6ELEN018W - Applied Robotics
Lecture 9: Robot Control - Intelligent Control Algorithms - Part II

Dr Dimitris C. Dracopoulos

The Most General and Challenging Control Problem for Robots

Robots need to operate:

- In unknown environments (be adaptive and including operation with sensor noise)
- Cope with high non-linear dynamics when interacting with other systems
- Be reconfigurable (in the case where part of the robot gets damaged and its dynamics change)

Markov Models

To formulate the general control problem for a robot, Markov models are useful.
A finite state Markov chain (stochastic finite state machine) can be defined:

- States: $s \in \{1, \ldots, m\}$, where m is finite.
- Starting state s_0: may be fixed or drawn from some a priori distribution $P_0(s_0)$.
- Transitions (dynamics): how the system moves from the current state s_t to the next state s_{t+1}.
- The transitions satisfy the first order Markov property:

$$P(s_{t+1}|s_t, s_{t-1}, \ldots, s_0) = P_1(s_{t+1}|s_t) \tag{1}$$

Markov Chains (cont'd)

Markov chains define a stochastic system which generates a sequence of states:

$$s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow \dots$$

where s_0 is drawn from $P_0(s_0)$ and each s_{t+1} from one step transition probabilities $P_1(s_{t+1}|s_t)$.

- A Markov chain can be represented as a state transition diagram.

Transition Probabilities

The conditional probability p_{ij} is defined as the probability that a system which occupies state i, will occupy state j after its next transition.

- Since the system must be in some state after its next transition:

$$\sum_{j=1}^{N} p_{ij} = 1 \tag{2}$$

- Since p_{ij} are probabilities:

$$0 \leq p_{ij} \leq 1 \tag{3}$$

Example - The Robot Maker

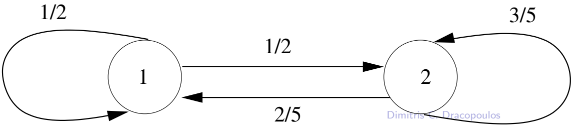
A robot maker is involved in the novelty robot business. He may be in either of two states:

1. The robot he is currently producing has found great favour with the public.
2. The robot is out of favour.

Transition probabilities:

- If in first state 50% chance of remaining in state 1, and 50% chance of unfortunate move to state 2 at following week.
- While in state 2, he experiments with new robots and he may return to state 1 after a week with probability $\frac{2}{5}$, or remain unprofitable in state 2 with probability $\frac{3}{5}$.

$$P = [p_{ij}] = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{2}{5} & \frac{3}{5} \end{bmatrix} \tag{4}$$



Markov Chain Problems

- *Prediction*: Probabilities that the system will be in state s_k after n transitions, given that at $n = 0$ is it in a known state.
- *Estimation*: Calculation of transition probabilities given some observed sequences of state transitions.

The Prediction Problem

Example: What is the probability that the robot maker will be in state 1 after n weeks, given that he is in state 1 at the beginning of the n-week period?
Define $\pi_i(n)$ as the probability that the system will occupy state i after n transitions, if its state at $n = 0$ is known.
Then:

$$\sum_{i=1}^{N} \pi_i(n) = 1 \tag{5}$$

$$\pi_j(n+1) = \sum_{i=1}^{N} \pi_i(n) p_{ij} \qquad n = 0, 1, 2, \ldots \tag{6}$$

The Prediction Problem (cont'd)

Define a row vector of state probabilities $\pi(n)$ with components $\pi_i(n)$.
Then:

$$\pi(n+1) = \pi(n)P \qquad n = 0, 1, 2, \ldots \tag{7}$$

Now:

$$\begin{aligned} \pi(1) &= \pi(0)P \\ \pi(2) &= \pi(1)P = \pi(0)P^2 \\ \pi(3) &= \pi(2)P = \pi(0)P^3 \end{aligned} \tag{8}$$

In general:

$$\pi(n) = \pi(0)P^n \qquad n = 0, 1, 2, \ldots \tag{9}$$

Application to the Robot Maker Example

Assume that the robot maker starts with a successful robot, then $\pi_1(0) = 1$, $\pi_2(0) = 0$.

$$\pi(1) = \pi(0)P = [1 \;\; 0] \left[\begin{array}{cc} \frac{1}{2} & \frac{1}{2} \\ \frac{2}{5} & \frac{3}{5} \end{array} \right] = \left[\frac{1}{2} \;\; \frac{1}{2} \right]$$

After 1 week the robot maker is equally likely to be successful or unsuccessful.
After 2 weeks:

$$\pi(2) = \pi(1)P = [\frac{1}{2} \;\; \frac{1}{2}] \left[\begin{array}{cc} \frac{1}{2} & \frac{1}{2} \\ \frac{2}{5} & \frac{3}{5} \end{array} \right] = \left[\frac{9}{20} \;\; \frac{11}{20} \right]$$

so that the robot maker is slightly more likely to be unsuccessful.

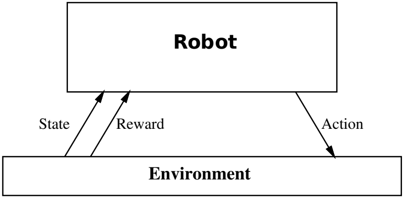
Example: Successive State Probabilities Starting with a Successful Robot

n	0	1	2	3	4	5	...
$\pi_1(n)$	1	0.5	0.45	0.445	0.4445	0.44445	...
$\pi_2(n)$	0	0.5	0.55	0.555	0.5555	0.55555	...

As n becomes very large:

- $\pi_1(n)$ approaches $\frac{4}{9}$
- $\pi_2(n)$ approaches $\frac{5}{9}$

The Reinforcement Learning Problem for a Robot



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

Goal: Learn to choose actions that maximise:

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots,$$

where $0 \leq \gamma < 1$

Robot's Learning Task

Execute actions in environment, observe results, and

- learn action policy $\pi : S \longrightarrow A$ that maximises

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

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

Value Function

How can a robot calculate the optimum action at each state?

- What if each state s_i has a value associated with it, measuring the total all future reward received after starting from this state and following a policy of actions?
- Then the robot could choose an action that will lead to a state with the highest value.

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

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

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

Now, the task is to learn the optimal policy π^*:

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

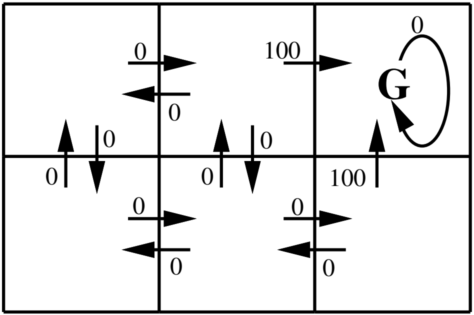


Figure 1: $r(s, a)$ (immediate reward) values.

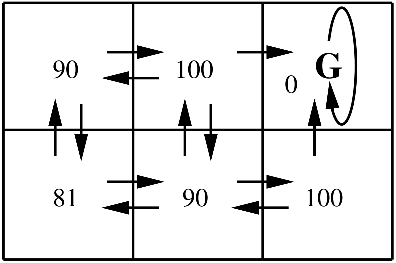


Figure 2: $V^*(s)$ values for $\gamma = 0.9$.

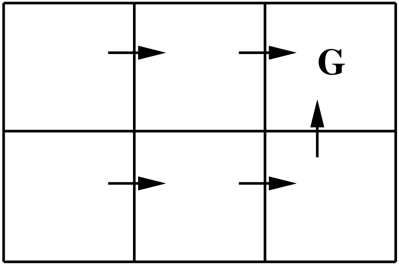


Figure 3: One optimal policy.

How to Calculate the V values?

- Select a move: Most of the time we move *greedily*, i.e. select the move that leads to the state with greatest value (**Exploitation step**).
- Occasionally, we select randomly from among the other moves instead (**Exploration step**).

How to do iteratively? Update the V for <u>only</u> greedy moves according to the formula:

$$V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1} - V(S_t)] \tag{10}$$

where α is a small positive number (in the range between 0 and 1), which affects the rate of learning.

ϵ-Greedy Methods for Exploration vs Exploitation

- To make sure that we explore while we exploit as well, ϵ-greedy actions can be applied:
- Most of the time a greedy action is selected (i.e. the one leading to the maximum V value estimated so far).
- With probability ϵ we apply an action which is selected randomly from all the actions (**including the greedy action**) with equal probability.

Q Function - An Alternative to choose Robot Actions

Define new function very similar to V^*

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

If agent learns Q, it can choose optimal action even without knowing δ! (the function which describes the transition between the current state and the next one if the robot takes a specific action)

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

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

Q is the evaluation function the agent will learn

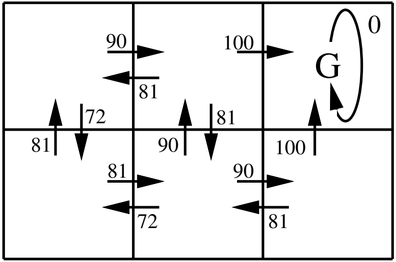


Figure 4: $Q(s, a)$ values for the grid problem previously seen.

Training Rule to Learn Q

Note Q and V^* closely related:

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

Which allows us to write Q recursively as

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

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

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

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

Q Learning Pseudocode for Deterministic Worlds

For each s, a initialise table entry $\hat{Q}(s, a) \longleftarrow 0$

Observe current state s

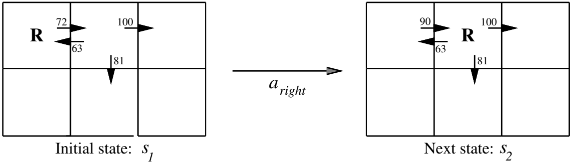
Do forever:

- 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:

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

- $s \longleftarrow s'$

Updating $\hat{Q}$



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

notice if rewards non-negative, then

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

and

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

Nondeterministic Case

What if reward and next state are non-deterministic?

We redefine V, Q by taking expected values

$$
\begin{aligned}
V^{\pi}(s) &\equiv E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots] \\
&\equiv E[\sum_{i=0}^{\infty} \gamma^i r_{t+i}] \\
&\equiv E[r_t + \gamma V^{\pi}(s+1)] \qquad (11)
\end{aligned}
$$

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

Nondeterministic Case

Q learning generalises to nondeterministic worlds

Alter training rule to

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

where

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

Can still prove convergence of $\hat{Q}$ to Q.

Temporal Difference Learning

Q learning (TD(0) algorithm): 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:

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

Families of Reinforcement Learning Algorithms

1. *Dynamic Programming*: based on Bellman equation (11), well developed mathematically, but require a complete and accurate model of the environment.
2. *Monte Carlo Methods*: do not require a model but not appropriate for step-by-step incremental learning.
3. *Temporal Difference Methods* (e.g. Q-learning (which is the TD(0) algorithm), temporal difference learning): require no model, they are fully incremental, but are more complex to analyse.

Other Improvements on what was discussed

- Extend to continuous action, state: Replace $\hat{Q}$ table with neural net or other generaliser
- Learn and use $\hat{\delta} : S \times A \longrightarrow S$