AI-Chinese Chess

Haotian wang
hwang2@oxy.edu
Occidental College

1 Intro

This paper is a tutorial report base on a online tutorial about implementing reinforcement learning algorithm 'Q-Leaning' algorithm on a Pacman game[1]. I will be following these tutorial and discuss the method i used and how I evaluated the project. This tutorial has already finished the the majority of the work including environment, and the game. The main work that i will be working on the Value iteration agents, Q-leaning Agents and the analysis file which answers the question in the tutorial.

2 Method

In this main section, I will walk through the main methods I used for the two provided python files; ValueIterationAgents.py; QLearningAgent.py.

2.1 Value Iteration Agents

In this section I will explain how the Value Iteration agents works including the Markov Decision process.

The Markov Decision process is a 5-tuple describing an environment consisting state, actions, transition function, reward and a discount factor. An MDP provides a mathematical framework for modeling decision-making situations where the outcomes are partly under the control of the decision maker.

- States: A state, s is any predefined momentary instance in the world that an agent can exist in.
- Actions: An action, a is an event facilitated by the agent that can transition the value from one state to another provided that such a transition is possible MDP.
- Transition Function: The transition function is a function that defines the probability of moving to a specific next state, given the current state and a valid action.
- Reward: The reward function specifies a real number value that defines the efficacy or a measure of "goodness" for being in a state, taking an action and landing in the next state.
- Discount Factor: The discount factor essentially determines how much the reinforcement learning agents cares about rewards in the distant future relative to those in the immediate future.

This section takes a Markov decision process on initialization and runs value iteration for a given number of iterations using the supplied discount factor. The agent should take an MDP(Markov decision process) on construction and run the indicated number of iterations and then act according to the resulting policy.

I used a for loop to set up for self.iterations to begin the iteration. A temporary counter is initialized to store an iteration's value for each state. When the loop is looking at each state, the loop will look over the iteration. If the state is terminal, the reward is the exit reward and no discounted rewards as it is the absorbing state. If the state is non-terminal, then finding the best value as the maximum of expected sum of rewards of different actions.

After the value of the state is returned, i need to compute the Q-value of action in state from the value function stored in self.values. I first computed the transitions states and probability, then for each transition, i calculated the value as the sum of the reward of getting to that transition and discounted value of transition state. In this loop the transition value will be summed and it will give the Q-value for a state action pair.

The last part will be computing the action from values. A State Action counter is used to hold the Q-value for each state action pair from the previous method. The policy or action is the one that gives the best expected sum of rewards. Finally, the iteration returns the policy at the state and the Q-value for the next iteration.

2.2 Q-leaning Agents

In the previous section, i have explained how the value iteration agent works. However, the vale iteration agent does not actually learn from experience, Rather, it mulls over its MDP model to develop a comprehensive policy before ever engaging with a real-world setting. When it does interact with the environment, it simply follows the pre-computed policy. This discrepancy may be trivial in a simulated environment like a Gridword, but it's highly crucial in the actual world, when the genuine MDP is not available.

As always, we need to initialize our Q-value and contract a counter first. We now need to compute a value from the Q-value, which is zero, because no state is input yet. The value is calculated by returning the maximum action, where the max is over legal actions. Next, The best action from is calculated from Q-values. The action will be take into the current state with probability self.epsilon. We we now chose the most recent action for an epsilon greedy approad and choose a random action epsilon times and optimal 1-epsilon time. After all the computation, we will update the state, action, the next state, the reward and the Q-value. Now we need a Approximate Q-agent. In this part, we need to initialize the agent, and get the weights and Q-value. For ever feature that we extracted for the state and action corresponding weight should be extracted to return the Q-value of a state, which is summation. Lastly we finish the whole process by update all the current state and check if the training is finished or not.

3 Evaluation

For this project, the main evaluation i will be using is the provided analysis and grading file. the analysis file provided a series of test to answer each question listed on the tutorial. The grading file auto checks if the code i wrote for each function works or not.

4 Discussion

In this project, i have trained 2000 training episodes. The results came out pretty positively. By playing 100 games, the win rate is 100 percent after such short training. However we also need to consider the fact that this Pacman game is a shortened version. The average score that came out is 499.88, which is close to the full 503 points. In addition, the program passed all the test assigned.

This tutorial walked me thought how to implement value iteration process, how to construct Q-value agent, and how machine leaning model trains. It also showed me how Pacman self plays visually. The whole process is seamless and straight-forward. I helped me so much in understanding how reinforcement algorithm works and established my confidence in my future project regarding comps. However, the majority of this project was already provided for me, which made it so much easier to work with. I have attached some of the screenshot of the training result in the same GitHub repository.

4.1 Software Documentation

See attached **reinforcement file** for the code from this tutorial.

References

[1] UC Berkeley. *Project3: Reinforcement Learning*. 2014. URL: `http://ai.berkeley.edu/reinforcement.html`.