Introduction to Reinforcement Learning

Yingyu Liang

yliang@cs.wisc.edu

Computer Sciences Department

University of Wisconsin, Madison

[Based on slides from David Page, Mark Craven]

Goals for the lecture

you should understand the following concepts

- the reinforcement learning task
- Markov decision process
- value functions
- value iteration

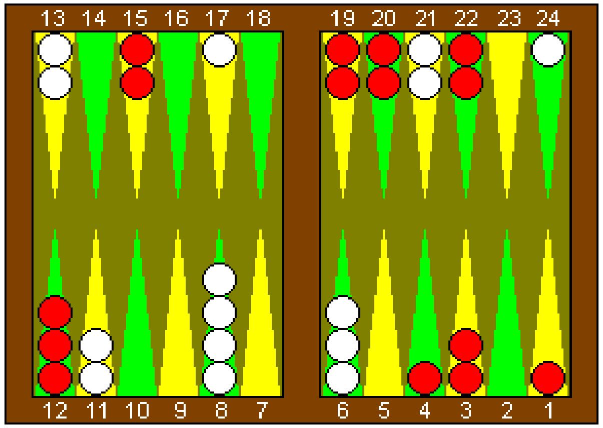
Reinforcement learning (RL)

Task of an agent embedded in an environment

repeat forever

1) sense world
2) reason
3) choose an action to perform
4) get feedback (usually reward = 0)
5) learn

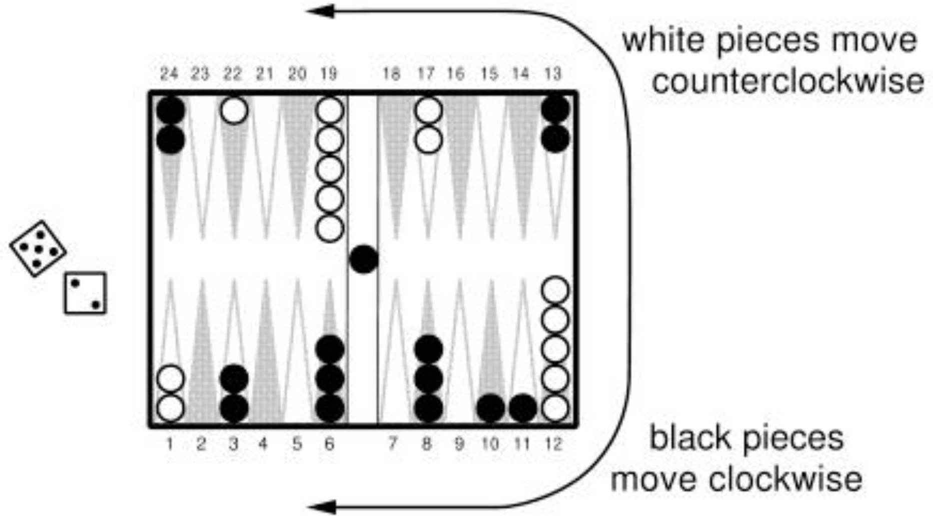
the environment may be the physical world or an artificial one



Example: RL Backgammon Player

[Tesauro, *CACM* 1995]

- world
  - 30 pieces, 24 locations
- actions
  - roll dice, e.g. 2, 5
  - move one piece 2
  - move one piece 5
- rewards
  - win, lose
- TD-Gammon 0.0
  - trained against itself (300,000 games)
  - as good as best previous BG computer program (also by Tesauro)
- TD-Gammon 2
  - beat human champion

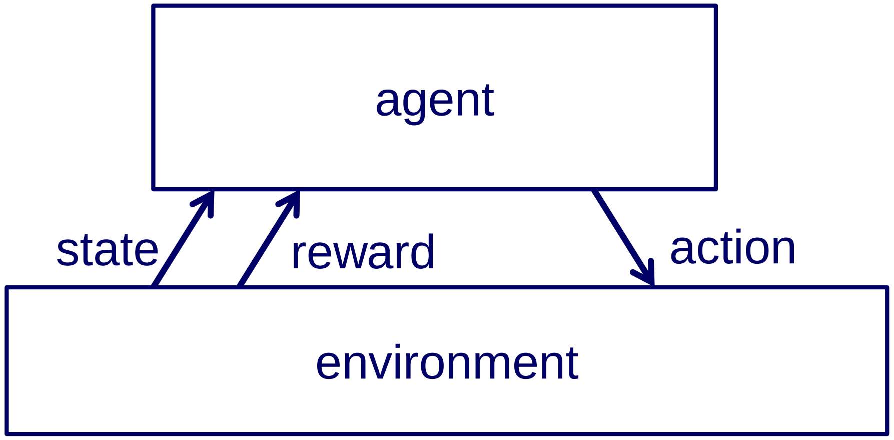

white pieces move counterclockwise
black pieces move clockwise
24 23 22 21 20 19 18 17 16 15 14 13
1 2 3 4 5 6 7 8 9 10 11 12

Example: AlphaGo
[Nature, 2017]

- world
  - 19x19 locations
- actions
  - Put one stone on some empty location
- rewards
  - win, lose
- 2016 beats World Champion Lee Sedol by 4-1
- Subsequent system (AlphaGo Master/zero) shows superior performance than humans
- Trained by supervised learning + reinforcement learning

Reinforcement learning

- set of states S
- set of actions A
- at each time t, agent observes state $s_t \in S$ then chooses action $a_t \in A$
- then receives reward r_t and changes to state s_{t+1}



$$\begin{tikzcd} s_0 \arrow[r, "a_0", "r_0"'] & s_1 \arrow[r, "a_1", "r_1"'] & s_2 \arrow[r, "a_2", "r_2"'] & {} \end{tikzcd}$$

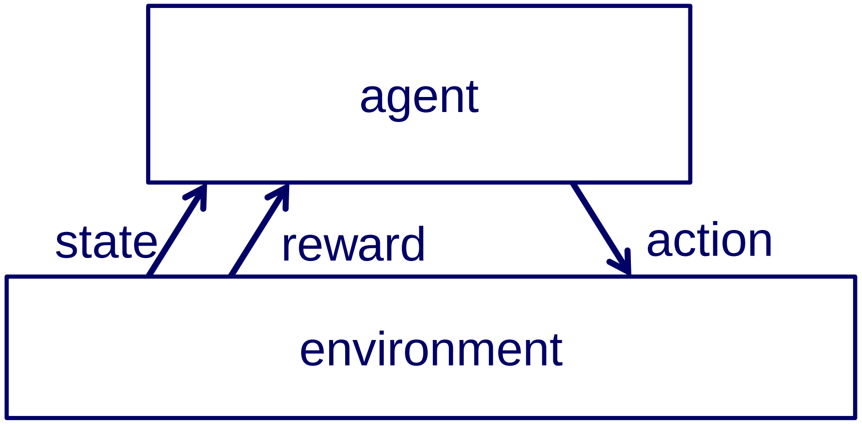
Reinforcement learning as a Markov decision process (MDP)

- Markov assumption

$$P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = P(s_{t+1} \mid s_t, a_t)$$

- also assume reward is Markovian

$$P(r_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = P(r_{t+1} \mid s_t, a_t)$$



$$\begin{tikzcd} s_0 \arrow[r, "a_0", "r_0"'] & s_1 \arrow[r, "a_1", "r_1"'] & s_2 \arrow[r, "a_2", "r_2"'] & {} \end{tikzcd}$$

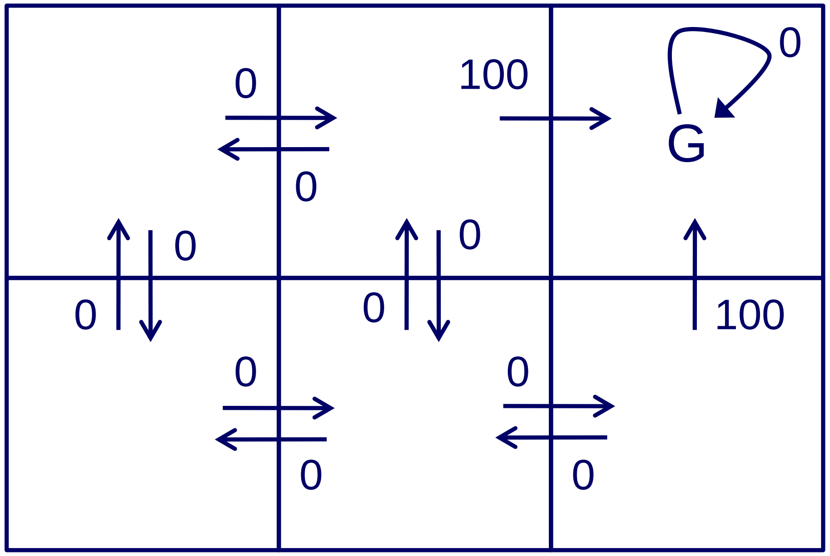
Goal: learn a policy $\pi : S \rightarrow A$ for choosing actions that maximizes

$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots] \quad \text{where } 0 \le \gamma < 1$$

for every possible starting state s_0

Reinforcement learning task

- Suppose we want to learn a control policy $\pi : S \rightarrow A$ that maximizes $\sum_{t=0}^{\infty} \gamma^t E[r_t]$ from every state $s \in S$



each arrow represents an action a and the associated number represents deterministic reward $r(s, a)$

VALUE FUNCTION

Value function for a policy

- given a policy $\pi : S \rightarrow A$ define

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^{t} E[r_t]$$

assuming action sequence chosen according to π starting at state s

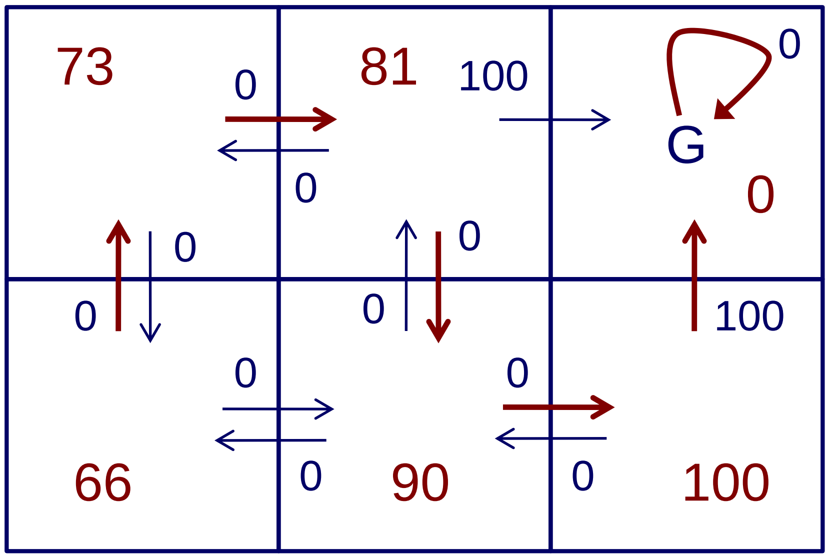
- we want the optimal policy π^* where

$$p^* = \arg\max_p V^p(s) \quad \text{for all } s$$

we'll denote the value function for this optimal policy as $V^*(s)$

Value function for a policy π

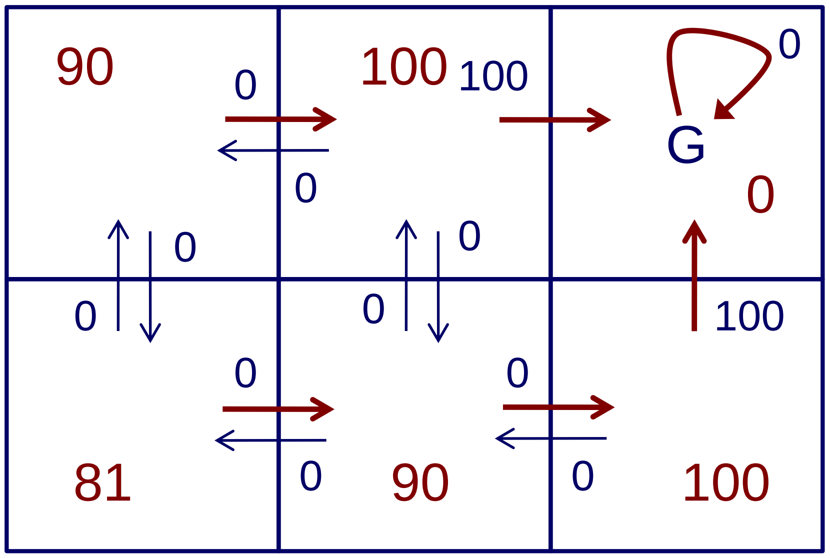
- Suppose π is shown by red arrows, $\gamma = 0.9$



$V^{\pi}(s)$ values are shown in red

Value function for an optimal policy π^*

- Suppose π^* is shown by red arrows, $\gamma = 0.9$



$V^*(s)$ values are shown in red

Using a value function

If we know $V^*(s)$, $r(s_t, a)$, and $P(s_t \mid s_{t-1}, a_{t-1})$
we can compute $\pi^*(s)$

$$\pi^*(s_t) = \arg\max_{a \in A} \left[r(s_t, a) + \gamma \sum_{s \in S} P(s_{t+1} = s \mid s_t, a) V^*(s) \right]$$

Value iteration for learning $V^*(s)$

```
initialize $V(s)$ arbitrarily
loop until policy good enough
{
    loop for $s \in S$
    {
        loop for $a \in A$
        {
            $Q(s,a) \leftarrow r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s')$
        }
        $V(s) \leftarrow \max_a Q(s,a)$
    }
}
```

Value iteration for learning $V^*(s)$

- $V(s)$ converges to $V^*(s)$
- works even if we randomly traverse environment instead of looping through each state and action methodically
  - but we must visit each state infinitely often
- implication: we can do online learning as an agent roams around its environment
- assumes we have a model of the world: i.e. know $P(\mathrm{s}_\mathrm{t} \mid s_{t-1}, a_{t-1})$
- What if we don't?

Q-LEARNING

Q functions

define a new function, closely related to $V*$

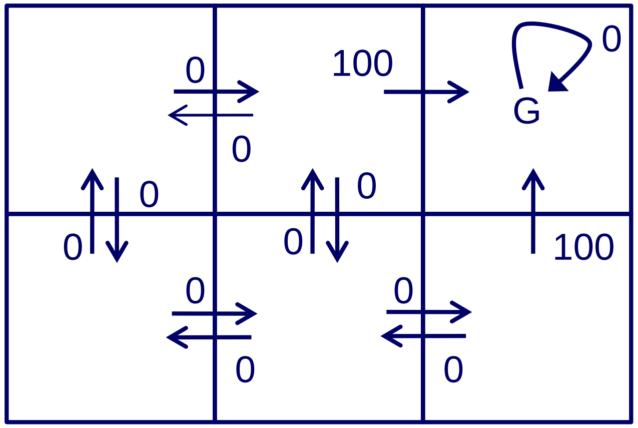
$$Q(s,a) \leftarrow E\left[r(s,a)\right] + \gamma E_{s'|s,a}\left[V^*(s')\right]$$

if agent knows $Q(s, a)$, it can choose optimal action without knowing $P(s' \mid s, a)$

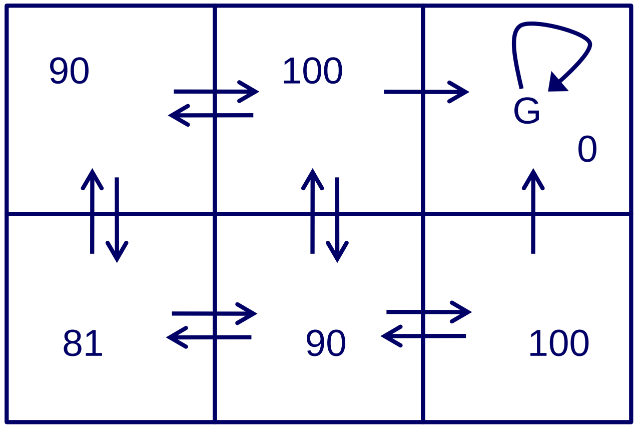
$$\pi^*(s) \leftarrow \arg\max_a Q(s,a) \qquad V^*(s) \leftarrow \max_a Q(s,a)$$

and it can learn $Q(s, a)$ without knowing $P(s' \mid s, a)$

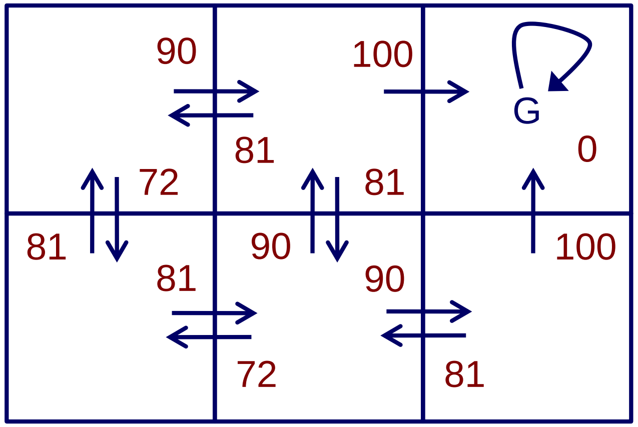
Q values



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



$V^*(s)$ values

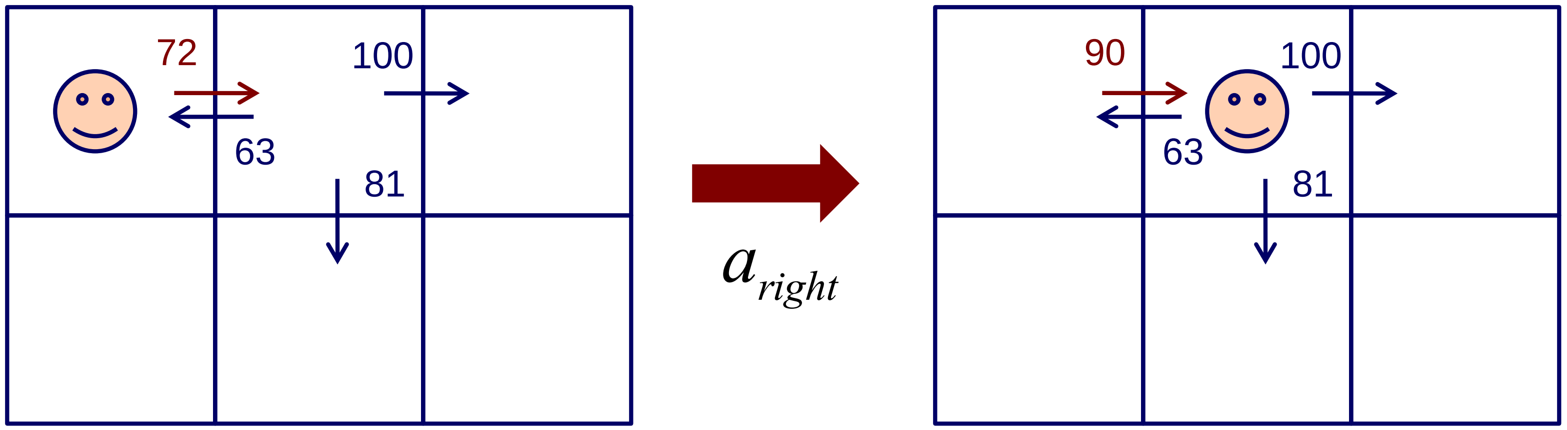


$Q(s, a)$ values

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

Updating Q



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

Q's vs. V's

$$\begin{tikzcd}
\boxed{V} \arrow[r, "Q"] \arrow[d, "Q"'] & \boxed{V} \\
\boxed{V} &
\end{tikzcd}$$

- Which action do we choose when we're in a given state?
- V's (model-based)
  - need to have a 'next state' function to generate all possible states
  - choose next state with highest V value.
- Q's (model-free)
  - need only know which actions are legal
  - generally choose next state with highest Q value.

Exploration vs. Exploitation

- in order to learn about better alternatives, we shouldn't always follow the current policy (exploitation)
- sometimes, we should select random actions (exploration)
- one way to do this: select actions probabilistically according to:

$$P(a_i \mid s) = \frac{c^{\hat{Q}(s,a_i)}}{\sum_j c^{\hat{Q}(s,a_j)}}$$

where $c > 0$ is a constant that determines how strongly selection favors actions with higher Q values