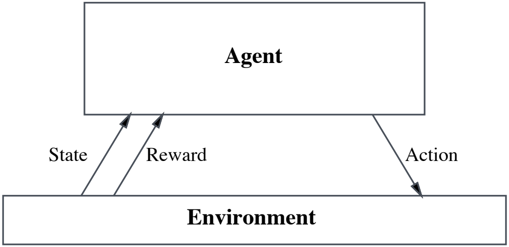
Reinforcement Learning



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \dots$$

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 $a_t \in A$ (among the possible actions in state s_t)
- then receives immediate reward r_t, that can be positive $(+)$, negative (-), o neutral (0)
- 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 δ and r may be nondeterministic
  - functions δ and r not necessarily known to agent

Agent's Learning Task

Agent goal: execute actions in environment, observe results, and

- learn action policy $\pi : S \to A$ that maximizes

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

  from any starting state in S
- here $0 \le \gamma < 1$ is the discount factor for future rewards

Note the difference with respect to supervised leaning:

- agent needs to learn a function $\pi : S \to A$
- using training examples of form $((s, a), r)$ (not (s, a))

Value Function

To begin, consider deterministic environments:

- for each possible policy π the agent might adopt, we can define an evaluation function over states

$$V^{\pi}(s) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$
$$\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

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

- restated, the task is to learn the optimal policy π^*

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

What to Learn

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

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

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

Problem:

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

$Q()$ Function

Solution:

- define new function very similar to V^*

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

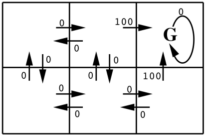
- if agent learns Q, it can choose optimal action even without knowing δ!

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

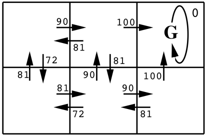
$$\pi^*(s) = arg \max_a Q(s,a)$$

Thus, Q is the evaluation function the agent will learn

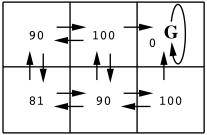
Example



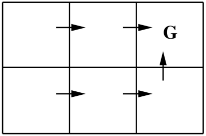
$r(s,a)$ (immediate reward) values



$Q(s,a)$ values



$V^*(s)$ values



One optimal policy

Training Rule to Learn Q

Note that Q and V^* are closely related:

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

Which allows us to write Q RECURSIVELY as

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

Nice! Let $\hat{Q}$ denote learner's current approximation to Q. Consider the following training rule:

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

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

$Q - Learning$ Algorithm

In case of deterministic worlds

1. for each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$
2. observe current state s
3. do forever
   1. select an action a and execute it
   2. receive immediate reward r
   3. observe the new state s'
   4. update the table entry for $\hat{Q}(s, a)$ as follows:

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

   5. $s \leftarrow s'$

$Q - Learning$ Algorithm

In case of deterministic worlds

1. for each s, a initialize table entry $\hat{Q}(s,a) \leftarrow 0$
2. observe current state s
3. do forever
   1. $\boxed{\text{select an action } a \text{ and execute it}}$ how to select ?
   2. receive immediate reward r
   3. observe the new state s'
   4. update the table entry for $\hat{Q}(s,a)$ as follows:

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

   5. $s \leftarrow s'$

$Q-\mathit{Learning}$ Algorithm

Università degli Studi di Padova

In case of deterministic worlds

1. for each s, a initialize table entry $\hat{Q}(s,a) \leftarrow 0$
2. observe current state s
3. do forever
   1. $\boxed{\text{select an action } a \text{ and execute it}}$

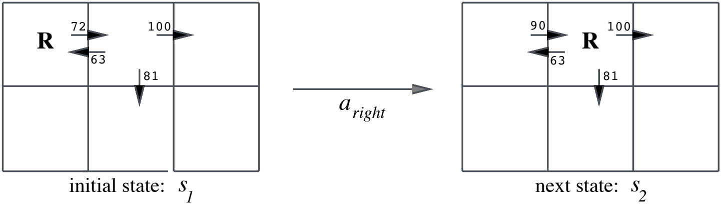
      strategy → random (exploration)

      strategy → $arg\ \max_a \hat{Q}(s,a)$ (exploitation)
   2. receive immediate reward r
   3. observe the new state s'
   4. update the table entry for $\hat{Q}(s,a)$ as follows:

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

   5. $s \leftarrow s'$

Example of application...



$$
\begin{aligned}
\hat{Q}(s_1, a_{\rightarrow}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\
&\leftarrow 0 + 0.9 \ \max\{66, 81, 100\} \\
&\leftarrow 90
\end{aligned}
$$

notice if rewards non-negative, then

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

and

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

$\hat{Q}$ converges to Q

Consider case of deterministic world where each (s, a) is 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 γ

Let $\hat{Q}_n$ be table after n updates, and Δ_n be the maximum error in $\hat{Q}_n$, i.e.

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

Convergence contd.

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

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

Note we used general fact that

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

Nondeterministic Case

Università degli Studi di Padova

What if reward and next state are non-deterministic?

We redefine V, Q by taking expected values

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

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

To learn, alter training rule to

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

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

Under specific conditions, can still prove convergence of $\hat{Q}$ to Q

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?

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

Blend all of these:

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

Temporal Difference Learning contd.

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

Equivalent expression:

$$\begin{aligned} Q^{\lambda}(s_t, a_t) &= r_t + \gamma[\ \ (1-\lambda)\max_a \hat{Q}(s_t, a_t) \\ &\qquad\qquad\quad +\lambda\ Q^{\lambda}(s_{t+1}, a_{t+1})] \end{aligned}$$

TD(λ) algorithm uses above training rule

- sometimes converges faster than Q learning
- converges for learning V^* for any $0 \leq \lambda \leq 1$
- Tesauro's TD-Gammon uses this algorithm

Many improvements/extensions are possible
— one is to replace $\hat{Q}$ table with a (deep) neural net!