Final Project: Multi-agent Reinforcement Learning

Hoan Duc Tran
UB CSE
hdtran@buffalo.edu

Abstract

Here I present a multi agent environment, that is compatible with OpenAI GYM. In this environment, we have multiple agents trying to reach their respective destination in the shortest time possible, while avoiding collision with each other. The agent can see other agents, and their destination. The environment is solved using Q-Learning and Deep Q-Learning

1 Environment

The environment is a simple empty grid world, with agents starting at the bottom edge of the grid, spaced an equal distance apart. Their respective destinations are on the top edge. Think of the destination placement as simply moving all agents to the top, then swap the destinations by flipping over the y-axis. A simple world that we are using with a 5x5 grid and 3 agents is demonstrated in Fig .1.

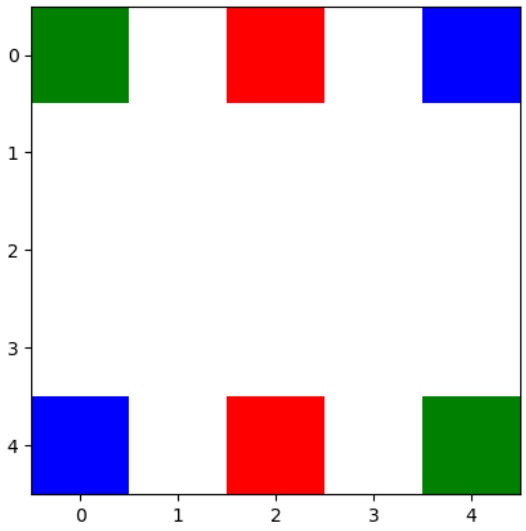


Figure 1: The environment with size = 5x5, and 3 agents

Obviously, this is not a particularly hard environment. But for the sake of demonstration, and due to limited hardware, this environment would suffice for the example.

1.1 Action space

Each agent has 5 moves: No-op, Up, Down, Left, or Right. If the agent has reached their destination, the move are not registered, meaning any move is a No-op move. The agent cannot move through each other, or out of bound. They can move through other agent's marked destination, but they cannot do so if another agent is there.

When a move is illegal, either out of bound or somehow caused agents to occupy the same spot (crashing), then the agents who makes the illegal move (multiple agents in case crashing occur) will be reset to their original position (somewhere in the bottom of the grid) and received a negative reward.

The game will end only if all agent have reached their destination.

1.2 Observation space and Rewards

The agent can see itself, the destination and the location of all other agents. The agent do not share observations, meaning that agents do not know of other's destination. An example of the observation space is given in Figure. 2.

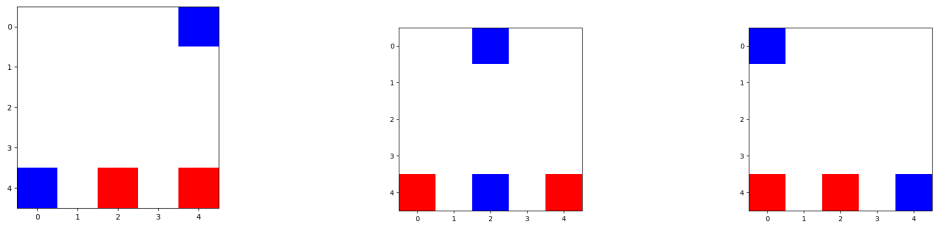


Figure 2: The observation space of each agent, following the setting in Figure.1. The agent and the destination is marked in blue, while other agents is in red.

At each step, all agents receive a -1 reward, unless reaching the destination. When the agent makes any illegal move (out of bound or crashing), the agent will be reset back to the original starting point, and takes a -5 reward.

2 Q-Learning

Q-Learning is a model-free tabular method. It learns the state-action value function, and thus deriving the optimal policy $\pi(s) = argmax_a Q(s, a)$. The problem with Q-learning in most cases is the exploration-exploitation trade-off. In this project, I have managed to find a nice balance that leads to an easy solution for this environment.

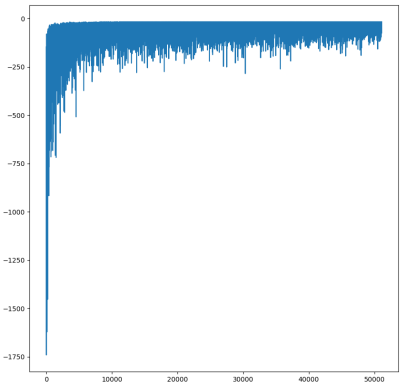
2.1 Training parameters

I chose simple Q-learning as the tabular method of choice to solve this maze. Each agent makes their move independently and at the same time. For learning and testing, I set the exploration epsilon to constant 0.1, and trained for 100000 steps (1 million). I limit the number of step per episode to 500.

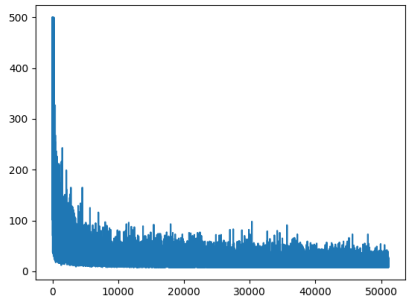
2.2 Result

The training is done in about 1 or 2 minutes, which is relatively fast. According to Figure. 3, the agents learn quite quickly, needing only a few epochs before figuring out the strategy. Due to the high epsilon, sometime the agent might crash or move out of bound. It is possible that reducing the epsilon for prolonged training might yield better result.

All agents successfully learn to reach their reward. Their ideal strategy is to prioritize the middle guy, while the other two guys wait.



(a) Cumulative reward across episode



(b) Number of step all agents spend across episode

Figure 3: The training result for simple Q-Learning.

3 DQN

Since Q-Learning works so well on this simple maze, I believed that DQN would give similar or even better result. DQN biggest issue is the need for an experience replay buffer, which allows to train the parameterization (often Deep Neural Network) with independent and uncorrelated data, which is a big requirement for a non-bias DL model. Another issue is the computation time, especially for big model.

In this case, I settle for a particularly small model, but demonstrate that it is robust enough to handle a variety of situation, from navigating changing environment as well as navigating where there is little room for maneuver.

3.1 Training parameters

I trained the agent on 20000 (20 thousand) steps, and let the world end in at most 500 steps. The replay buffer size is set to 5000. Every 1000 steps, the target model is updated. The epsilon started at 1, and is reduced linearly for the first 100000 steps until it reaches 0.01.

3.2 Model architecture

A simple FCN model, with two hidden layers with size 128 each. At each layer, except for the output, there is a Rectified Linear Unit activation function.

3.3 Results

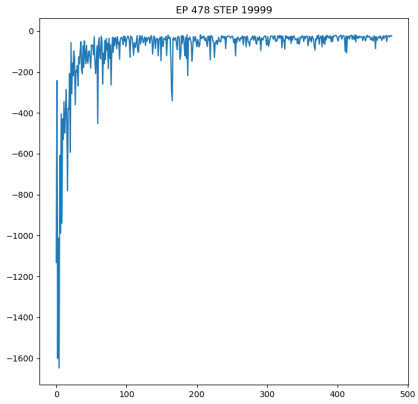
The training takes considerably longer time, with 30 minutes. But overall, the result is similar to what we obtained in Q-Learning. The result is in Figure .4

As you can see, the result looks slightly better with less variance. I believe that this might be partly due to the lower epsilon toward the later half of the training.

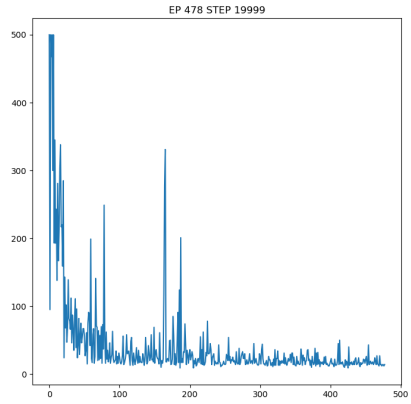
3.4 Adding the flavor: Introducing obstacle to the grid

To make it more interesting, I added obstacle to the grid. The obstacle will be somewhere in the middle row of the grid, and I ensured that there will be a valid path for all agents to reach their destination. Some of the new environment setting can be found in Figure .5

Notice that, I have reduced the number of agents, for better training time.



(a) Cumulative reward across episode



(b) Number of step all agents spend across episode

Figure 4: The training result for DQN.

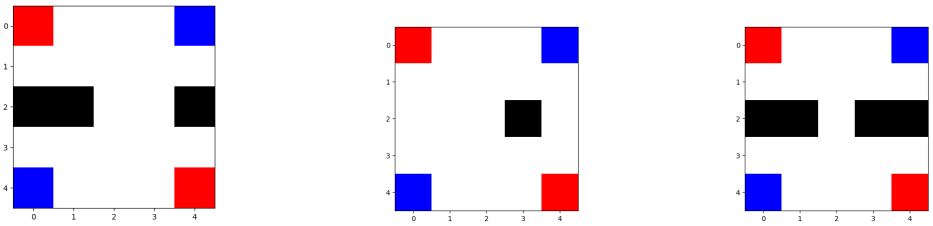


Figure 5: The environment, with some obstacle

My question is, can the agents learn to navigate on any randomly generated obstacle?

3.4.1 Training and Result

For training, I cranked the number of training steps to 200000, and letting the epsilon decay over 150000 steps. The experience replay size is also increased to contain the last 20000 transitions. To make life easier on a low-end laptop, I lower the number of steps before done with an episode to 200. Every 10 episodes, I changed the obstacle. The agents do learn to navigate the environment, but every time new obstacle is introduced, there is a steep spike, as evident in Figure .6.

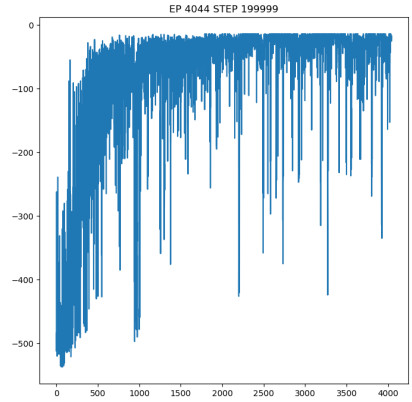
But although there is a spike, overall, the agent do learn to eventually deal with the environment, and move toward success.

3.5 Adding the topping: Adding more agents to the grid

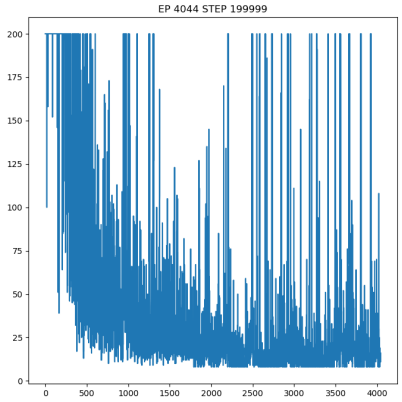
What if there are 4 agents (the max number of agents) in a 4x4 grid? Surely, the place would be more cramped, and harder to navigate (Figure .7. I again used the same model on this environment.

3.5.1 Training and Result

Since the environment size is smaller, I reduced the number of training steps to 100000, and let the episode ends after 100 steps. Other parameter is similar to the previous section, and of course, the same model architecture. From the results, report in Figure .8, the agents seem to be able to learn, but the number of training steps is not enough. Potentially, after further training the agents might learn to navigate the maze.



(a) Cumulative reward across episode



(b) Number of step all agents spend across episode

Figure 6: The training result for the obstacle environment, with DQN.

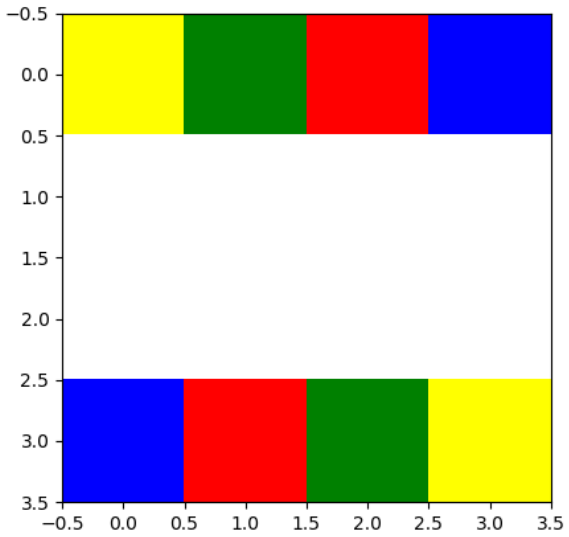
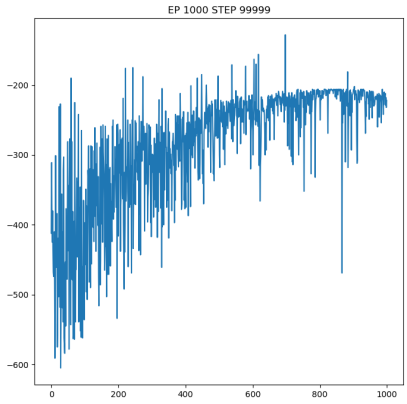
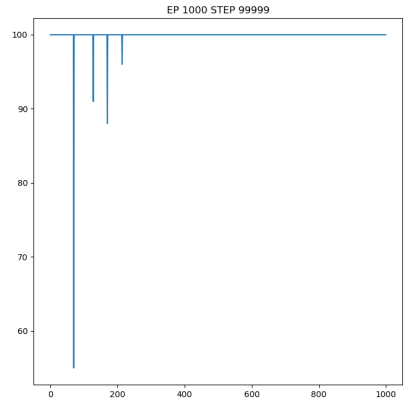


Figure 7: The environment with size = 4x4 with 4 agents. Surely this would limit the space to move



(a) Cumulative reward across episode



(b) Number of step all agents spend across episode

Figure 8: The training result for the many agents environment, with DQN.