6COSC020W - APPLIED AI
An Introduction to Reinforcement Learning

Dr Dimitris C. Dracopoulos

Control Learning

Consider learning to choose actions, e.g.,

- Robot learning to dock on battery charger
- Learning to choose actions to optimise factory output
- Learning to play Backgammon

Note several problem characteristics:

- Delayed reward
- Opportunity for active exploration
- Possibility that state only partially observable
- Possible need to learn multiple tasks with same sensors/effectors

Machine (X) vs Machine (O)

Learn to play the tic-tac-toe game.

X	X	O
	X	O
	O	X

- 2 machines playing each other without previous knowledge
- Playing 20000 games
- Learning in each game!

Another Example: TD-Gammon

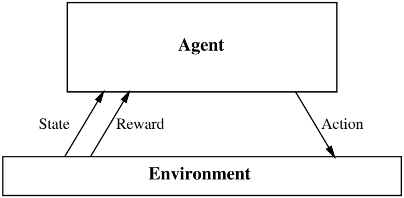
Learn to play Backgammon.
Immediate reward:

- $+100$ if win
- -100 if lose
- 0 for all other states

Trained by playing 1.5 million games against itself

Now approximately equal to best human player

The Reinforcement Learning Problem



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

Goal: Learn to choose actions that maximise:

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

where $0 \leq \gamma < 1$

Markov Decision Processes

Assume:

- finite set of states S
- set of actions A
- at each discrete time agent observes state $s_t \in S$ and chooses action $a_t \in A$
- then receives immediate reward r_t
- and state changes to s_{t+1}
- Markov assumption: $s_{t+1} = \delta(s_t, a_t)$ and $r_t = r(s_t, a_t)$
  - i.e., r_t and s_{t+1} depend only on *current* state and action
  - functions δ and r may be nondeterministic
  - functions δ and r not necessarily known to agent

Agent'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

Note something new:

- Target function is $\pi : S \longrightarrow A$
- but we have no training examples of form $\langle s, a \rangle$
- training examples are of form $\langle \langle s, a \rangle, r \rangle$

Value Function

To begin, consider deterministic worlds...
For each possible policy π the agent might adopt, we can define an evaluation function over states

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

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

Restated, the task is to learn the optimal policy π^*

$$\pi^* \equiv \arg\max_{\pi} V^{\pi}(s), (\forall s)$$

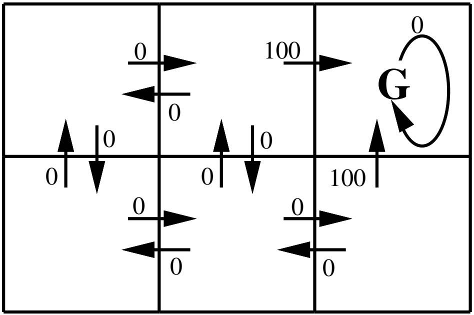


Figure 1: $r(s, a)$ (immediate reward) values.

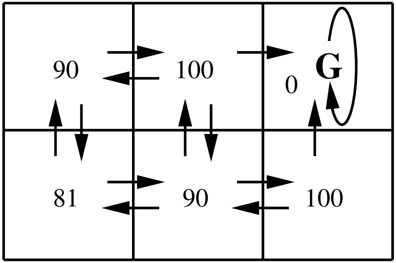


Figure 2: $V^*(s)$ values for $\gamma = 0.9$.

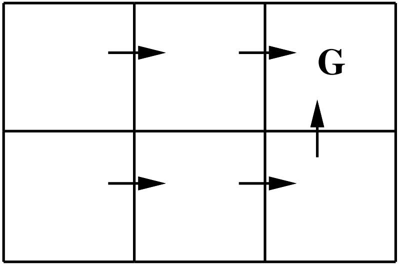


Figure 3: One optimal policy.

What to Learn

We might try to have agent learn the evaluation function V^{π^*} (which we write as V^*)

It could then do a lookahead search to choose best action from any state s because

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

A problem:

- This works well if agent knows $\delta : S \times A \longrightarrow S$, and $r : S \times A \longrightarrow \Re$
- But when it doesn't, it can't choose actions this way

Q Function

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!

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

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

Q is the evaluation function the agent will learn

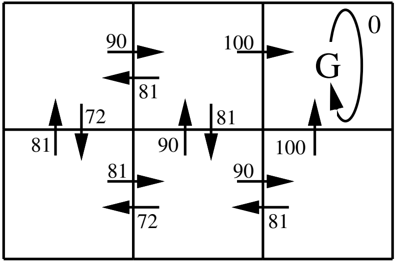


Figure 4: $Q(s, a)$ values for the grid problem previously seen.

Learning V in a Problem

- Select the moves.
- 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:

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

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.

An Example of ϵ-Greedy Selection

- Assume a problem with 3 states s_1, s_2, s_3.
- There are 2 actions available from each state s_i. Each action will lead to one of the other 2 states.
- Assume that the currently estimated V values for the 3 states are $V(s_1) = 10$, $V(s_2) = 50$ and $V(s_3) = 20$.
- The probability of selecting a non-greedy action is $\epsilon = 0.2$.

This can be implemented as follows:

- If the random generator (generating a number between 0 and 1) returns a value in the range $[0, 0.2]$ a random action is selected (i.e. it could be a greedy or a non-greedy one).
- If the random generator (generating a number between 0 and 1) returns a value in the range $(0.2, 1.0]$ a greedy approach is selected.

$\longrightarrow$ This means that the overall probability of selecting a greedy action is $0.8 + 0.1 = 0.9$ (i.e. selecting the greedy action because an exploitation move was selected + selecting the greedy action because an exploration move is made)

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 for Deterministic Worlds

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

Observe current state s

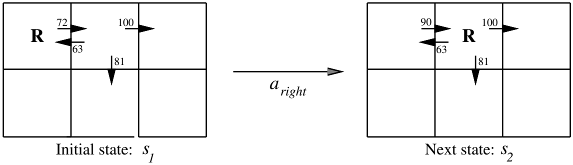
Do forever:

- 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

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

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

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

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

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

Temporal Difference Learning

Q learning: 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:

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

Temporal Difference Learning (cont'd)

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

Equivalent expression:

$$\begin{aligned} Q^{\lambda}(s_t, a_t) &= r_t + \gamma [\quad (1-\lambda) \max_a \hat{Q}(s_t, a_t) \\ &\qquad\qquad + \lambda \ Q^{\lambda}(s_{t+1}, a_{t+1})] \end{aligned}$$

TD(λ) algorithm uses above training rule

- Sometimes converges faster than Q learning
- converges for learning V^* for any $0 \leq \lambda \leq 1$
- Tesauro's TD-Gammon uses this algorithm

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, temporal difference learning): require no model, they are fully incremental, but are more complex to analyse.

Subtleties and Ongoing Research

- Replace $\hat{Q}$ table with neural net or other generaliser
- Handle case where state only partially observable
- Design optimal exploration strategies
- Extend to continuous action, state
- Learn and use $\hat{\delta} : S \times A \longrightarrow S$
- Relationship to dynamic programming