

Reinforcement Learning Part 2

Yingyu Liang
Computer Sciences 760
Fall 2017

<http://pages.cs.wisc.edu/~yliang/cs760/>

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

Goals for the lecture

you should understand the following concepts

- value functions and value iteration (review)
- Q functions and Q learning (review)
- exploration vs. exploitation tradeoff
- compact representations of Q functions
- reinforcement learning example

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

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

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

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

initialize $V(s)$ arbitrarily

loop until policy good enough

{

 loop for $s \in S$

 {

 loop for $a \in A$

 {

$$Q(s, a) \leftarrow r(s, a) + \gamma \sum_{s' \in S} P(s' | s, a) V(s')$$

 }

$$V(s) \leftarrow \max_a Q(s, a)$$

 }

}

Q learning

define a new function, closely related to V^*

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

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

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

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

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

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

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

$s \leftarrow s'$

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

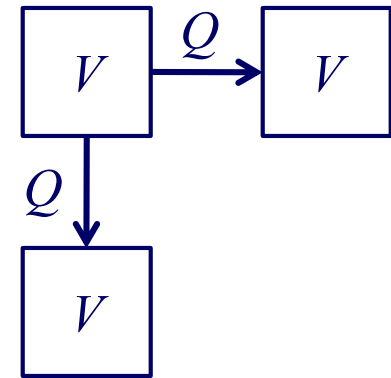
$$\hat{Q}_n(s, a) \leftarrow (1 - \alpha_n)\hat{Q}_{n-1}(s, a) + \alpha_n \left[r + \gamma \max_{a'} \hat{Q}_{n-1}(s', a') \right]$$

$s \leftarrow s'$

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

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

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:

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

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

Q learning with a table

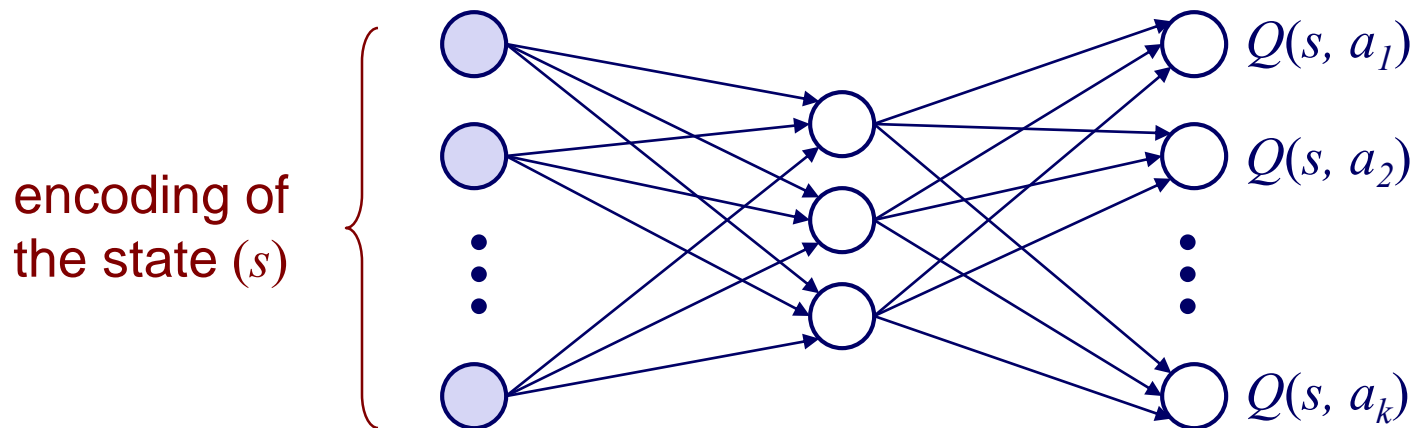
As described so far, Q learning entails filling in a huge table

		states				
		s_0	s_1	s_2	\dots	s_n
actions	a_1			.		
	a_2			.		
	a_3	\dots		$Q(s_2, a_3)$		
	.					
	.					
	a_k					

A table is a very verbose way to 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



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

or could have one net
for each possible action

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)

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

weights between
inputs and HU's

weights between
HU's and outputs

Representing Q functions more compactly

- we can use other regression methods to represent Q functions
 - k -NN
 - regression trees
 - support vector regression
 - etc.

Q learning with function approximation

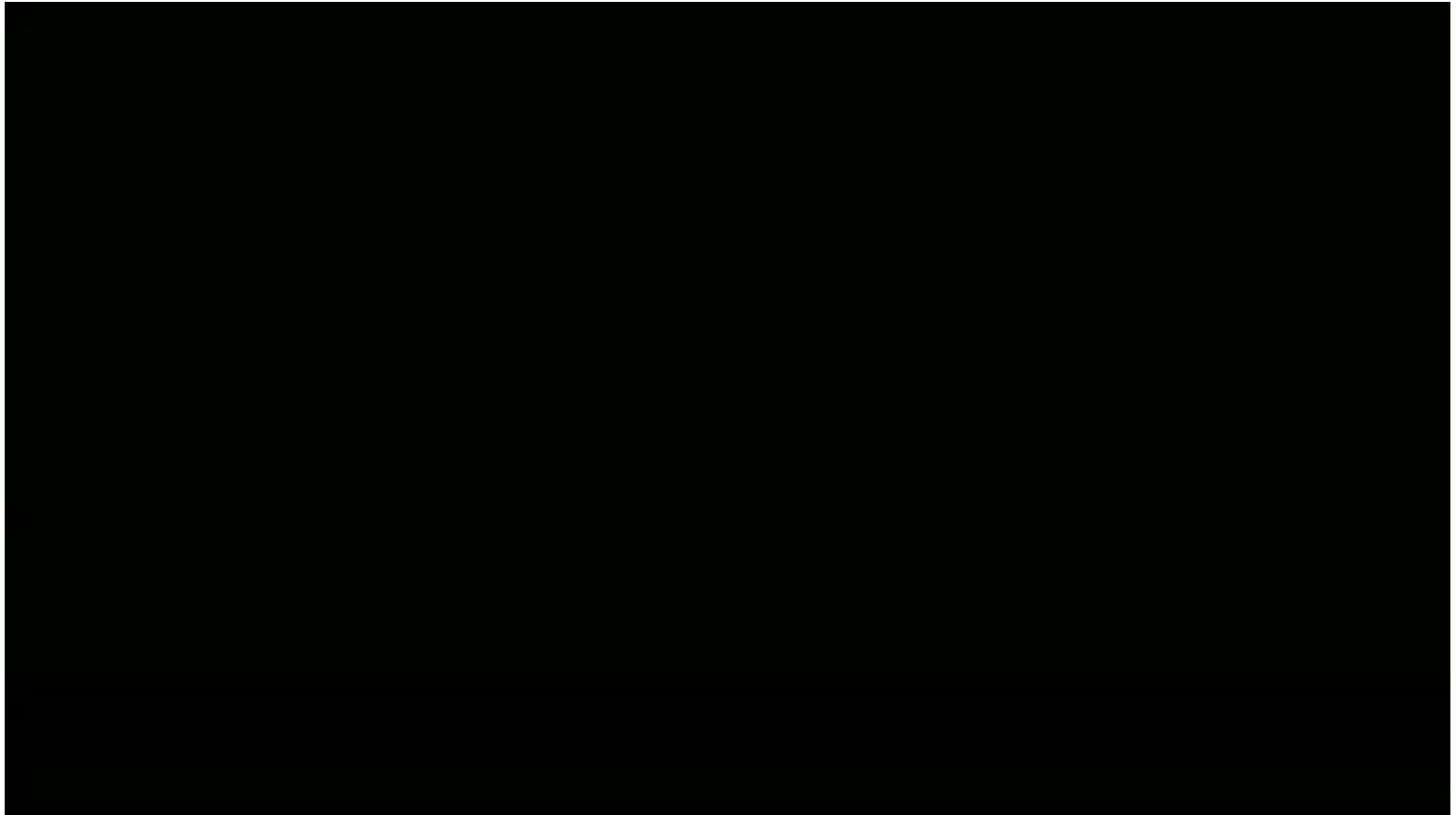
1. measure sensors, sense state s_0
2. predict $\hat{Q}_n(s_0, a)$ for each action a
3. select action a to take (with randomization to ensure exploration)
4. apply action a in the real world
5. sense new state s_1 and immediate reward r
6. calculate action a' that maximizes $\hat{Q}_n(s_1, a')$
7. train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

$$\mathbf{x} = s_0$$

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

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

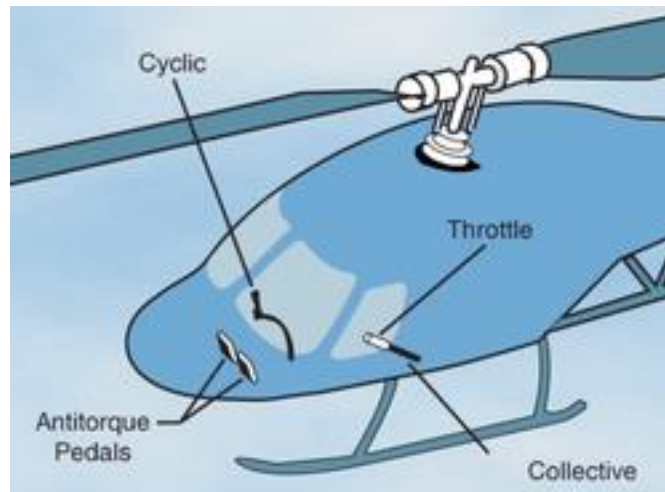


Stanford autonomous helicopter

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

- state represented by helicopter's

position (x, y, z)

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

angular velocity (W_x, W_y, W_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

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

action on j^{th} step of trajectory k

state on j^{th} step of trajectory k

- try to infer the implicit desired trajectory

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

Learning a desired trajectory

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

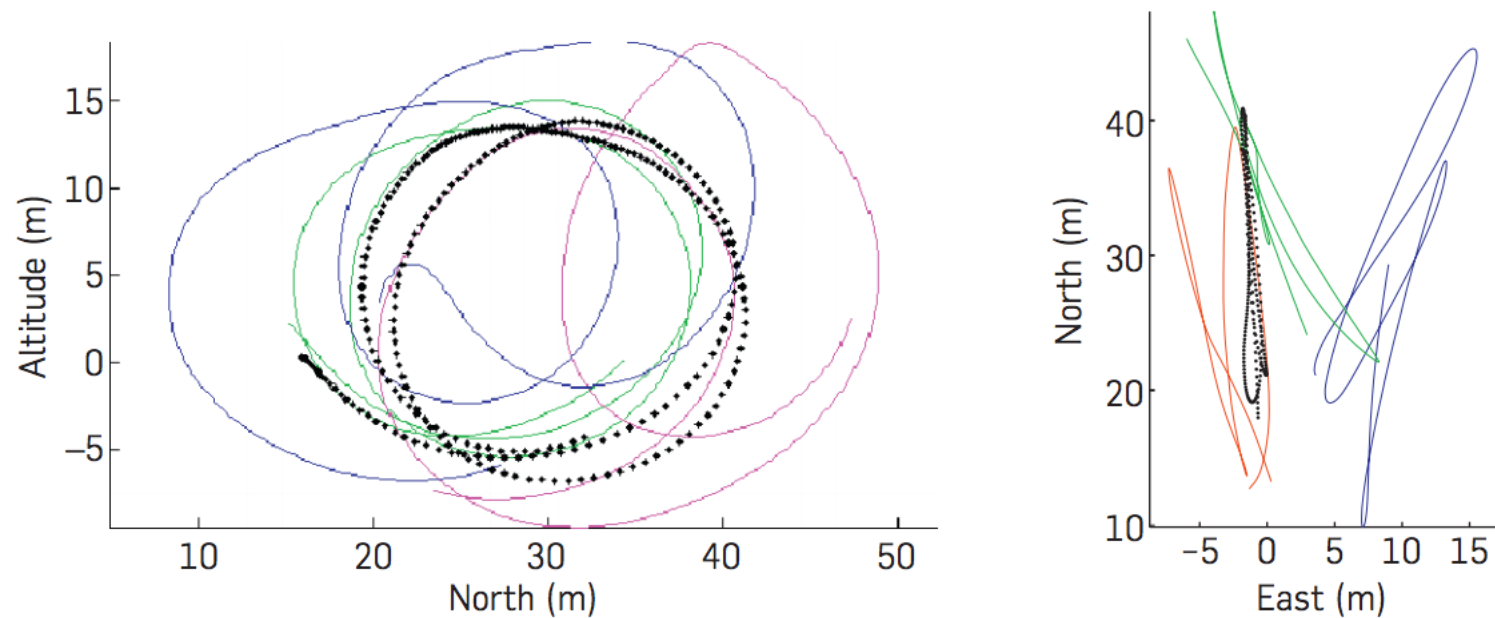


Figure from Coates et al., *CACM* 2009

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