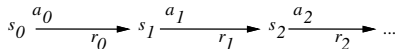
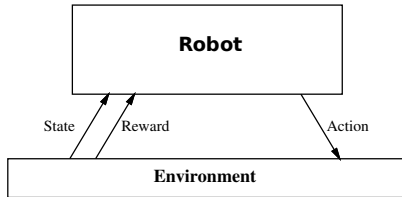


6ELEN018W - Applied Robotics
Lecture 4: Robot Control - Intelligent Control
Algorithms - Part II

Dr Dimitris C. Dracopoulos

The Reinforcement Learning Problem for a Robot



Goal: Learn to choose actions that maximise:

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

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

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

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^* :

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

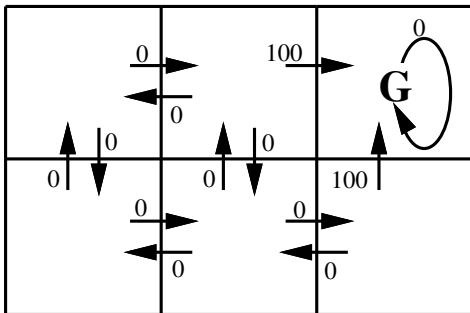


Figure 1: $r(s, a)$ (immediate reward) values.

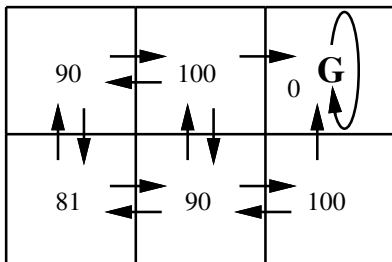


Figure 2: $V^*(s)$ values for $\gamma = 0.9$.

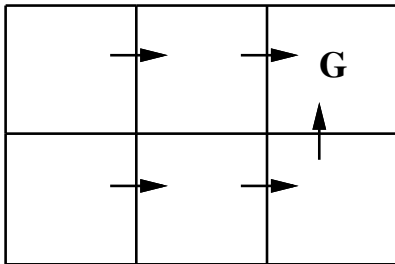


Figure 3: One optimal policy.

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 only greedy moves according to the formula:

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

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

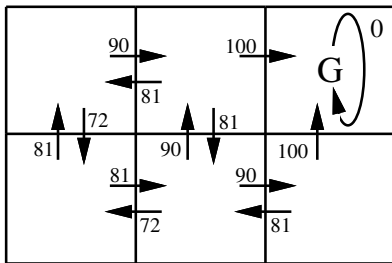


Figure 4: $Q(s, a)$ values for the grid problem previously seen.

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 $\hat{Q}(s, a) \leftarrow 0$

Observe current state s

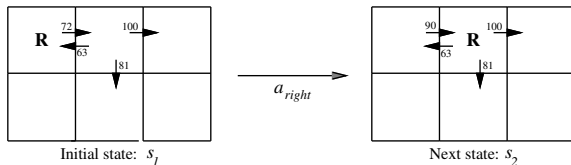
Do forever:

- ▶ Select an action a and execute it
- ▶ Receive immediate reward r
- ▶ Observe the new state s'
- ▶ Update the table entry for $\hat{Q}(s, a)$ as follows:

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

- ▶ $s \leftarrow s'$

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

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

and

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

Nondeterministic Case

What if reward and next state are non-deterministic?

We redefine V, Q by taking expected values

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

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

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

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

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 (2), 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 $\hat{\delta} : S \times A \longrightarrow S$