
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

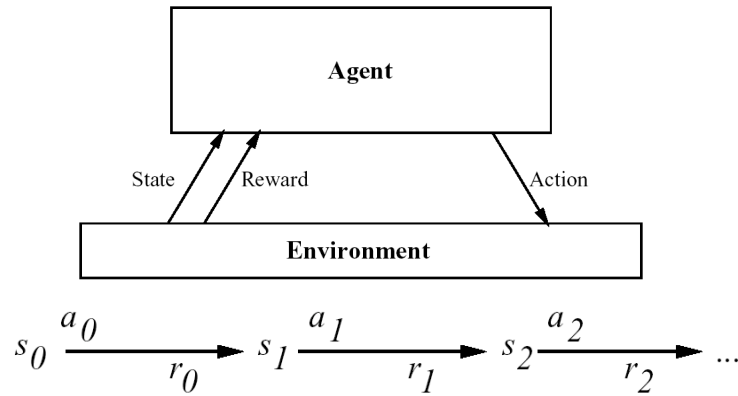
Immediate reward

- +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 $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$, 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 $s_{t+1} = \gamma(s_t, a_t)$ depend only on *current* state and action
 - γ and r may be nondeterministic
 - γ 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

$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots]$$

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 $\langle s, a \rangle$
- training examples are of form $\langle \langle s, a \rangle, r \rangle$

Value Function

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

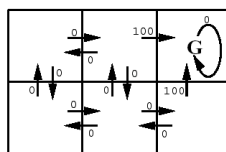
$$V^\pi(s) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

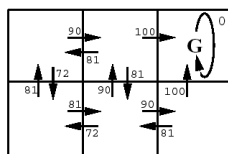
where r_t, r_{t+1}, \dots are generated by following policy π starting at state s

Restated, the task is to learn the optimal policy π^* : $\pi^* \equiv \operatorname{argmax}_\pi V^\pi(s), (\forall s)$.

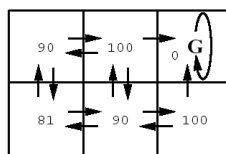
Example MDP



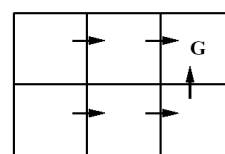
$r(s, a)$ (immediate reward) values



$Q(s, a)$ 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

$$V^*(s) = \operatorname{argmax}_a [r(s, a) + \gamma V^*(\tau(s, a))]$$

A problem:

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

Q Function

Define new function very similar to V^*

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\tau(s, a))$$

If agent learns Q , it can choose optimal action even without knowing V^* !

$$V^*(s) = \operatorname{argmax}_a [r(s, a) + \gamma V^*(\tau(s, a))]$$

$$V^*(s) = \operatorname{argmax}_a Q(s, a)$$

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

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma V^*(\phi(s_t, a_t)) \\ &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

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

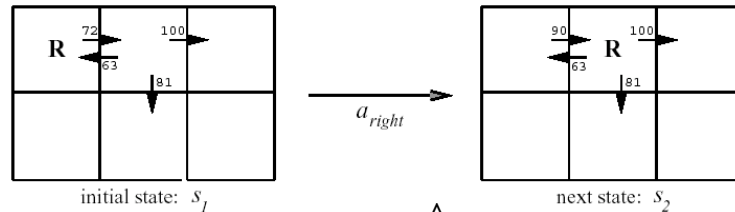
$$\hat{Q}(s, a) \leftarrow 0$$

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}



$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_a \hat{Q}(s_2, a)$$

$$\leftarrow 0 + 0.9 \max \{63, 81, 100\} = 90$$

notice if rewards non-negative, then

$$(\forall s, a, n) \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a)$$

and

$$(\forall s, a, n) 0 \leq \hat{Q}_n(s, a) \leq Q(s, a)$$

Convergence Proof

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

Proof. Define a full interval to be an interval during which each $\langle s, a \rangle$ is visited.

During each full interval the largest error in \hat{Q} table is reduced by factor of γ

Let \hat{Q}_n be table after n updates, and ϵ_n be the maximum error in \hat{Q}_n ; that is

$$\epsilon_n = \max_{s,a} |\hat{Q}_n(s, a) - Q(s, a)|$$

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

$$\begin{aligned} & |\hat{Q}_{n+1}(s, a) - Q(s, a)| \\ &= |(r + \gamma \max_{a'} \hat{Q}_n(s', a')) - (r + \gamma \max_{a'} Q(s', a'))| \\ &= \gamma |\max_{a'} \hat{Q}_n(s', a') - \max_{a'} Q(s', a')| \\ &\leq \gamma \max_{a'} |\hat{Q}_n(s', a') - Q(s', a')| \\ &\leq \gamma \max_{a', s''} |\hat{Q}_n(s'', a') - Q(s'', a')| \\ &|\hat{Q}_{n+1}(s, a) - Q(s, a)| \leq \gamma \gamma_n \end{aligned}$$

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

$$\begin{aligned} V(s) &\equiv E[r_t + \gamma V_{t+1} + \gamma^2 V_{t+2} + \dots] \\ &\equiv E[\sum_{i=0}^{\infty} \gamma^i r_{t+i}] \end{aligned}$$

$$Q(s, a) \equiv E[r(s, a) + \gamma V^*(\phi(s, a))]$$

Nondeterministic Case

Q learning generalizes to
nondeterministic worlds

Alter training rule to

$$\hat{Q}_n(s, a) \equiv (1 - \alpha_n) \hat{Q}_{n-1}(s, a) + \alpha_n [r + \alpha \max_{a'} \hat{Q}_{n-1}(s', a')]$$

where

$$\alpha_n = 1 / (1 + \text{visits}_n(s, a)).$$

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:

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

Why not 2 steps?

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

Or n ?

$$Q^{(n)}(s_t, a_t) \equiv r_t + \dots + \alpha^{n-1} r_{t+n-1} + \alpha^n \max_a Q(s_{t+n}, a)$$

Blend all of these:

$$Q \equiv (s_t, a_t) \equiv (1 - \alpha) [Q^{(1)}(s_t, a_t) + \alpha Q^{(2)}(s_t, a_t) + \dots]$$

Temporal Difference Learning

$$Q^\alpha(s_t, a_t) \equiv (1-\alpha) [Q^{(1)}(s_t, a_t) + \alpha Q^{(2)}(s_t, a_t) + \dots]$$

Equivalent expression:

$$Q^\alpha(s_t, a_t) \equiv r_t + \alpha [(1-\alpha) \max_a \hat{Q}(s_t, a_t) + \alpha Q^\alpha(s_{t+1}, a_{t+1})]$$

TD(α) algorithm uses above training rule

- Sometimes converges faster than Q learning (not understood in control case)
- converges for learning V for any $0 \leq \alpha \leq 1$ (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 $\hat{Q}: S \times A \rightarrow S$
- Relationship to dynamic programming and heuristic search