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

Goal: learn to choose actions that maximize
\[ r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \]
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
\[ E[r_1 + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots] \]
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 \( \langle s, a, r \rangle \)
- training examples are of form \( \langle s, a >, r \rangle \)
Value Function
To begin, consider deterministic worlds …
For each possible policy $\pi$ the agent might adopt, we
can define an evaluation function over states
$$V^\pi(s) = r_s' + \gamma r_{t+1}' + \gamma^2 r_{t+2}' + \ldots$$
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 $\pi^*$
$$\pi^* = \arg\max_{\pi} V^\pi(s), \forall s$$

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

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
$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))$$
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:
  $$\hat{Q}(s,a) \leftarrow r + \gamma \max_a \hat{Q}(s',a')$$
• $s \leftarrow s'$

$Q$ Function
Define new function very similar to $V^*$
$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))$$
If agent learns $Q$, it can choose optimal action even
without knowing $\delta$!
$$\pi^*(s) = \arg\max_a \left[ r(s,a) + \gamma V^*(\delta(s,a)) \right]$$
$$\pi^*(s) = \arg\max_a Q(s,a)$$
$Q$ is the evaluation function the agent will learn
Updating

\[ Q(s_t, a_{t\text{old}}) \leftarrow r + \gamma \max_a Q(s_{t+1}, a) \]

\[ \equiv 0.9 \max \{63, 81, 100\} = 90 \]

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

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

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 \( \Delta_n \) be the maximum error in \( \hat{Q} \); 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 \( \hat{Q}_{n+1}(s, a) \) is

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

\[ = \gamma \max_{s', a'} |\hat{Q}(s', a') - \max_a Q(s', a')| \]

\[ \leq \gamma \max_{s', a'} |\hat{Q}(s', a') - Q(s', a')| \]

Note we used general fact that

\[ \max_{s, a} f(a) - \max_{s, a} f(a) \leq \max_{s, a} |f(a) - f(a)| \]

Nondeterministic Case

What if reward and next state are non-deterministic?

We redefine \( V, Q \) by taking expected values

\[ V^\pi(s) = E[r_{t+1} + \gamma V^\pi(s_{t+1}) + \ldots] \]

\[ = E \sum_{t=0}^\infty \gamma^t r_{t+1} \]

\[ Q(s, a) = E[r(s, a) + \gamma V^\pi(s_{t+1})] \]

Temporal Difference Learning

\( Q \) learning: reduce discrepancy between successive

\( Q \) estimates

One step time difference:

\[ Q^{(1)}(s, a) = r + \gamma \max_a Q(s_{t+1}, a) \]

Why not two steps?

\[ Q^{(2)}(s, a) = r + \gamma r_{t+1} + \gamma^2 \max_a Q(s_{t+2}, a) \]

Or \( n \)?

\[ Q^{(n)}(s, a) = r + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n \max_a Q(s_{t+n}, a) \]

Blend all of these:

\[ Q(s, a) = (1-\lambda) Q^{(1)}(s, a) + \lambda Q^{(2)}(s, a) + \lambda^2 Q^{(3)}(s, a) + \ldots \]
Temporal Difference Learning

\[ Q'(s, a) = (1-\lambda)[Q^0(s, a) + \lambda Q^0(s', a') + \lambda^2 Q^0(s'', a'') + \ldots] \]

Equivalent expression:

\[ Q'(s, a) = r + \gamma \max_a Q(s', a') + \lambda Q'(s', a') \]

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 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