

Reinforcement Learning

Based on “Machine Learning”, T. Mitchell, McGRAW Hill, 1997, ch. 13

Acknowledgement:

The present slides are an adaptation of slides drawn by T. Mitchell

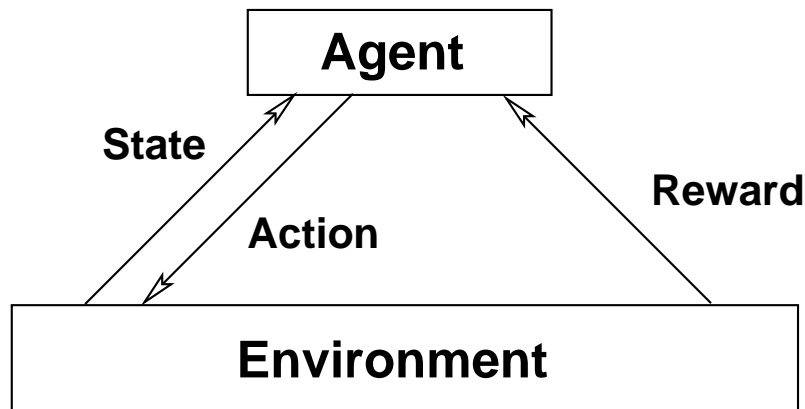
Reinforcement Learning — Overview

- Task: **Control learning**
make an autonomous **agent** (robot) to perform actions, observe consequences and learn a control strategy
- The **Q learning** algorithm — main focus of the chapter
acquire **optimal control strategies** from delayed **rewards**, even when the agent has no prior knowledge of the effect of its actions on the environment
- Reinforcement Learning is related to **dynamic programming**, used to solve optimization problems.

While DP assumes that the agent/program knows the effect (and rewards) of all its actions, in RL the agent has to experiment in the real world.

Reinforcement Learning Problem

2.



Target function: $\pi : S \rightarrow A$

Goal:

maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$$

where $0 \leq \gamma < 1$

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

Example: play Backgammon (TD-Gammon [Tesauro, 1995])

Immediate reward: +100 if win, -100 if lose, 0 otherwise

Other examples: robot control, flight/taxy scheduling, optimizing factory output

Control learning characteristics

- **training examples** are not provided (as $\langle s, \pi(s) \rangle$);
the trainer provides a (possibly delayed) reward $\langle \langle s, a \rangle, r \rangle$
- learner faces the **problem of temporal credit assignment**:
which actions are to be credited for the actual reward
- especially in case of continuous spaces there is an opportunity for the learner to actively perform space **exploration**
- the current **state** may be only **partially observable**;
the learner must consider previous observations to improve the current observability

Learning Sequential Control Strategies Using Markov Decision Processes

- assume a finite set of states S and the set of actions A
- at each discrete time t the agent observes the state $s_t \in S$ and chooses an action $a_t \in A$
- then it receives an immediate reward r_t and the state changes to s_{t+1}
- the **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 the current state and action
- the functions δ and r may be non-deterministic;
they may not necessarily be known to the 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 ;

$\gamma \in [0, 1)$ is the discount factor for future rewards.

Note: In the sequel, we will consider that the actions are taken in a deterministic way, and show how the problem can be solved. Then we will generalize to the non-deterministic case.

The Value Function V

6.

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

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

with r_t, r_{t+1}, \dots generated according to the applied policy π starting at state s . Therefore, the learner's **task** is to learn the optimal policy π^*

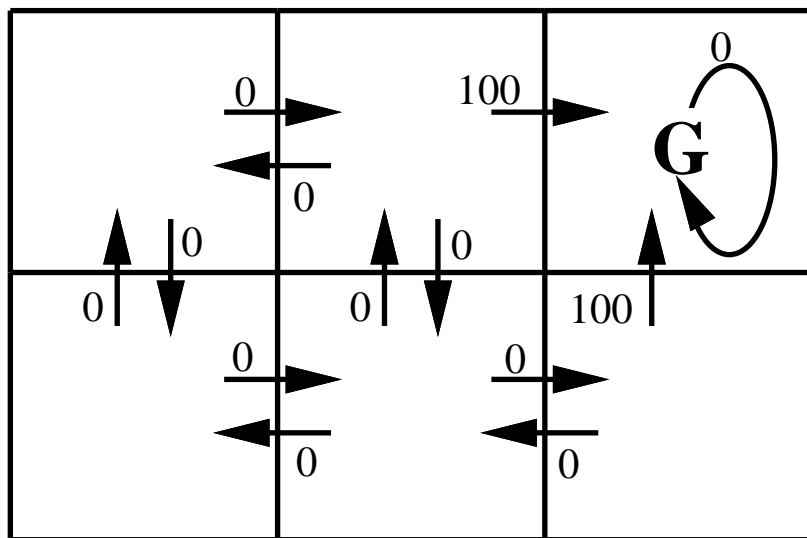
$$\pi^* \equiv \operatorname{argmax}_{\pi} V^\pi(s), (\forall s)$$

Note: $V^\pi(s)$ as above is the discounted cumulative reward.

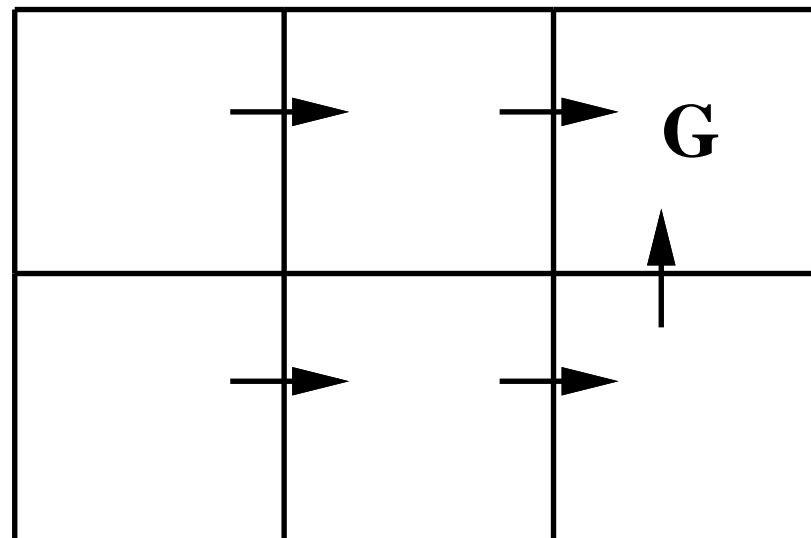
Other possible definitions for the total reward are:

- the final horizon reward: $\sum_{i=0}^h r_{t+i}$
- the average reward: $\lim_{h \rightarrow \infty} \frac{1}{h} \sum_{i=0}^h r_{t+i}$

Illustrating the basic concepts of Q-learning: A simple deterministic world



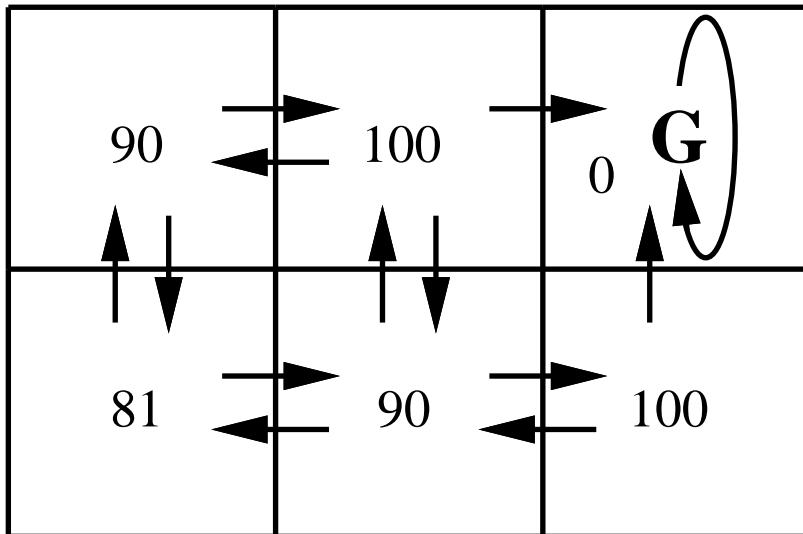
$r(s, a)$ (immed. reward) values



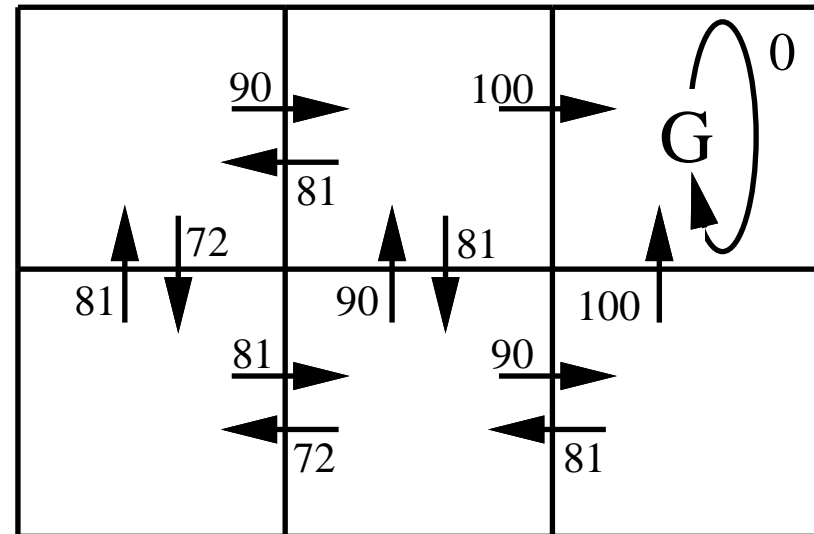
an optimal policy

Legend: state \equiv location, $\rightarrow \equiv$ action, G \equiv goal state
G is an “absorbing” state

Illustrating the basic concepts of Q -learning (Continued)



$V^*(s)$ values



$Q(s, a)$ values

How to learn them?

The V^{π^*} Function: the “value” of being in the state s

What to learn?

- We might try to make the agent learn the evaluation function V^{π^*} (which we write as V^*)
- It could then do a lookahead search to choose the best action from any state s because

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

Problem:

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

The Q Function [Watkins, 1989]

Let's **define** a new function, very similar to V^*

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

Note: If the agent can learn Q , then it will be able choose the optimal action even without knowing δ :

$$\begin{aligned}\pi^*(s) &= \operatorname{argmax}_a [r(s, a) + \gamma V^*(\delta(s, a))] \\ &= \operatorname{argmax}_a Q(s, a)\end{aligned}$$

Next: We will show the algorithm that the agent can use to learn the evaluation function Q

Training Rule to Learn Q

Note that Q and V^* are closely related: $V^*(s) = \max_{a'} Q(s, a')$.
That allows us to write Q recursively as

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

Let \hat{Q} denote the learner's current approximation to Q .
Consider the **training rule**

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

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

The Q Learning Algorithm

The Deterministic Case

Let us use a table $S \times A$ to store the \hat{Q} values.

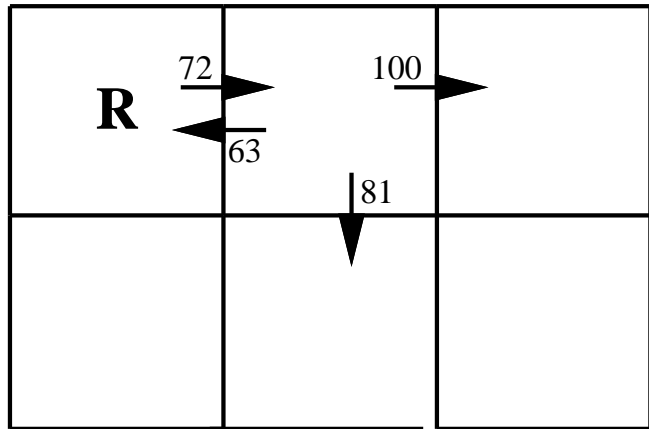
- For each s, a initialize the table entry $\hat{Q}(s, a) \leftarrow 0$
- Observe the current state s
- Do forever:
 - Select an action a and execute it
 - Receive immediate reward r
 - Observe the new state s'
 - Update the table entry for $\hat{Q}(s, a)$ as follows:

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

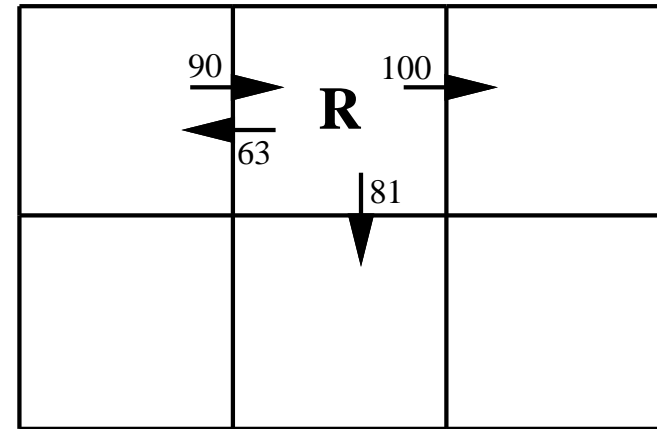
- $s \leftarrow s'$

Iteratively Updating \hat{Q}

Training as a series of episodes

Initial state: s_1

$\xrightarrow{a_{right}}$

Next state: s_2

$$\begin{aligned} \hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow 0 + 0.9 \max\{63, 81, 100\} \\ &\leftarrow 90 \end{aligned}$$

Convergence of Q Learning

The Theorem

Assuming that

1. the system is deterministic
2. $r(s, a)$ is bound, i.e $\exists c$ such that $|r(s, a)| \leq c$, for all s, a
3. actions are taken such that every pair $\langle s, a \rangle$ is visited infinitely often

then \hat{Q}_n converges to Q .

Convergence of Q Learning

The Proof

Define a **full interval** to be an interval during which each $\langle s, a \rangle$ is visited. We will show that **during each full interval the largest error in \hat{Q} table is reduced by the factor γ .**

Let the maximum error in \hat{Q}_n be denoted as $\Delta_n = \max_{s,a} |\hat{Q}_n(s, a) - Q(s, a)|$. 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

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

(We used the general fact that $|\max_a f_1(a) - \max_a f_2(a)| \leq \max_a |f_1(a) - f_2(a)|$.)

Therefore $|\hat{Q}_{n+1}(s, a) - Q(s, a)| \leq \gamma \Delta_n$, which implies $\Delta_{n+1} \leq \gamma \Delta_n$.

It follows that $\{\Delta\}_{n \in \mathbb{N}}$ is convergent (to 0) and so $\lim_{n \rightarrow \infty} Q^n(s, a) = Q(s, a)$.

Experimentation Strategies

16.

Let us introduce $K > 0$ and define

$$P(ai|s) = \frac{K^{\hat{Q}(s,a_i)}}{\sum_j K^{\hat{Q}(s,a_j)}}$$

If the agent choose actions according to probabilities $P(ai|s)$, then

for large values of K the agent can **exploit** what it has learned and seek actions it believes will maximize its reward;

for small values of K the agent will **explore** actions that do not currently have high \hat{Q} values.

Note: K may be varied with the number of iterations.

Updating Sequence — Improve Training Efficiency

1. Change the way \hat{Q} values are computed so that during one episode as many as possible values ($\hat{Q}(s, a)$) along the traversal paths get updated.
2. Store past state-action transitions along with the received reward and retrain on them periodically;
if a \hat{Q} predecessor state has a large update, then it is very possible that the current state get updated too.

The Q Algorithm — The Nondeterministic Case

When the reward and the next state are generated in a non-deterministic way,

the training rule $\hat{Q} \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$ would not converge.

We redefine V and Q by taking the expected values:

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

$$\begin{aligned} Q(s, a) &\equiv E[r(s, a) + \gamma V^*(\delta(s, a))] \equiv E[r(s, a)] + \gamma E[V^*(\delta(s, a))] \\ &\equiv E[r(s, a)] + \gamma \sum_{s'} P(s'|s, a) V^*(s') \\ &\equiv E[r(s, a)] + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a') \end{aligned}$$

Q Learning — Nondeterministic Case (Cont'd)

The training rule:

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

where α_n can be chosen as $\alpha_n = \frac{1}{1 + \text{visits}_n(s, a)}$

with $\text{visits}_n(s, a)$ being the number of times the pair $\langle s, a \rangle$ has been visited up to and including the n -th iteration.

Note: if $\alpha_n \rightarrow 1$ we get the deterministic form of updating (\hat{Q}).

Key idea: revisions to Q are made now more gradually than in the deterministic case.

Theorem [Watkins and Dayan, 1992]: \hat{Q} converges to Q .

Temporal Difference Learning

Q Learning reduces discrepancy between Q **successive** estimates:

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

Why not two, ..., or n steps?

In some settings, this is more convenient:

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

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

Temporal Difference Learning (TD) blends all of these:

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

Note that $Q^0(s_t, a_t) = Q^{(1)}(s_t, a_t)$;

as λ increases, more emphasis is put on more distant steps.

Temporal Difference Learning (Cont'd)

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

Equivalent expression:

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

The **TD(λ) algorithm**

- uses the 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 TD(λ) to learn Backgammon

Further Developments

- One can replace the \hat{Q} table e.g. with a neural net to learn Q values for an unseen pair $\langle s, a \rangle$ pair based on Q values already seen.

TD-Gammon uses does so, but in general the convergence is not ensured once a generalizer for learning the Q function is introduced.

- Handle the 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

When the δ and r functions are known, **dynamic programming** algorithms can be used to solve optimisation problems more efficiently than Q learning, and in general reinforcement learning.

See [[Kaelbling, 1996](#)] for a survey of such algorithms.