Reinforcement Learning

• 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

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
Reinforcement Learning Problem

Environment

Agent

state

action

reward

s_0 \xrightarrow{a_0} r_0 \xrightarrow{s_1} a_1 \xrightarrow{r_1} s_2 \xrightarrow{a_2} r_2 \xrightarrow{\ldots}

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 Process

Assume

- finite set of states $S$
- set of actions $A$
- at each discrete time, agent observes state $s_t \in S$ and choose action $a_t \in A$
- then receives immediate reward $r_t$
- and state changes to $s_{t+1}$
- Markov assumption: $s_{t+1} = \delta(s_t, a_t)$ and $r_t = r(s_t, a_t)$
  - i.e., $r_t$ and $s_{t+1}$ depend only on current state and action
  - functions $\delta$ and $r$ may be nondeterministic
  - functions $\delta$ and $r$ no necessarily known to agent
Agent’s Learning Task

Execute action 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

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 $\pi$ the agent might adopt, we can define an evaluation function over states

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

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

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

Restated, the task is to learn the optimal policy $\pi^*$

$$\pi^* \equiv \arg\max_\pi V^\pi(s), (\forall s)$$
$r(s,a)$ (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 $V_{\pi^*}$ (which we write as $V^*$). We could then do a lookahead search to choose best action from any state $s$ because

$$\pi^* (s) \equiv \arg\max_a \left[ r(s,a) + \gamma \ V^*(\delta (s,a)) \right]$$

A problem:

- This works well if agent knows a $\delta : S \times A \rightarrow S$, and $r : S \times A \rightarrow \mathbb{R}$
- But when it doesn’t, we can’t choose actions this way
Q Function

Define new function very similar to $V^*$

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

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

$$\pi^*(s) \equiv \arg\max_a [r(s, a) + \gamma V^*(\delta (s, a))]$$

$$\pi^*(s) \equiv \arg\max_a Q(s, a)$$

$Q$ 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

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

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

Let $\hat{Q}$ denote learner’s current approximation to $Q$.

Consider 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 for Deterministic Worlds**

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 \( \hat{Q}(s, a) \) as follows:

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

- \( s \leftarrow s' \)
Updating

initial state: $s_1$

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

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

notice if rewards non-negative, then

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

and

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

$\hat{Q}$ converges to $Q$. Consider case of deterministic world where 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 $\gamma$

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

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

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

$$\left|\hat{Q}_{n+1}(s, a) - Q(s, a)\right| = \left|(r + \gamma \max_{a'} \hat{Q}_n(s', a')) - (r + \gamma \max_{a'} Q(s', a'))\right|$$

$$= \gamma \left|\max_{a'} \hat{Q}_n(s', a') - \max_{a'} Q(s', a')\right|$$

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

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

$$\left|\hat{Q}_{n+1}(s, a) - Q(s, a)\right| \leq \gamma \Delta_n$$

Note we used general fact that

$$\left|\max_a f_1(a) - \max_a f_2(a)\right| \leq \max_a |f_1(a) - f_2(a)|$$
Nondeterministic Case

What if reward and next state are non-deterministic? We redefine $V, Q$ by taking expected values

$$V^\pi(s) \equiv E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...]$$

$$\equiv E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$$

$$Q(s, a) \equiv E[r(s, a) + \gamma V^\pi(\delta(s, a))]$$
Nondeterministic Case

$Q$ learning generalizes to nondeterministic worlds

Alter training rule to

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

where

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

Can still prove converge of $\hat{Q}$ to $Q$ [Watkins and Dayan, 1992]
Temporal Difference Learning

\( Q \) learning: reduce discrepancy between successive \( Q \) estimates

One step time difference:

\[
Q^{(1)}(s_t, a_t) \equiv r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)
\]

Why not two steps?

\[
Q^{(2)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2}, a)
\]

Or \( n \)?

\[
Q^{(n)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n \max_a \hat{Q}(s_{t+n}, a)
\]

Blend all of these:

\[
Q^\lambda(s_t, a_t) \equiv (1 - \lambda)[Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \ldots]
\]
Temporal Difference Learning

\[ Q^\lambda (s_t, a_t) \equiv (1-\lambda) \left[ Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \ldots \right] \]

Equivalent expression:

\[ Q^\lambda (s_t, a_t) \equiv r_t + \gamma \left( 1 - \lambda \right) \max_a \hat{Q}(s_t, a_t) + \lambda Q^\lambda (s_{t+1}, a_{t+1}) \]

TD(\lambda) algorithm uses above training rule

- Sometimes converges faster than Q learning
-_converges for learning \( V^* \) for any \( 0 \leq \lambda \leq 1 \) (Dayan, 1992)
- Tesauro’s TD-Gammon uses this algorithm
Subtleties and Ongoing Research

- Replace $\hat{Q}$ table with neural network or other generalizer
- Handle case where state only partially observable
- Design optimal exploration strategies
- Extend to continuous action, state
- Learn and use $d : S \times A \rightarrow S$, $d$ approximation to $\delta$
- Relationship to dynamic programming
RL Summary

• Reinforcement learning (RL)
  – control learning
  – delayed reward
  – possible that the state is only partially observable
  – possible that the relationship between states/actions unknown

• Temporal Difference Learning
  – learn discrepancies between successive estimates
  – used in TD-Gammon

• $V(s)$ - state value function
  – needs known reward/state transition functions
RL Summary

• $Q(s,a)$ - state/action value function
  – related to $V$
  – does not need reward/state trans functions
  – training rule
  – related to dynamic programming
  – measure actual reward received for action and future value using current $Q$ function
  – deterministic - replace existing estimate
  – nondeterministic - move table estimate towards measure estimate
  – convergence - can be shown in both cases