Chapter 13: Reinforcement Learning

CS 536: Machine Learning
Littman (Wu, TA)

Administration

Midterms due
Reinforcement Learning

[Read Chapter 13]

• [Exercises 13.1, 13.2, 13.4]
• Control learning
• Control policies that choose optimal actions
• Q learning
• Convergence

Control Learning

Consider learning to choose actions, e.g.,

• Robot learning to dock on battery charger
• Learning to choose actions to optimize factory output
• Learning to play Backgammon
Problem Characteristics

Note several problem characteristics:
• Delayed reward
• Opportunity for active exploration
• Possibility that state only partially observable
• Possible need to learn multiple tasks with same sensors/effectors

One Example: TD-Gammon

[Tesauro, 1995]
Learn to play Backgammon
Immediate reward
• +100 if win
• -100 if lose
• 0 for all other states
Trained by playing 1.5 million games against itself
Now approximately equal to best human player
The RL Problem

Goal: Learn to choose actions that maximize
\[ r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots , \text{ where } 0 \leq \gamma < 1 \]

Markov Decision Processes

Assume
- finite set of states \( S \); set of actions \( A \)
- at each discrete time agent observes state \( s_t \) in \( S \) and chooses action \( a_t \) in \( A \)
- then receives immediate reward \( r_t \) & state changes to \( s_{t+1} \)
- Markov assumption:
  - \( r_t = r(s_t, a_t) \) and \( s_{t+1} = \Delta(s_t, a_t) \) depend only on current state and action
  - \( \Delta \) and \( r \) may be nondeterministic
  - \( \Delta \) and \( r \) not necessarily known to agent
Agent's Learning Task

Execute actions in environment, observe results, and

- learn action policy \( \pi : S \rightarrow A \) that maximizes
  \[
  E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots]
  \]
  from any starting state in \( S \)
- here \( 0 \leq \gamma < 1 \) is the discount factor for future rewards

Different Learning Problem

Note something new:

- Target function is \( \pi : S \rightarrow A \)
- but we have no training examples of form \( <s, a> \)
- training examples are of form \( <<<s, a>, r> \)
Value Function

To begin, consider deterministic worlds...
For each possible policy the agent might adopt, we can define an evaluation function over states

\[ V^\pi(s) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \]

where \( r_t, r_{t+1}, \ldots \) are generated by following policy starting at state \( s \)

Restated, the task is to learn the optimal policy:

\[ \pi^* = \arg\max_{\pi} V^\pi(s), (\pi,s) \]

Example MDP

- \( r(s, a) \) (immediate reward) values
- \( Q(s, a) \) values
- \( V^*(s) \) values
- One optimal policy
What to Learn

We might try to have agent learn the evaluation function $V^*$ (we write as $V^*$)

It could then do a lookahead search to choose best action from any state $s$ because

$$V^*(s) = \arg\max_a [r(s, a) + V^*(d(s, a))]$$

A problem:

- This works well if agent knows $\mathcal{S} \subseteq \mathcal{A} \subseteq \mathcal{S}$, and $r: \mathcal{S} \times \mathcal{A} \to \mathcal{R}$
- But when it doesn't, it can't choose actions this way

Q Function

Define new function very similar to $V^*$

$$Q(s, a) \equiv r(s, a) + V^*(d(s, a))]$$

If agent learns $Q$, it can choose optimal action even without knowing $d$!

$$V^*(s) = \arg\max_a [r(s, a) + V^*(d(s, a))]$$

$$V^*(s) = \arg\max_a Q(s, a)$$

$Q$ is the evaluation function the agent will learn
Training Rule to Learn $Q$

Note $Q$ and $V^*$ closely related:

$$V^*(s) = \max_{a'} Q(s, a')$$

Which allows us to write $Q$ recursively as

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma V^*(d(s_t, a_t))$$

$$= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')$$

Nice! Let $\hat{Q}$ denote learner’s current approximation to $Q$. Use training rule

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

where $s'$ is the state resulting from applying action $a$ in state $s$.

---

Q Learning in Deterministic Case

For each $s, a$ initialize table entry

$$\hat{Q}(s, a) \leftarrow 0$$

Observe current state $s$

Do forever:

- Select an action $a$ and execute it
- Receive immediate reward $r$
- Observe the new state $s'$
- Update the table entry for $\hat{Q}(s, a)$ via:
  $$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$
- $s \leftarrow s'$
Updating $\hat{Q}$

\[ \hat{Q}(s_1, a_{\text{right}}) = r + 0.9 \max_a \hat{Q}(s_2, a) \]
\[ = 0 + 0.9 \max \{63, 81, 100\} = 90 \]

notice if rewards non-negative, then
\[ (s, a, n) \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a) \]
and
\[ (s, a, n) 0 \leq \hat{Q}_n(s, a) \leq Q(s, a) \]

Convergence Proof

$\hat{Q}$ converges to $Q$. Consider case of deterministic world where see each $<s, a>$ visited infinitely often.

Proof: Define a full interval to be an interval during which each $<s, a>$ is visited.
During each full interval the largest error in $\hat{Q}$ table is reduced by factor of \[. \]
Let $\hat{Q}_n$ be table after $n$ updates, and $n$ be the maximum error in $\hat{Q}_n$; that is
\[ \square_n = \max_{s,a} |\hat{Q}_n(s, a) - Q(s, a)| \]
Proof Continued

For table entry $\hat{Q}_n(s, a)$ updated on iteration $n + 1$, the error in the revised estimate $\hat{Q}_{n+1}(s, a)$ is

$$|\hat{Q}_{n+1}(s, a) - Q(s, a)|$$

$$= |(r + \max_a \hat{Q}_n(s', a')) - (r + \max_a Q(s', a'))|$$

$$= \max_a |\hat{Q}_n(s', a') - Q(s', a')|$$

$$\leq \max_a |\hat{Q}_n(s', a') - Q(s', a')|$$

$$\leq \max_{a', s''} |\hat{Q}_n(s'', a') - Q(s'', a')|$$

$$|\hat{Q}_{n+1}(s, a) - Q(s, a)| \leq \bar{D}_n$$

Note that we used the fact that

$$|\max_a f_1(a) - \max_a f_2(a)| \leq \max_a |f_1(a) - f_2(a)|$$

Nondeterministic Case

What if reward and next state are non–deterministic?

We redefine $V, Q$ by taking expected values

$$V^\pi(s) = E[r_t + \boxplus r_{t+1} + \boxplus^2 r_{t+2} + \ldots]$$

$$= E[\sum_{i=0}^{\infty} \boxplus r_{t+i}]$$

$$Q(s, a) = E[r(s, a) + \boxplus V^\pi(s, a)]$$
Nondeterministic Case

Q learning generalizes to nondeterministic worlds

Alter training rule to

$$\hat{Q}_n(s, a) = (1-\frac{1}{visits_n(s, a)})Q_{n-1}(s, a) + \frac{1}{visits_n(s, a)} [r + \max_a Q_{n-1}(s', a')]$$

where

$$\frac{1}{visits_n(s, a)} = 1/(1 + visits_n(s, a))$$

Can still prove convergence of $\hat{Q}$ to $Q$ [Watkins and Dayan, 1992].

Temporal Difference Learning

$Q$ learning: reduce discrepancy between successive $Q$ estimates

One step time difference:

$$Q^{(1)}(s_t, a_t) = r_t + \max_a Q(s_{t+1}, a)$$

Why not 2 steps?

$$Q^{(2)}(s_t, a_t) = r_t + \max_a Q(s_{t+1}, a)$$

Or $n$?

$$Q^{(n)}(s_t, a_t) = r_t + \max_a Q(s_{t+n}, a)$$

Blend all of these:

$$Q(s_t, a_t) = (1-\frac{1}{\lambda}) [Q^{(1)}(s_t, a_t) + \frac{1}{\lambda}Q^{(2)}(s_t, a_t) + ...]$$
**Temporal Difference Learning**

\[ Q(s_t, a_t) = (1-l)(Q(s_t, a_t) + lQ(s_{t+1}, a_{t+1})) \]

Equivalent expression:

\[ Q(s_t, a_t) = r_t + l[(1-l)\max_a \hat{Q}(s_t, a_t) + l\hat{Q}(s_{t+1}, a_{t+1})] \]

**TD(l) algorithm** uses above training rule

- Sometimes converges faster than \( Q \) learning (not understood in control case)
- Converges for learning \( V \) for any \( 0 \leq l \leq 1 \) (Dayan, 1992)
- Tesauro's TD–Gammon uses this algorithm to estimate the value function via self play.

**Subtleties & Ongoing Research**

- Replace \( \hat{Q} \) table with neural net or other generalizer
- Handle case where state only partially observable
- Design optimal exploration strategies
- Extend to continuous action, state
- Learn and use \( \hat{S}A\hat{S} \)
- Relationship to dynamic programming and heuristic search