Reinforcement Learning: Q-Learning

Andrea Danyluk
March 3, 2017

[Many of these slides adapted from CS 188, Stuart Russell, Andrew Moore, or Szepesvari and Sutton.]

Announcements

• Programming Assignment 2 due last night
  – Sign up for code reviews
• Assignment for Monday
  – Read Holte et al.’s AAAI 2016 paper on bi-directional search
  – Turn in brief reading response (no more than one page, 12pt font, 1.5 spacing) at start of class
• Sample midterm available online
• RL assignment now posted
  – Confirm partners with me by Monday 9am

Today’s Lecture

• Reinforcement Learning
  – Finish up TD example
  – Q-Learning

The TD Algorithm

function TD(S, R, S’, V)
    Input: S is the current state, S’ is the next state, R is the immediate reward for the transition, V is the array storing the current value estimates
    0: if S’ is a new non-terminal state, then V[S’] = R;
        if terminal state, V[S’] = 0
    1: δ ← R + γ · V[S’] − V[S]
    2: V[S] ← V[S] + α · δ
    3: return V

Initialize to arbitrary values. Often we choose 0.

Example: TD Policy Evaluation

Episodes:

\[ \begin{align*}
(1, 1) & \rightarrow (1, 2) -1, (1, 3) -1, (1, 2) -1, (1, 3) -1, (1, 3) -1, (1, 2) -1, (3, 3) -1, (4, 3) +100 \\
(1, 1) & \rightarrow (1, 2) -1, (1, 3) -1, (2, 3) -1, (2, 3) -1, (1, 3) -1, (3, 2) -1, (4, 2) -100 \\
\end{align*} \]

\[ V^\pi(s) = V(s) + \alpha (\text{sample} - V(s)) \]

\[ \text{sample} = R(s, \pi(s), s’) + \gamma V(s’) \]

\[ \gamma = 1, \alpha = 0.5, R = -1, \text{except where indicated} \]

Terminal States have 0 future rewards.
Example: TD Policy Evaluation

Episodes:

\[(1, 1) \rightarrow (1, 2) \rightarrow -1, (1, 2) \rightarrow -1, (1, 3) \rightarrow -1, (2, 3) \rightarrow -1, (3, 3) \rightarrow -1, (3, 2) \rightarrow -1, (4, 3) \rightarrow +100\]

\[(1, 1) \rightarrow (1, 2) \rightarrow -1, (1, 3) \rightarrow -1, (2, 3) \rightarrow -1, (3, 3) \rightarrow -1, (3, 2) \rightarrow -1, (4, 2) \rightarrow -100\]

\[V(s) = V(s) + \alpha(sample - V(s))\]

\[sample = R(s, \pi(s), s') + \gamma V(s')\]

\[V(1, 1) = V(1, 1) + 0.5 \cdot [R(1, 2) + V(1, 2) - V(1, 1)]\]

\[V(1, 1) = 0 + 0.5(-1 + -1 - 0) = -1\]
Example: TD Policy Evaluation

Episodes:

V(2,3) = -1 + 0.5(-1 + -1 + -1) = -1.5
V(2,3) = V(2,3) + 0.5[-1 + V(3,3) - V(2,3)]

Episodes:

V(1,3) = -1 + 0.5(-1 + -1 + -1) = -1.5
V(1,3) = V(1,3) + 0.5[-1 + V(2,3) - V(1,3)]

Episodes:

V(1,2) = -1.5 + 0.5(-1 + -1 + -1.5) = -1.75
V(1,2) = V(1,2) + 0.5[-1 + V(1,3) - V(1,2)]

Episodes:

V(1,1) = V(1,1) + 0.5[-1 + V(2,1) - V(1,1)]
V(1,1) = V(1,1) + 0.5[-1 + V(3,1) - V(1,1)]
V(1,1) = V(1,1) + 0.5[-1 + V(4,1) - V(1,1)]
V(1,1) = V(1,1) + 0.5[-1 + V(5,1) - V(1,1)]
Example: TD Policy Evaluation

Episodes:

\[(1, 1) \rightarrow (1, 2) -1, (1, 2) -1, (1, 3) -1, (2, 3) -1, (3, 3) -1, (4, 3) +100\]
\[(1, 1) \rightarrow (1, 2) -1, (1, 3) -1, (2, 3) -1, (3, 3) -1, (4, 3) -1, (4, 2) -100\]

\[V^*(s) = V^*(s) + \alpha(sample - V^*(s))\]

\[sample = R(s, \pi(s), s') + \gamma V^*(s')\]

\[V(3, 3) = V(3, 3) + 0.5[-1 + V(4, 3) - V(3, 3)]\]

\[V(3, 3) = -1 + 0.5(-1 - 1) = -1.5\]

\[V(3, 2) = V(3, 2) + 0.5[-1 + V(3, 3) - V(3, 2)]\]

\[V(3, 2) = -1 + 0.5(-1 - 1 - 1) = -1.75\]

\[V(3, 2) = -1 + 0.5(-1 + 0 - -1) = -1.75\]

\[\gamma = 1, \alpha = 0.5, R = -1\]

The Problem with TD Value Learning

- We now have a model-free way to do policy evaluation
- But we can’t turn it into a new policy without more work
  - Have an estimate of the state transition probabilities for the fixed policy
  - But need a model with complete probabilities for all actions
  - Can learn policy with exploring starts and generalized policy iteration

Terminal States have 0 future rewards.
What we really want

- A model-free way to do policy evaluation (as we explore)
- Ability to move toward a new (and ultimately optimal) policy

Active RL

- Given:
  - Ability to perceive states and rewards
  - Knowledge of available actions
  - No knowledge of \( P(s' | s, a) \)
  - No knowledge of rewards \( R(s, a, s') \)
- Goal: learn state values and optimal policy
- Learner actively explores the world
  - Tradeoff between exploration and exploitation

Recall: Optimal Values (Utilities)

\[
Q^*(s, a) = \sum P(s' | s, a) \cdot [R(s') + \gamma V^*(s')] ,
\]
where the sum is over all \( s' \)

\[
V^*(s) = \max_a Q^*(s, a)
\]

Aim to learn the Q values directly

Q-Learning

- An alternative TD method
- Choose an action in the given state, \( s \). Apply that action, \( a \). Now have:
  \( (s, a, s', r) \)
- Get sample of \( Q(s, a) \):
  \[\text{sample} = R(s, a, s') + \gamma \max_{a'} Q(s', a')\]
- Update \( Q(s, a) \) — i.e., compute a running average:
  \[
  Q(s, a) = (1 - \alpha) Q(s, a) + \alpha \text{sample}
  \]

Q-Learning Properties

Q-learning converges to the optimal policy
- If the learning rate is small enough
- If you explore enough
  - Want the exploration method to be greedy in the limit of infinite exploration
    - Aim to try each action in each state an infinite number of times
    - Need to eventually become greedy so that the agent’s actions become optimal with respect to the learned (true) model

Exploration/Exploitation Schemes

- \( \epsilon \)-greedy selection
  - When choosing an action, flip a coin
  - With probability \( \epsilon \), act randomly
  - Else follow the current policy (breaking ties randomly)
- \( \epsilon \)-greedy selection, but lower \( \epsilon \) over time
- Give some weight to actions the agent has not tried often
- More complex selection functions
Demo 1: Mazeworld

- Magenta: taking best action
- Yellow: exploring other action
- Smiling face: successfully completes action
- Sad face: transitions to unintended state

- Remember: this demo aims to minimize value – not maximize.

Demo 2: Crawler

- States?
- Actions?
- Rewards?

python crawler.py