Arnold Cheung

CID: 01184493

Reinforcement Learning Part 2: Coursework Part 2

The agent has been trained by implementing both basic and advanced Q-Learning techniques described in this document. Line 24 - 126 implements the underlying Q-network of the agent. Here the main Q-network, target network, and loss computing function is implemented. The full Bellmen equation with $\gamma = 0.9$ and the double Q-learning technique is implemented.

Line 129 to 168 is the implementation of the experience replay buffer, it has been initialised with a capacity of 1000 transitions with mini batch size of 300 in order to train on fairly recent transitions. It will only begin to sample and train the Q-network when agent has experienced at least 300 steps in the environment and that the agent has seen the goal. The sampling of the experience replay buffer is prioritised with $\alpha = 1$. The weighting and probability update of the transitions are executed by the update_weight() and update_p() functions between line 160 - 166.

From line 171 on is the main implementation of the agent class, all hyper parameters is defined in the constructor of the class. The episode length has been set to 100000 to allow the agent to freely explore the environment. Although attempted the continuous action Q-network has not been implemented due to its heavy computation, it slows down the exploration and training session too much and is therefore ineffective for a time constrained problem. To balance between continuous and discrete actions, a total of eight actions, along the x and y axis and the diagonals, have been implemented, each with step size 0.02 along the chosen direction.

The get_next_action() function has been modified for the agent to explore using the greedy epsilon algorithm, with initial $\epsilon = 0.875$, and $\delta = 0.0001$. However ϵ will only decrease beyond 0.7 if the agent has seen the goal during exploration. In the same function, the target network is updated every 100 steps.

Multiple reward functions have been tried out in order to find the optimal one, the one implemented in the end is get_reward_1(), in which the the function discretise the environment and provide increasing steps of reward as the agent gets closer to the goal. The reward function also heavily penalise standing still or transitions that result in increasing distance from the goal, so the agent will learn to avoid walls or turn back turning training.

The has_finished_episode() function has also been modified to terminate the episode under different circumstances, including the agent being stuck at the wall or bouncing between two adjacent states, and the agent reaching the goal; if the agent is stuck and $\epsilon < 0.15$ then ϵ is set to 0.15. In the same function, if the agent has reached the goal twice consecutively when $\epsilon = 0$, then the Q-network is copied to best_network and the training terminates.