Active Reinforcement Learning

David Cline
Active Reinforcement Learning

- Goal: earn the optimal policy as well as utilities.
- In particular, learn the best action for each state.
Exploration vs. Exploitation

- To learn optimal policy and utilities agent must
  - Explore the environment, and not get locked into a suboptimal policy.
  - Exploit good paths already learned to optimize utilities.
Active ADP Agent

• Use an "optimistic estimate" for unexplored paths to trade off exploration and exploitation:

\[ U^+(s) \leftarrow R(s) + \gamma \max_a f \left( \sum_{s'} P(s' | s, a) U^+(s'), N(s, a) \right) \]

- f is a function the optimistic utility \( U^+ \), and the number of times the action \( a \) from \( s \) has been taken: \( N(s, a) \)

• One function is:

\[ f(u, n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases} \]

- Best possible reward for any state
- The standard reward
Active Temporal Differencing (TD)

- Use same update as before, replacing the current best policy with the optimistic policy.

\[
U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))
\]
Q-learning

• Place utilities on taking actions rather than being in a particular state

  - $Q(s,a)$ – value of doing action $a$ in state $s$
  - $U(s) = \max Q(s,a)$
  - "Model free" because no model needed for $P(s' | s,a)$
  - Q-learning solves equation similar to Bellman

$$Q(s, a) = R(s) + \gamma \sum_{a'} P(s' | s, a) \max Q(s', a')$$
Q-learning Update

- Q-learning update rule:
  \[ Q(s, a) \leftarrow Q(s, a) + \alpha (R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)) \],

- (Q) Draw this update graphically.
- (Q) What is the term: \( \max Q(s', a') \)?
Q-learning Agent

- (Q) Perform a few Q updates on the world below, assuming that nonterminal states start with Q values of 0.
  - Use gamma=0.9, and alpha=0.25
- (Q) What do we need to store for each state?
Generalizations

• Usually state space is too big (or continuous)

• Use a model with base functions and coefficients, and learn the coefficients

\[
\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \cdots + \theta_n f_n(s)
\]

- \( \theta_i \) = coefficient we want to learn
- \( f_i \) = feature extracted from state (we pick these)
Widrow-Hoff rule (delta rule)

- One way to learn the coefficient:
  - Move according to the gradient of change with respect to the coefficients:

\[
\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j(s)}{\partial \theta_i} = \theta_i + \alpha (u_j(s) - \hat{U}_\theta(s)) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}
\]

- \(\theta_i\) = coefficient \(i\)
- \(\alpha\) = learning rate
- \(u_j(s)\) = actual reward from state \(s\) on trial \(j\)
- \(\hat{U}_\theta(s)\) = estimated reward
- \(\frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}\) = partial derivative