Artificial Intelligence Programming

Q-learning

Chris Brooks

Department of Computer Science
University of San Francisco
We’re interested in stochastic environments

We can formulate this problem as a Markov decision process

Problem has:
- An initial state $s_0$
- A discrete set of states and actions
- A Transition model: $T(s, a, s')$ that indicates the probability of reaching state $s'$ from $s$ when taking action $a$.
- A reward function: $R(s)$
A *policy* is a solution to an MDP

Specifies the optimal action to take in every state

the optimal action is one that maximizes discounted expected utility from this state.

Therefore, the utility of a state is the reward for that state, plus the discounted expected reward for following the optimal policy from this state.

This is the Bellman equation:

\[ U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s') \]
Can’t solve Bellman equations directly

We can use value iteration to solve a system of Bellman equations.

- Set nonterminal utilities to random values.
- Foreach state, compute optimal action given these values.
- Use this to update the state’s estimated utility
- Continue until converged.
Problems with value iteration
  △ Convergence is slow, even when policy isn’t changing
  △ We don’t really care about the utility of a state.

Policy iteration updates policies directly.
24-4: Policy iteration algorithm

inititalize states to random utilities
Pi = random policy vector indexed by state
do
    U = evaluate the utility of each state for Pi
    for s in states
        a = find action that maximizes expected utility for that state
        Pi(s) = a
while some action changed
So far, we’ve assumed a great deal of knowledge.
In particular, we’ve assumed that a model of the world is known.
   ▲ This is the state transition model
What if we don’t have a model?
All we know is that there are a set of states, and a set of actions.
We still want to learn an optimal policy.
Learning a policy directly is difficult

Problem: our data is not of the form: \(<\text{state}, \text{action}\>

Instead, it’s of the form \(s_1, s_2, s_3, \ldots, R\).

Since we don’t know the transition function, it’s also hard to learn the utility of a state.

Instead, we’ll learn a function \(Q(s, a)\). This will estimate the “utility” of taking action \(a\) in state \(s\).
More precisely, $Q(s, a)$ will represent the value of taking $a$ in state $s$, then acting optimally after that.

$$Q(s, a) = R(s, a) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

The optimal policy is then to take the action with the highest $Q$ value in each state.

If the agent can learn $Q(s, a)$ it can take optimal actions even without knowing either the reward function or the transition function.
To learn $Q$, we need to be able to estimate the value of taking an action in a state even though our rewards are spread out over time.

We can do this iteratively.

Notice that $U(s) = \max_a Q(s, a)$

We can then rewrite our equation for $Q$ as:

$$Q(s, a) = R(s, a) + \gamma \max_{a'} Q(s', a')$$
24-9: Learning the Q function

6 Let’s denote our estimate of $Q(s, a)$ as $\hat{Q}(s, a)$

6 We’ll keep a table listing each state-action pair and estimated $Q$-value

6 the agent observes its state $s$, chooses an action $a$, then observes the reward $r = R(s, a)$ that it receives and the new state $s'$.

6 It then updates the Q-table according to the following formula:

$$\hat{Q}(s, a) = r + \gamma \max_{a'} \hat{Q}(s', a')$$
The agent uses the estimate of $\hat{Q}$ for $s'$ to estimate $\hat{Q}$ for $s$.

Notice that the agent doesn’t need any knowledge of $R$ or the transition function to execute this.

Q-learning is guaranteed to converge as long as:

△ Rewards are bounded

△ The agent selects state-action pairs in such a way that it each infinitely often.

△ This means that an agent must have a nonzero probability of selecting each $a$ in each $s$ as the sequence of state-action pairs approaches infinity.
So how to guarantee this?

Q-learning has a distinct difference from other learning algorithms we’ve seen:

The agent can select actions and observe the rewards they get.

This is called active learning

Issue: the agent would also like to maximize performance

- This means trying the action that currently looks best
- But if the agent never tries “bad-looking” actions, it can’t recover from mistakes.

Intuition: Early on, $\hat{Q}$ is not very accurate, so we’ll try non-optimal actions. Later on, as $\hat{Q}$ becomes better, we’ll select optimal actions.
24-12: Boltzmann exploration

- One way to do this is using Boltzmann exploration.
- We take an action with probability:

\[ P(a|s) = \frac{k^\hat{Q}(s,a)}{\sum_j k^\hat{Q}(s,a_j)} \]

- Where \( k \) is a temperature parameter.
- This is the same formula we used in simulated annealing.
Q-learning is an example of what’s called reinforcement learning.

Agents don’t see examples of how to act.

Instead, they select actions and receive rewards or punishment.

An agent can learn, even when it takes a non-optimal action.

Extensions deal with delayed reward and nondeterministic MDPs.
Q-learning is sometimes referred to as model-free learning.

Agent only needs to choose actions and receive rewards.

Problems:
- How to generalize?
- Scalability
- Speed of convergence

Q-learning has turned out to be an empirically useful algorithm.