Pacman Approx. Q Learning and Reflex Agents for Aritificial Intelligence COSE 361

Hyunjin Lee 2020320023 [1]

Abstract

The final project, Pacman Capture the Flag Contest, requires a sophisticated agent in order to achieve a high win rate and beat the baseline agents. The agent needs to be able to perceive its environment and act in a way that maximizes the reward. To achieve this result, I have implemented an approximation Q learning agent for offense and a strategic reflex agent for defense. The code explanation is commented in the python files.

1. Introduction

COSE361 final project is a multi-player capture the flag variant of Pacman, where agents control both Pacman and ghosts. The contest is an adversarial game, which one team is trying to maximize the food from the other side of the map and minimize the food stolen from the home side.

One important aspect of the contest is to have a good balance between offense and defense. Eating foods from enemy territory is just as important as defending the food from home territory. Thus, I started with an offensive reflex agent that paths toward the closest food and capsule and a defensive agent that paths toward an invader if seen. The result was not great especially on the offensive side. The defensive reflex agent did not have a trouble defending the food from home territory. However, the offensive reflex agent had a hard time escaping the enemy while eating food. So, for my second implementation, I have created a offensive approximation Q-Learning agent while keeping the defensive reflex agent the same. The approximation Q-Learning agent had 3 features: bias, distance toward capsule/food, and distance toward the invader. For the third implementation, I have improved the second agents by introducing strategic aspects. I have also introduced a "dead end" feature, which allows the agent to decide whether to enter a dead end based on its calculation. Finally, for the last agent, I have created 2 approximation Q-Learning offensive agents.

[*]Equal contribution [1]Computer Science, Korea University, Korea. Correspondence to: Hyunjin Lee 2020320023 <hyunjin1109@korea.ac.kr>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

2. Methods

Four different agents were implemented for the final project.

2.1. Reflex Offensive Defensive Agent

The idea behind reflex offensive defensive agent was simply to see how well it performs under an adversarial game. The reflex offensive agent would simply take the action that would bring the agent closest to its current target (food or capsule). The defensive agent would first move toward the "entrances" (open area where the opponent can invade), then take the action that would bring the agent to the invader if located. The offensive agent performed poorly because it did not consider the opponent's movement. However, the defensive agent did surprisingly well and defended well against the baseline agent provided.

2.2. Approximation Q-Learning Offensive and Reflex Defensive Agent

Because the reflex defensive agent performed well from the first implementation, I have focused on improving the offensive agent. The reflex offensive agent was not able to perform well because it did not consider the enemy's movement. In order to consider enemy's movement, I have implemented an approximation Q-Learning agent. The agent has a bias feature, target distance feature, enemy distance feature. The target distance is the distance between the agent's position and its target's position. The target can be capsule, food (prioritizing capsule over food) or entrance to home territory. The agent will set the target to home entrance once it has eaten all the food, once it has eaten certain number of food while winning, or is running out of time. The agent receives a negative living reward and receives a positive reward if it has reached the target and eaten a food or capsule, or has brought food back to home territory. The weights were learned with learning rate 0.1 and discount rate 0.8. The agent performed relatively well compared to the first agent. It was able to eat food while escaping from

the enemies. However, the agent had a hard time in a layout with many dead ends. Once the agent entered a dead end attempting to eat food, the agent was unable to leave the dead end because it was trying to maximize the distance between the incoming ghost and itself.

2.3. Strategic Approximation Q-Learning Agent

The best result was from introducing some strategies along with the original Q-Learning offensive and reflex defensive agent. Before the game starts, I made the agent calculate all the dead ends region along with the depth of the dead ends using dead end filling algorithm. This allows me to add another feature into the Q-Learning agent: ”dead end” feature. With this feature, the agent can decide whether to enter a dead end region based on the distance between itself and the ghost. If the agent was far enough or the agent has eaten a capsule, it would enter the dead end region without any hesitation. However, if the agent is currently being chased, and the dead end's depth is longer than the distance between the agent and the ghost, it would not enter the dead end region.

I have also noticed the offensive agent often fails to penetrate into enemy's opponent because of its attempt to get toward the target (food or capsule) in a shortest path. In order to fix such problem, I have implemented an optimal path search algorithm, such as BFS. When an agent has hard time penetrating, or is making repetitive moves, the BFS would return an optimal safe path to entrance that is farthest from its current position. Because the enemy agent cannot react fast enough, openings are created for the offensive agent to enter.

2.4. Two Q-Learning Offensive Agent

The idea behind two offensive agent is simply eat all of enemy's food before enemy finishes eating ours. Because the game ends once a side eats all of the opponent's food, the two attacker paradigm is theoretically possible. For the first offensive agent, I prioritized getting the food; for the second agent, I prioritized getting the capsules. This would allow the first agent to finish eating most of the food while the second agent activates the capsules. Unfortunately, with my Q-Learning agent, was simply not able to eat the food faster than the enemy agent while also escaping from the opponent ghost. In order for such implementation to work, I would need the two agent to communicate with each other and eat the food in the most efficient way. For example, one agent may act as a bait while the other agent eats all the food hidden inside dead ends.

3. Results

	A	B
1		your_best(red)
2		<Average Winning Rate>
3	your_base1	0.8
4	your_base2	1
5	your_base3	1
6	baseline	0.9
7	Num_Win	4
8	Avg_Winning_Rate	0.925
9		<Average Scores>
10	your_base1	4.3
11	your_base2	10.7
12	your_base3	18
13	baesline	8.5
14	Avg_Score	10.375

4. Conclusion

Question: Is it better to have many complex features to evaluate observed state/action (ex. using a neural network)? What is the optimal number of features?

Answer: Although it depends on the types of problem and number of samples, generally, I believe having too many of complex features many result in over fitting problem. However, with few numbers of features, the agent cannot distinguish between states and make the ”optimal” moves. For our given adversarial problem, around four features seemed to be enough for making relatively optimal moves in every state. Also, according to Google's DeepMind, 4 hidden layer will usually suffice for evaluating state and action.

Discussion: Although my agent performed relatively well compared to the baseline agent, many improvements could have been made. For example, implementing communication between the offensive and defensive agent could have resulted in performance improvements: if losing, the defensive agent can start attacking with the offensive agent, if winning, the offensive agent can come back and start defending, etc. Also, I have realized there were many dead ends in many of the layouts. I can also make the defensive agent clog the opponent inside a dead end while coordinating with the offensive agent to come back if necessary. Next time, I would like to try to implement neural network, such as DQN, for evaluating state and action instead of simple linear approximation.