Reinforcement Learning
Agent / Environment Interaction

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

Different rewards
Emphasize
different behavior
Examples and Properties (1)

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
Examples and Properties (2)

• Each unique location is NOT a state
• Can be applied to any autonomous agent.
  – Software agents, web-crawlers
• Not unlike Pavlovian conditioning
  – Condition the learner to optimal behavior
• Agent receives indirect & delayed rewards, no guidance or corrections given.
  – Different than supervised learning
• Agent does not necessarily know how their actions change their state.
• Recent move towards competing agents.
A Successful Example

[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
Different Environments and Agents

• Environments
  – Direct or indirect rewards/training

• Agents
  – Deterministic versus non-deterministic actions.
  – Incomplete knowledge of effect of actions
  – Incomplete knowledge of the state
  – Agent memory or model of the environment

• Regardless of differences always have:
  – $\delta(s_t, a_t)$ chooses the action to perform
  – $r(s_t, a_t)$ provides the reward given the action
  – Agent may not know either function
Markov Decision Process RL

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 $r_t$
- 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 $r$ not necessarily known to agent
Difference to Supervised Learning

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

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 $\langle \langle s, a \rangle, r \rangle$
Learning in a Deterministic World

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 $\pi^*$

$$\pi^* \equiv \arg\max_{\pi} V^\pi(s), (\forall s)$$
Grid World

\( \gamma = 0.9 \)

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

\( Q(s, a) \) values

\( V^*(s) \) values

One optimal policy
Use of $Q$ instead of $V$ Function (1)

We might try to have agent learn the evaluation function $V^{\pi^*}$ (which we write as $V^*$)

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

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

A problem:

- This works well if agent knows $\delta : S \times A \to S$, and $r : S \times A \to \mathbb{R}$
- But when it doesn’t, it can’t choose actions this way
Use of $Q$ instead of $V$ Function (2)

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 $\delta$!

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

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

$Q$ is the evaluation function the agent will learn
Learning $\hat{Q}$ an Estimate of $Q$

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

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

Which allows us to write $Q$ recursively as

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

Nice! 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 $s$
Learning $\hat{Q}$ - Deterministic Worlds

For each $s, a$ initialize table entry $Q(s, a) \leftarrow 0$

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 $\hat{Q}$

\[
\hat{Q}(s_1, \alpha_{\text{right}}) \leftarrow r + \gamma \max_{\alpha'} \hat{Q}(s_2, \alpha')
\]
\[
\leftarrow 0 + 0.9 \max\{63, 81, 100\}
\]
\[
\leftarrow 90
\]

notice if rewards non-negative, then

\[
(\forall s, \alpha, n) \quad \hat{Q}_{n+1}(s, \alpha) \geq \hat{Q}_n(s, \alpha)
\]

and

\[
(\forall s, \alpha, n) \quad 0 \leq \hat{Q}_n(s, \alpha) \leq Q(s, \alpha)
\]
Reading

• Next Lecture
  – Pages page 377 – 386

• Interested readers
  – NASA Mars rovers and reinforcement learning
  – http://anytime.cs.umass.edu/~shlomo/research/NASA01.html
Convergence: $\lim_{n \to \infty} \hat{Q} = Q$

**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

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

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

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

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

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

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

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

What if reward and next state are non-deterministic?

We redefine $V, Q$ by taking expected values

$$V^x(s) \equiv E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots]$$
$$\equiv E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$$

$$Q(s, a) \equiv E\left[r(s, a) + \gamma V^*(s')\right]$$
Learning $\hat{Q}$ – Non-Deterministic Worlds

$Q$ 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

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

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

$Q$ 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 $n$?

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

Blend all of these:

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

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

Equivalent expression:

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

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
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 $\hat{\delta} : S \times A \rightarrow S$
- Relationship to dynamic programming