Decision Theory: Q-Learning

CPSC 322 – Decision Theory 5

Textbook §12.5

Lecture Overview

1. Recap
2. Asynchronous Value Iteration
3. Q-Learning

Value of the Optimal Policy

- $Q^*(s,a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following the optimal policy.
- $V^*(s)$, where s is a state, is the expected value of following the optimal policy in state s.
- Q^* and V^* can be defined mutually recursively:

$$
\begin{aligned}
Q^*(s,a) &= \sum_{s'} P(s'|a,s)\left(r(s,a,s') + \gamma V^*(s')\right) \\
V^*(s) &= \max_a Q^*(s,a) \\
\pi^*(s) &= \arg\max_a Q^*(s,a)
\end{aligned}
$$

Value Iteration

- Idea: Given an estimate of the k-step lookahead value function, determine the $k+1$ step lookahead value function.
- Set V_0 arbitrarily.
  - e.g., zeros
- Compute Q_{i+1} and V_{i+1} from V_i:

$$\begin{aligned} Q_{i+1}(s,a) &= \sum_{s'} P(s'|a,s)\left(r(s,a,s') + \gamma V_i(s')\right) \\ V_{i+1}(s) &= \max_a Q_{i+1}(s,a) \end{aligned}$$

- If we intersect these equations at Q_{i+1}, we get an update equation for V:

$$V_{i+1}(s) = \max_a \sum_{s'} P(s'|a,s)\left(r(s,a,s') + \gamma V_i(s')\right)$$

Pseudocode for Value Iteration

procedure value_iteration(P, r, θ)
inputs:
  P is state transition function specifying $P(s'|a, s)$
  r is a reward function $R(s, a, s')$
  θ a threshold $\theta > 0$
returns:
  $\pi[s]$ approximately optimal policy
  $V[s]$ value function
data structures:
  $V_k[s]$ a sequence of value functions
begin
  for $k = 1 : \infty$
    for each state s
      $V_k[s] = \max_a \sum_{s'} P(s'|a, s)(R(s, a, s') + \gamma V_{k-1}[s'])$
    if $\forall s \; |V_k(s) - V_{k-1}(s)| < \theta$
      for each state s
        $\pi(s) = \arg\max_a \sum_{s'} P(s'|a, s)(R(s, a, s') + \gamma V_{k-1}[s'])$
      return π, V_k
end

Value Iteration Example: Gridworld

See
`http://www.cs.ubc.ca/spider/poole/demos/mdp/vi.html`.

Lecture Overview

Asynchronous Value Iteration

- You don't need to sweep through all the states, but can update the value functions for each state individually.
  - This converges to the optimal value functions, if each state and action is visited infinitely often in the limit.
  - Typically this is done by storing $Q[s, a]$

Pseudocode for Asynchronous Value Iteration

```
procedure asynchronous_value_iteration(P, r)
inputs:
    P is state transition function specifying P(s'|a, s)
    r is a reward function R(s, a, s')
returns:
    π approximately optimal policy
    Q value function
data structures:
    real array Q[s, a]
    action array π[s]
begin
    repeat
        select a state s
            select an action a
                Q[s, a] = Σ_{s'} P(s'|a, s)(R(s, a, s') + γ max_{a'} Q[s', a'])
    until some stopping criteria is true
    for each state s
        π[s] = arg max_a Q[s, a]
    return π, Q
end
```

Lecture Overview

1. Recap
2. Asynchronous Value Iteration
3. Q-Learning

Q-Learning

- This still required us to know the transition probabilities P.
- What if we just move around in the state space, never knowing these probabilities, but just taking actions and receiving rewards?

- We can use Asynchronous Value Iteration as the basis of a reinforcement learning algorithm
  - Why is this learning?

Q-Learning

- This still required us to know the transition probabilities P.
- What if we just move around in the state space, never knowing these probabilities, but just taking actions and receiving rewards?

- We can use Asynchronous Value Iteration as the basis of a reinforcement learning algorithm
  - Why is this learning?
  - It answers the question, "How should an agent behave in an MDP if it doesn't know the transition probabilities or the reward function?"

Q-Learning

- Choose actions:
  - Choose the action that appears to maximize Q (based on current estimates) most of the time
  - Choose a random action the rest of the time
  - Reduce the chance of taking a random action as time goes on

Q-Learning

- Choose actions:
  - Choose the action that appears to maximize Q (based on current estimates) most of the time
  - Choose a random action the rest of the time
  - Reduce the chance of taking a random action as time goes on
- Update the Q-functions:
  - Let α be a learning rate, $0 < \alpha < 1$
  - Let γ be the discount factor.
  - Whenever the agent starts out in state s, takes action a and ends up in state s', update $Q[s, a]$ as:

$$Q[s,a] \leftarrow (1-\alpha)Q[s,a] + \alpha[R(s,a,s') + \gamma \max_{a} Q[s',a']]$$

- Under reasonable conditions, Q-learning converges to the true Q, even though it never learns transition probabilities.
  - Why can we get away without them?

Q-Learning

- Choose actions:
  - Choose the action that appears to maximize Q (based on current estimates) most of the time
  - Choose a random action the rest of the time
  - Reduce the chance of taking a random action as time goes on
- Update the Q-functions:
  - Let α be a learning rate, $0 < \alpha < 1$
  - Let γ be the discount factor.
  - Whenever the agent starts out in state s, takes action a and ends up in state s', update $Q[s, a]$ as:

$$Q[s,a] \leftarrow (1-\alpha)Q[s,a] + \alpha[R(s,a,s') + \gamma \max_{a} Q[s',a']]$$

- Under reasonable conditions, Q-learning converges to the true Q, even though it never learns transition probabilities.
  - Why can we get away without them? Because the frequency of observing each s' already depends on them.
  - Thus, we say Q-learning is model-free.

Q-Learning Example: Gridworld Again

See http://www.cs.ubc.ca/spider/poole/demos/rl/q.html