Reinforcement Learning

CS 760@UW-Madison

Goals for the lecture

you should understand the following concepts

- the reinforcement learning task
- Markov decision process
- value functions
- Q functions
- value iteration
- Q learning
- exploration vs. exploitation tradeoff
- compact representations of Q functions
- reinforcement learning example

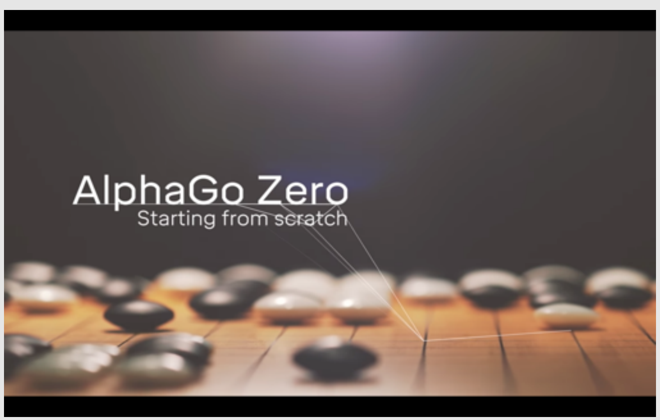
Reinforcement learning (RL)

Task of an **agent** embedded in an **environment**

repeat forever

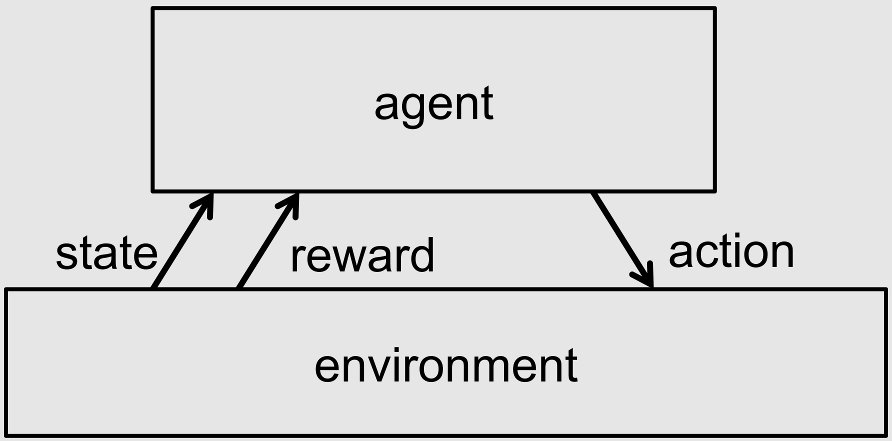
1) sense world
2) reason
3) choose an action to perform
4) get feedback (usually a reward in $\mathbb{R}$)
5) learn

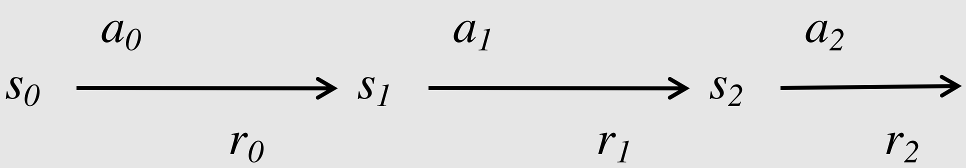
the **environment** may be the physical world or an artificial one



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 moves to state s_{t+1}.





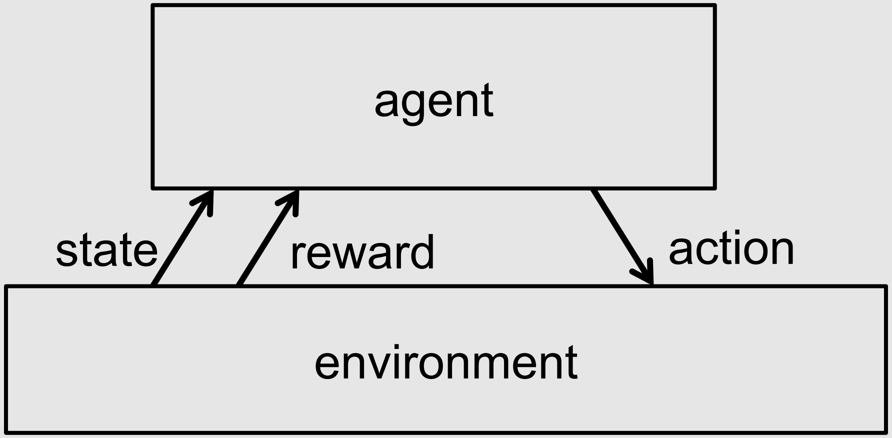
RL as Markov decision process (MDP)

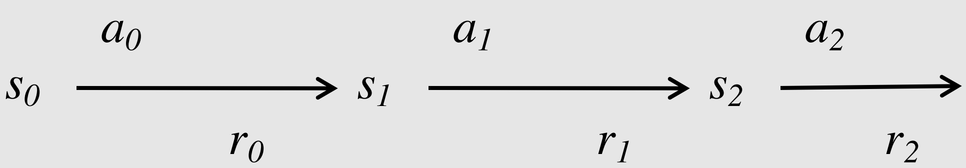
- Markov assumption

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

- also assume reward is Markovian

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





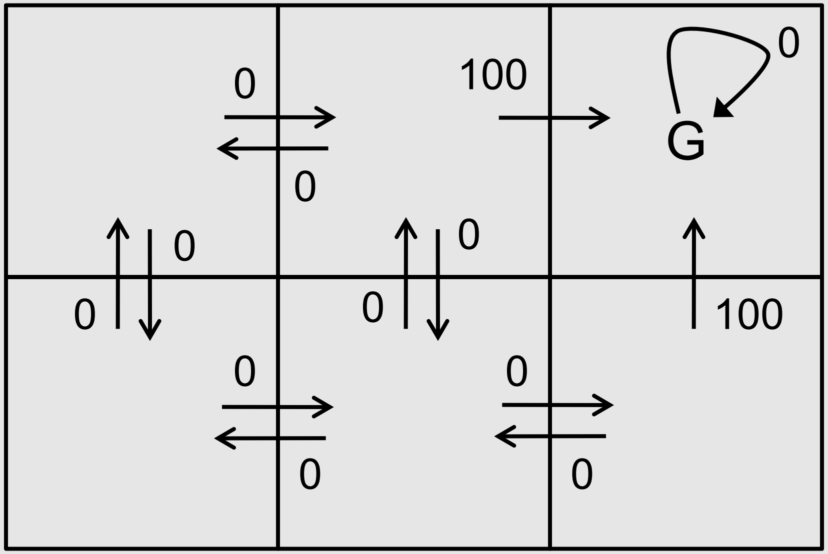
Goal: learn a policy $\pi : S \rightarrow A$ for choosing actions that maximizes

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

for every possible starting state s_0

The grid world

- As a running example:



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

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

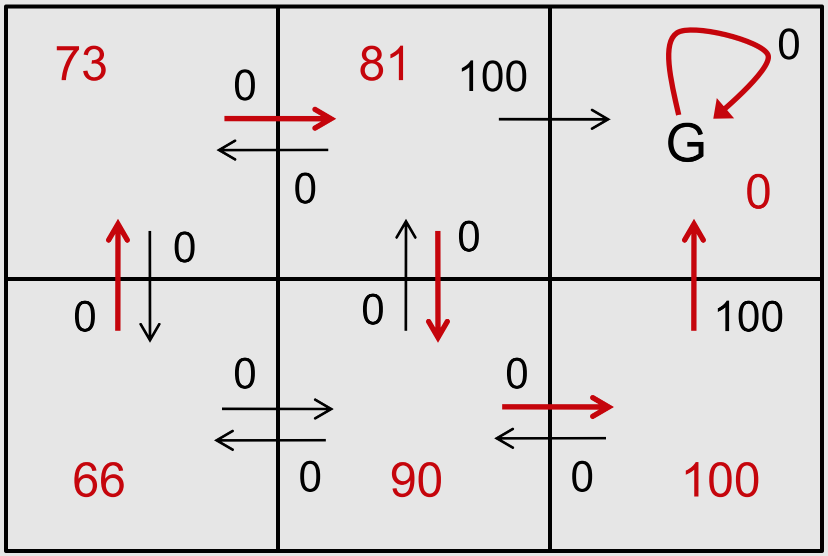
- we want the optimal policy π^* where

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

- 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, $\gamma = 0.9$



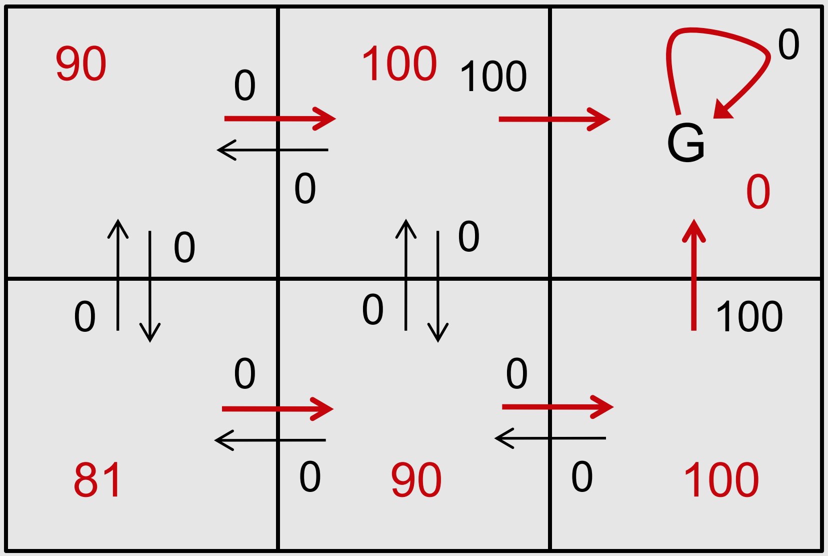
$V^{\pi}(s)$ values are shown in red

- **The Bellman's equation** (for deterministic transition):

$$V^{\pi}(s) = r(s, \pi(s)) + \gamma V^{\pi}(s')$$

Value function for an optimal policy π^*

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



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

- **The Bellman's equation** (for deterministic transition):

$$V^\pi(s) = r(s, \pi(s)) + \gamma V^\pi(s')$$

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 ∈ S
    {
        loop for a ∈ A
        {
            Q(s,a) ← r(s,a) + γ Σ_{s'∈S} P(s'|s,a) V(s')
        }
        V(s) ← max_a Q(s,a)
    }
}
```

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 $P(s_t \mid s_{t-1}, a_{t-1})$
- What if we don't?

Using a Q function

define a new function, closely related to V^*

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

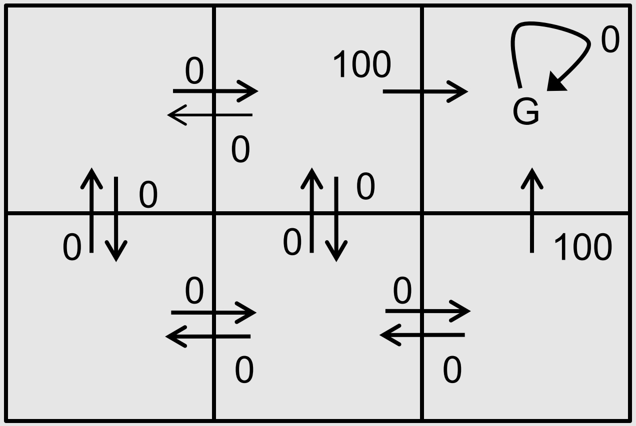
$$Q(s,a) \leftarrow E\left[r(s,a)\right] + \gamma E_{s'|s,a}\left[V^*(s')\right]$$

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

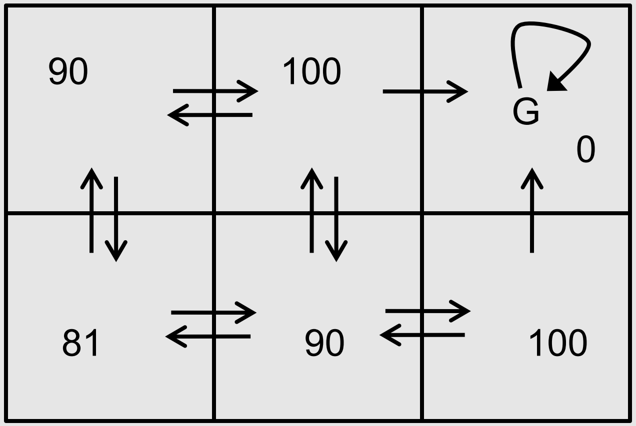
$$\pi^*(s) \leftarrow \arg\max_a Q(s,a) \qquad V^*(s) \leftarrow \max_a Q(s,a)$$

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

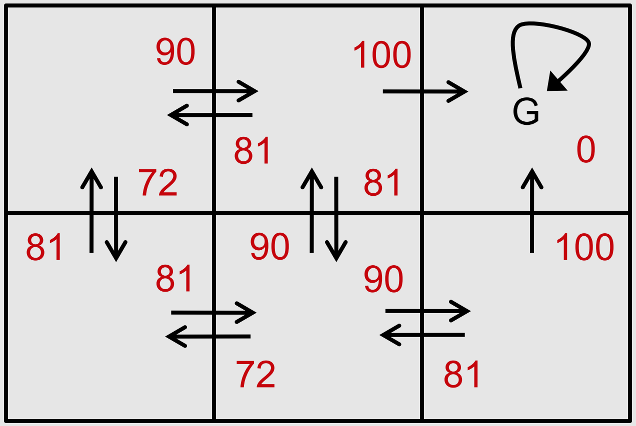
Q values



$r(s, a)$ (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

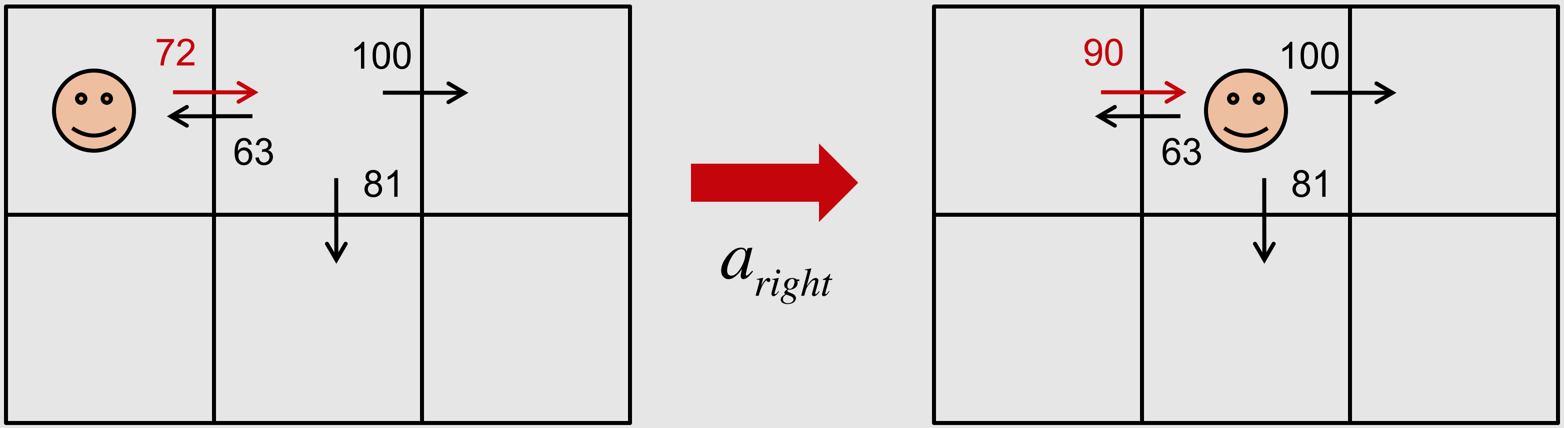
observe the new state s'

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 learning for *nondeterministic* 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

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

- $s \leftarrow s'$

where α_n is a parameter dependent on the number of visits to the given (s, a) pair

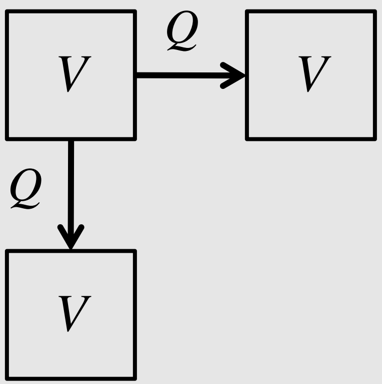
$$\alpha_n = \frac{1}{1 + \text{visits}_n(s,a)}$$

Convergence of Q learning

- Q learning will converge to the correct Q function
  - in the deterministic case
  - in the nondeterministic case (using the update rule just presented)

- in practice it is likely to take many, many iterations

Q's vs. V's



- 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

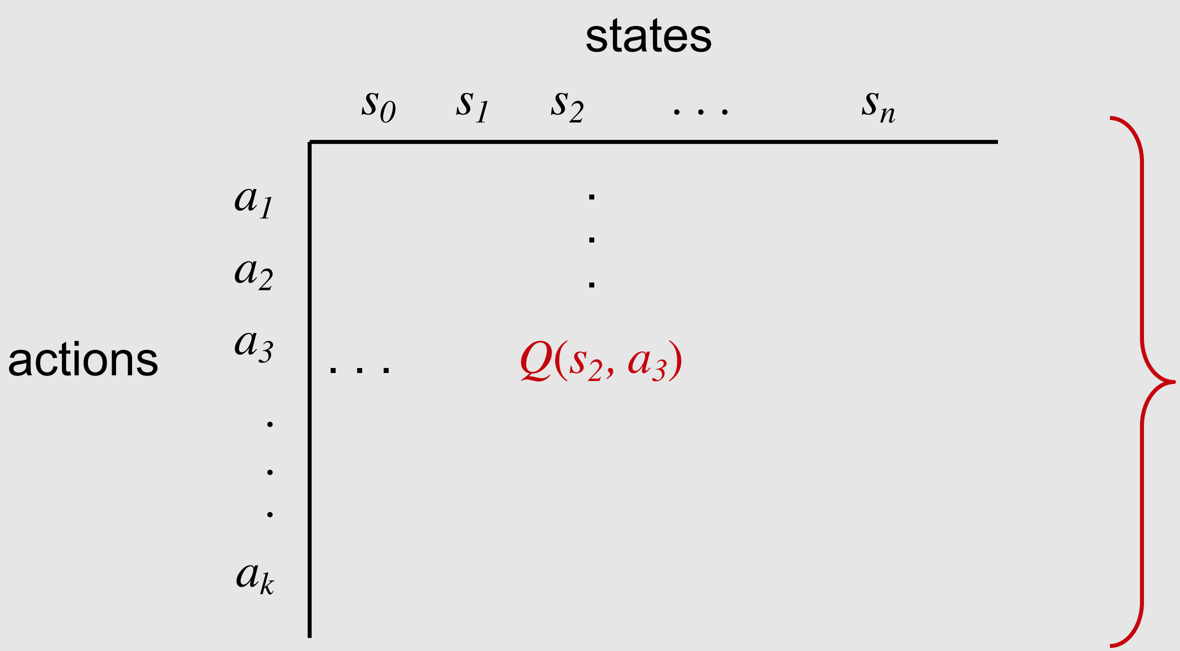
- in order to learn about better alternatives, we shouldn't always follow the current policy (exploitation)
- sometimes, we should select random actions (exploration)
- one way to do this: select actions probabilistically according to:

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

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

Q learning with a table

As described so far, Q learning entails filling in a huge table

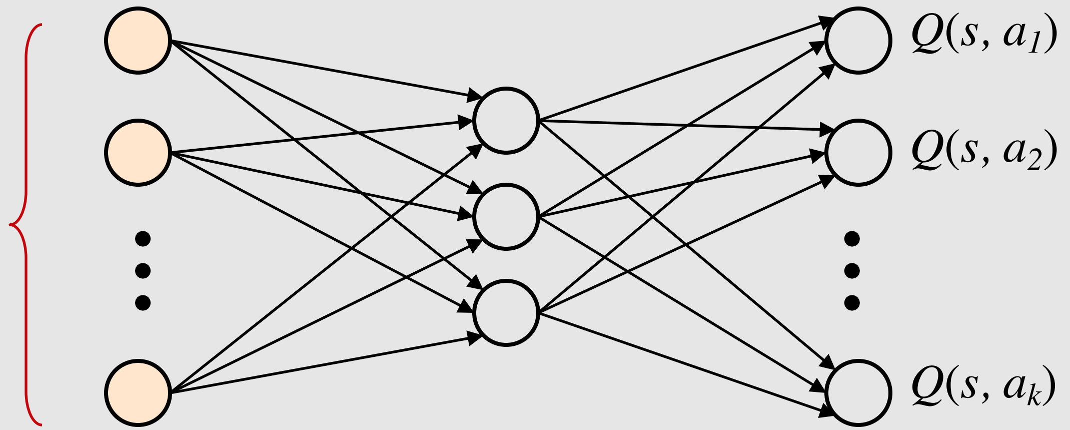


A table is a very <u>verbose way to</u> represent a function

Representing Q functions more compactly

We can use some other function representation (e.g. a neural net) to compactly encode a substitute for the big table

encoding of the state (s)



each input unit encodes a property of the state (e.g., a sensor value)

Why use a compact Q function?

1. Full Q table may not fit in memory for realistic problems
2. Can generalize across states, thereby speeding up convergence

   i.e. one instance 'fills' many cells in the Q table

Notes

1. When generalizing across states, cannot use $\alpha=1$
2. Convergence proofs only apply to Q tables
3. Some work on bounding errors caused by using compact representations (e.g. Singh & Yee, *Machine Learning* 1994)

Q tables vs. Q nets

Given: 100 Boolean-valued features

10 possible actions

Size of Q table

10×2^{100} entries

Size of Q net (assume 100 hidden units)

$\underbrace{100 \times 100}_{\text{weights between inputs and HU's}} + \underbrace{100 \times 10}_{\text{weights between HU's and outputs}} = 11{,}000$ weights

weights between inputs and HU's

weights between HU's and outputs

Q learning with function approximation

1. measure sensors, sense state s_0
2. predict $\hat{Q}_n(s_0, a)$ for each action a
3. select action a to take (with randomization to ensure exploration)
4. apply action a in the real world
5. sense new state s_1 and immediate reward r
6. calculate action a' that maximizes $\hat{Q}_n(s_1, a')$
7. train with new instance

$$\boldsymbol{x} = s_0$$

$$y \leftarrow (1-\alpha)\hat{Q}(s_0, a) + \alpha\left[r + \gamma \max_{a'} \hat{Q}(s_1, a')\right]$$

Calculate Q-value you would have put into Q-table, and use it as the training label

ML example: reinforcement learning to control an autonomous helicopter

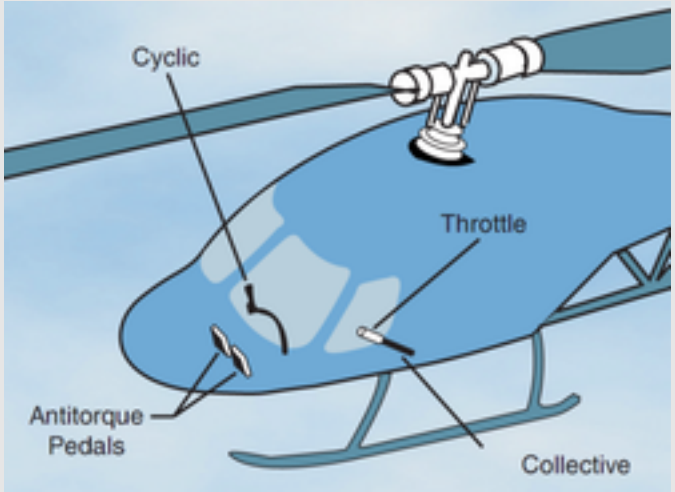
video of Stanford University autonomous helicopter from http://heli.stanford.edu/

Stanford autonomous helicopter

sensing the helicopter's state

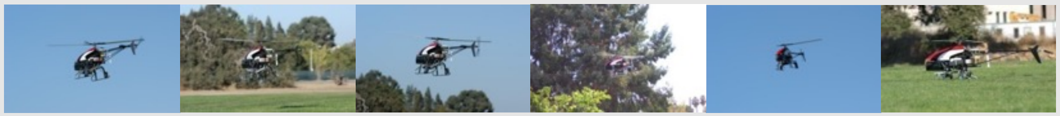
- orientation sensor
  - accelerometer
  - rate gyro
  - magnetometer
- GPS receiver ("2cm accuracy as long as its antenna is pointing towards the sky")
- ground-based cameras

actions to control the helicopter

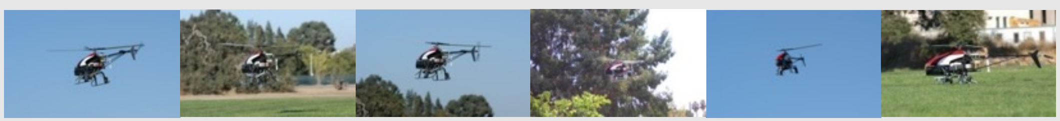


Experimental setup for helicopter

1. Expert pilot demonstrates the airshow several times



2. Learn a reward function based on desired trajectory
3. Learn a dynamics model
4. Find the optimal control policy for learned reward and dynamics model
5. Autonomously fly the airshow



6. Learn an improved dynamics model. Go back to step 4

Learning dynamics model $P(s_{t+1} \mid s_t,\ a)$

- state represented by helicopter's

position (x, y, z)

velocity $(\dot{x}, \dot{y}, \dot{z})$

angular velocity $(\omega_x, \omega_y, \omega_z)$

- action represented by manipulations of 4 controls

$$(u_1, u_2, u_3, u_4)$$

- dynamics model predicts accelerations as a function of current state and actions
- accelerations are integrated to compute the predicted next state

Learning dynamics model $P(s_{t+1} | s_t, a)$

dynamics model

$$\ddot{x}^b = A_x \dot{x}^b + g_x^b + w_x,$$
$$\ddot{y}^b = A_y \dot{y}^b + g_y^b + D_0 + w_y,$$
$$\ddot{z}^b = A_z \dot{z}^b + g_z^b + C_4 u_4 + D_4 + w_z,$$
$$\dot{\omega}_x^b = B_x \omega_x^b + C_1 u_1 + D_1 + w_{\omega_x},$$
$$\dot{\omega}_y^b = B_y \omega_y^b + C_2 u_2 + D_2 + w_{\omega_y},$$
$$\dot{\omega}_z^b = B_z \omega_z^b + C_3 u_3 + D_3 + w_{\omega_z}.$$

- A, B, C, D represent model parameters
- g represents gravity vector
- w's are random variables representing noise and unmodeled effects

- linear regression task!

Learning a desired trajectory

- repeated expert demonstrations are often suboptimal in different ways
- given a set of M demonstrated trajectories

$$y_j^k = \begin{bmatrix} s_j^k \\ u_j^k \end{bmatrix} \quad \text{for } j = 0,\ldots,N-1, k = 0,\ldots,M-1$$

action on j^{th} step of trajectory k

state on j^{th} step of trajectory k

- try to infer the implicit desired trajectory

$$z_t = \begin{bmatrix} s_t^* \\ u_t^* \end{bmatrix} \quad \text{for } t = 0,\ldots,H$$

Learning a desired trajectory

colored lines: demonstrations of two loops
black line: inferred trajectory

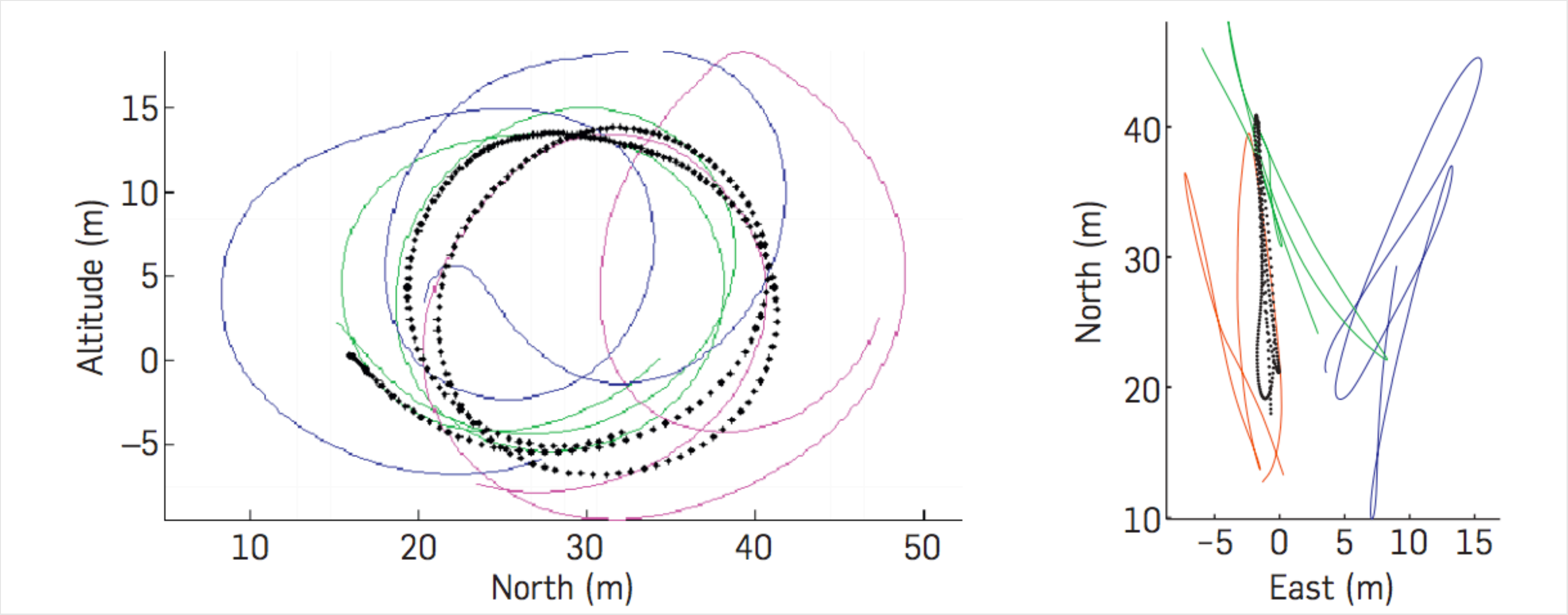


Figure from Coates et al., *CACM* 2009

Learning reward function

- EM is used to infer desired trajectory from set of demonstrated trajectories

- The reward function is based on deviations from the desired trajectory

Finding the optimal control policy

- finding the control policy is a reinforcement learning task

$$\pi^* \leftarrow \arg\max_{\pi} E\left[\sum_t r(s_t, a) \mid \pi\right]$$

- RL learning methods described earlier don't quite apply because state and action spaces are both continuous
- A special type of Markov decision process in which the optimal policy can be found efficiently
  - reward is represented as a linear function of state and action vectors
  - next state is represented as a linear function of current state and action vectors
- They use an iterative approach that finds an approximate solution because the reward function used is quadratic

THANK YOU

Some of the slides in these lectures have been adapted/borrowed from materials developed by Yingyu Liang, Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, and Pedro Domingos.