Q-Learning

PAN Mengyu,ZHANG kai

October 2020

1 Introduction

1.1 Introduction of Q-Learning

Q-Learning is one of reinforcement learning algorithm. As it doesn't use any transition probable distribution matrix, it is model-free. Its significant advantage is it can handle the problem with complex rewards and transitions.

1.2 Introduction of Pac-man

Pac-man is a video game. In this game, there is a pac-man and the ghost.The aim to control the pac-man to eat all the candies and avoid to be caught by the ghost. We create a virtual agent trained by Q-Learning to control our pac-man.

2 Method

2.1 Initialization

We create a variable to store the information for our pac-man like state and action using class Counter.

```
self.Q_table = Counter()
```

2.2 getQValue

We use class Counter to store the information of state so our variable has to be two-dimension.

```
# if the state and action have not been created
if type(self.Q_table[state]) is not Counter:
    self.Q_table[state]= Counter()
return self.Q_table[state][action]
```

2.3 computeValueFromQValues

We choose the best action for the state. If we had the best action, we give the corresponding reward. If not, we give 0.

```
best_action=self.computeActionFromQValues(state)
if best_action is None:
    return 0.0
else:
    return self.getQValue(state,best_action)
```

2.4 computeActionFromQValues

We use the function ”getLegalActions” to check if there is any possible legal action. If there exist the legal action, we choose the legal action with the max Q value.

```
actions = self.getLegalActions(state)
if len(actions) < 1:
    return None
max_value=self.getQValue(state, actions[0])
best_action=actions[0]
for a in actions:
    q_value = self.getQValue(state, a)  # Q value of action "a"
    if q_value > max_value:
        max_value = q_value
        best_action=a
return best_action
```

2.5 getAction

There are two types of choosing the next action, randomly choosing one from legal actions or choose the action with the biggest quality.

```
legalActions = self.getLegalActions(state)
if util.flipCoin(self.epsilon):
    action=random.choice(legalActions)
else:
    action=self.computeActionFromQValues(state)
return action
```

2.6 update

Update the Q-value at state S and action a.

$$Q(S,a) \leftarrow (1-\alpha)Q(S,a) + \alpha[R(S,a) + \gamma \max_a Q(S',a)]$$

```
next_max = self.computeValueFromQValues(nextState)
old_value=self.getQValue(state,action)
new_value=(1-self.alpha)*old_value+self.alpha*(reward+self.discount*next_max)
self.Q_table[state][action]=new_value
```

3 Result

The results of our code.

```
Average Score: 501.4
Scores:        503.0, 503.0, 499.0, 495.0, 503.0, 503.0, 503.0, 499.0, 503.0, 503.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```