a sequence of observation is necessary for the agent to understand the procedure.

Given more time, we would definitely came up with slower epsilon decay function and apply state stacking to our model.

VII. Conclusion

The application of Value-Decomposition Networks (VDN) in the multi-agent Overcooked environment has yielded a promising result, showing how powerful VDN can facilitating cooperative behavior among agents in complex task settings. This study applied VDN to train two agents tasked with collaboratively cooking and delivering soup across five different layout that yield significant result. The methodology incorporated advanced techniques such as reward shaping, mirror experience, and soft update Q-target networks, which collectively enhanced the learning efficiency and policy stability.

Despite these improvement, the VDN model achieved varying degree of success across different layout. While it convincingly solved the challenge posed in first three layouts, last two layouts required a higher degree of collaboration and coordination, which proved to be beyond the current capabilities of our implemented model. Although, the reward shaping and mirrored experience techniques implemented were effective to an extent, they were insufficient for overcoming the most challenging scenarios presented by layout 5.

Looking forward, the implementation of state stacking is anticipated to improve the agents' ability to make informed decision based on the progression of the environment rather than single snapshots.

Finally, this study highlights both the potential and challenges of applying VDN in a structured multi-agent environment. The success and limitations in this experiment emphasizes the need for continuous improvement in multi-agent learning methodologies. Future work will focus on refining these techniques and exploring new strategies to increase robustness and adaptability, help the agent optimize their strategies across broader range of collaborative tasks.

References

[1] Value-Decomposition Networks for Cooperative Multi-agent learning, Peter Sunehag, Guy Leve, Audrunas Gruslys, Wojcjech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Leibo, Karl Tuyls, June 2017.

[2] Multi-Agent Reinforcement Learning. Foundations and Modern Approaches. Stefano Albrecht, Filippos Christianos, Lukas Schafer. MIT Press 2024.