Learning to control Markov Decision Processes

CS7032: AI & Agents for IET

November 24, 2015
Outline

- Reinforcement Learning problem as a Markov Decision Process (MDP)
- Rewards and returns
- Examples
- The Bellman Equations
- Optimal state- and action-value functions and Optimal Policies
- Computational considerations
The Abstract architecture revisited (yet again)

- Add the ability to **evaluate** feedback:

```
Perception (S)  ▶️  Action (A)
```

```
Agent
```

```
Environment
```
The Abstract architecture revisited (yet again)

- Add the ability to **evaluate** feedback:

![Diagram of the abstract architecture](image)
The Abstract architecture revisited (yet again)

- Add the ability to **evaluate** feedback:

  ![Diagram of Agent, Environment, Perception, Action, Reward]

- How to represent **goals**?
Interaction as a Markov decision process

- We start by simplifying *action* (as in purely reactive agents):
  - \( \text{action} : S \rightarrow A \) (*New notation: \( \text{action} \overset{\text{def}}{=} \pi \))
  - \( \text{env} : S \times A \rightarrow S \) (New notation: \( \text{env} \overset{\text{def}}{=} \delta \))

- 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 \( s_{t+1} \) (deterministic case)
Levels of abstraction

- Time steps need not be fixed real-time intervals.
- Actions can be low level (e.g., voltages to motors), or high level (e.g., accept a job offer), mental (e.g., shift in focus of attention), etc.
- States can be low-level sensations, or they can be abstract, symbolic, based on memory, or subjective (e.g., the state of being surprised or lost).
- An RL agent is not like a whole animal or robot.
  - The environment encompasses everything the agent cannot change arbitrarily.
- The environment is not necessarily unknown to the agent, only incompletely controllable.
Specifying goals through rewards

▶ The reward hypothesis [Sutton and Barto, 1998, see]:

All of what we mean by goals and purposes can be well thought of as the maximization of the cumulative sum of a received scalar signal (reward).

▶ Is this correct?
Specifying goals through rewards

- The reward hypothesis [Sutton and Barto, 1998, see]:
  All of what we mean by goals and purposes can be well thought of as the maximization of the cumulative sum of a received scalar signal (reward).

- Is this correct?

- Probably not: but simple, surprisingly flexible and easily disprovable, so it makes scientific sense to explore it before trying anything more complex.
Some examples

- Learning to play a game (e.g. draughts):

  +1 for winning, \(-1\) for losing, \(0\) for drawing

- Learning how to escape from a maze:

  set the reward to zero until it escapes and \(+1\) when it does.

- Recycling robot:

  +1 for each recyclable container collected, \(-1\) if container isn't recyclable, \(0\) for wandering, \(-1\) for bumping into obstacles etc.
Some examples

- Learning to play a game (e.g. draughts):
  - +1 for winning, −1 for losing, 0 for drawing
  - (similar to the approach presented previously.)
Some examples

- Learning to play a game (e.g. draughts):
  - +1 for winning, −1 for losing, 0 for drawing
  - (similar to the approach presented previously.)
- Learning how to escape from a maze:
Some examples

- Learning to play a game (e.g. draughts):
  - +1 for winning, −1 for losing, 0 for drawing
  - (similar to the approach presented previously.)

- Learning how to escape from a maze:
  - set the reward to zero until it escapes
  - and +1 when it does.

- Recycling robot:
Some examples

- Learning to play a game (e.g. draughts):
  - +1 for winning, −1 for losing, 0 for drawing
  - (similar to the approach presented previously.)
- Learning how to escape from a maze:
  - set the reward to zero until it escapes
  - and +1 when it does.
- Recycling robot: +1 for each recyclable container collected, −1 if container isn’t recyclable, 0 for wandering, −1 for bumping into obstacles etc.
Important points about specifying a reward scheme

- the reward signal is the place to specify what the agent’s goals are (given that the agent’s high-level goal is always to maximise its rewards)
- the reward signal is not the place to specify how to achieve such goals
- Where are rewards computed in our agent/environment diagram?
- Rewards and goals are outside the agent’s direct control, so they it makes sense to assume they are computed by the environment!
From rewards to returns

- We define (expected) returns ($R_t$) to formalise the notion of rewards received in the long run.

- The simplest case:

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T$$  \hspace{1cm} (1)

where $r_{t+1}, \ldots$ is the sequence of rewards received after time $t$, and $T$ is the final time step.

- What sort of agent/environment is this definition most appropriate for?
From rewards to returns

- We define (expected) returns \((R_t)\) to formalise the notion of rewards received in the long run.
- The simplest case:

\[
R_t = r_{t+1} + r_{t+2} + \cdots + r_T \tag{1}
\]

where \(r_{t+1}, \ldots\) is the sequence of rewards received after time \(t\), and \(T\) is the final time step.
- What sort of agent/environment is this definition most appropriate for?
- Answer: episodic interactions (which break naturally into subsequences; e.g. a game of chess, trips through a maze, etc).
Non-episodic tasks

- Returns should be defined differently for continuing (aka non-episodic) tasks (i.e. $T = \infty$).
- In such cases, the idea of discounting comes in handy:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$  \hspace{1cm} (2)

where $0 \leq \gamma \leq 1$ is the discount rate

- Is this sum well defined?
- One can thus specify far-sighted or myopic agents by varying the discount rate $\gamma$. 
The pole-balancing example

Task: keep the pole balanced (beyond a critical angle) as long as possible, without hitting the ends of the track [Michie and Chambers, 1968]

▶ Modelled as an episodic task:
  ▶ reward of +1 for each step before failure ⇒ $R_t = \text{number of steps before failure}$

▶ Can alternatively be modelled as a continuing task:
  ▶ “reward” of $-1$ for failure and 0 for other steps ⇒ $R_t = -\gamma^k$ for $k$ steps before failure
Episodic and continuing tasks as MDPs

- Extra formal requirements for describing episodic and continuing tasks:
  - need to distinguish episodes as well as time steps when referring to states: \( s_{t,i} \) for time step \( t \) of episode \( i \) (we often omit the episode index, though)
  - need to be able to represent interaction dynamics so that \( R_t \) can be defined as sums over finite or infinite numbers of terms [equations (1) and (2)]
Extra formal requirements for describing episodic and continuing tasks:

- need to distinguish episodes as well as time steps when referring to states: \( s_{t,i} \) for time step \( t \) of episode \( i \) (we often omit the episode index, though)
- need to be able to represent interaction dynamics so that \( R_t \) can be defined as sums over finite or infinite numbers of terms [equations (1) and (2)]

Solution: represent termination as an absorbing state:

\[
\begin{align*}
  r_1 &= +1 \\
  r_2 &= +1 \\
  r_3 &= +1 \\
  r_4 &= 0 \\
  r_5 &= 0 \\
\end{align*}
\]

and making

\[
R_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}
\]

(where we could have \( T = \infty \) or \( \gamma = 1 \), but not both)
We assume that a reinforcement learning task has the Markov Property:

\[
P(s_{t+1} = s', r_{t+1} = r|s_t, a_t, r_t, \ldots r_1, s_0, a_0) = P(s_{t+1} = s', r_{t+1} = r|s_t, a_t)
\]

(3)

for all states, rewards and histories.

So, to specify a RL task as an MDP we need:

- to specify \( S \) and \( A \)
- and \( \forall s, s' \in S, a \in A: \)
  - transition probabilities:
    \[
    \mathcal{P}_{ss'}^a = P(s_{t+1} = s'|s_t = s, a_t = a)
    \]
  - and rewards \( \mathcal{R}_{ss'}^a \), Where a reward could be specified as an average over transitions from \( s \) to \( s' \) when the agent performs action \( a \)
    \[
    \mathcal{R}_{ss'}^a = E\{r_{t+1}|s_t = s, a_t = a, s_{t+1} = s'\}
The recycling robot revisited

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if it runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: high, low.
- Rewards = number of cans collected (or $-3$ if robot needs to be rescued for a battery recharge and 0 while recharging)
As a state-transition graph

- $S = \{\text{high, low}\}$, $A = \{\text{search, wait, recharge}\}$
- $R_{\text{search}} =$ expected no. of cans collected while searching
- $R_{\text{wait}} =$ expected no. of cans collected while waiting
  ($R_{\text{search}} > R_{\text{wait}}$)
Value functions

- RL is (almost always) based on estimating value functions for states, i.e. how much return an agent can expect to obtain from a given state.

- We can define the state-value function under policy $\pi$ as the expected return when starting in $s$ and following $\pi$ thereafter:

$$V^\pi(s) = E_\pi\{R_t|s_t = s\} \quad (4)$$

- Note that this implies averaging over probabilities of reaching future states, that is, $P(s_{t+1} = s'|s_t = s, a_t = a)$ over all $t$.

- We can also generalise the action function (policy) to $\pi(s, a)$, returning the probability of taking action $a$ while in state $s$, which implies also averaging over actions.
we can also define an action-value function to give the value of taking action \( a \) in state \( s \) under a policy \( \pi \):

\[
Q^\pi(s, a) = E_\pi\{R_t | s_t = s, a_t = a\}
\]  

(5)

where \( R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \).

Both \( v^\pi \) and \( Q^\pi \) can be estimated, for instance, through simulation (Monte Carlo methods):

- for each state \( s \) visited by following \( \pi \), keep an average \( \hat{V}^\pi \) of returns received from that point on.
- \( \hat{V}^\pi \) approaches \( V^\pi \) as the number of times \( s \) is visited approaches \( \infty \).
- \( Q^\pi \) can be estimated similarly.
The Bellman equation

- Value functions satisfy particular recursive relationships.
- For any policy \( \pi \) and any state \( s \), the following consistency condition holds:

\[
V^\pi(s) = E_\pi \{ R_t | s_t = s \}
\]
The Bellman equation

- Value functions satisfy particular recursive relationships.
- For any policy $\pi$ and any state $s$, the following consistency condition holds:

$$V^\pi(s) = E_\pi \{ R_t | s_t = s \} = E_\pi \{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \}$$

\[ (6) \]
The Bellman equation

- Value functions satisfy particular recursive relationships.
- For any policy $\pi$ and any state $s$, the following consistency condition holds:

$$V^\pi(s) = E_\pi \{ R_t | s_t = s \}$$

$$= E_\pi \{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \}$$

$$= E_\pi \{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s \}$$

(6)
The Bellman equation

- Value functions satisfy particular recursive relationships.
- For any policy \( \pi \) and any state \( s \), the following consistency condition holds:

\[
V^\pi(s) = E_\pi\{R_t | s_t = s\} = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\} = E_\pi\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\} = \sum_{a} \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s'\}] \]

(6)
The Bellman equation

- Value functions satisfy particular recursive relationships.
- For any policy $\pi$ and any state $s$, the following consistency condition holds:

$$V^\pi(s) = E_\pi \{ R_t | s_t = s \}$$

$$= E_\pi \{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \}$$

$$= E_\pi \{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s \}$$

$$= \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [ R_{ss'}^a + \gamma E_\pi \{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s' \} ]$$

$$= \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [ R_{ss'}^a + \gamma V^\pi(s') ]$$

(6)
Backup diagrams

- The Bellman equation for $V^\pi$ (6) expresses a relationship between the value of a state and the value of its successors.
- This can be depicted through backup diagrams

representing transfers of value information back to a state (or a state-action pair) from its successor states (or state-action pairs).
Backup diagrams

- The Bellman equation for $V^\pi$ (6) expresses a relationship between the value of a state and the value of its successors.
- This can be depicted through backup diagrams.

```
V^\pi \rightarrow
```

- representing transfers of value information back to a state (or a state-action pair) from its successor states (or state-action pairs).
The Bellman equation for $V^\pi$ (6) expresses a relationship between the value of a state and the value of its successors.

This can be depicted through backup diagrams representing transfers of value information back to a state (or a state-action pair) from its successor states (or state-action pairs).
An illustration: The GridWorld

- Deterministic actions (i.e. $P_{ss'}^a = 1$ for all $s, s', a$ such that $s'$ is reachable from $s$ through $a$; or 0 otherwise);
- Rewards: $R^a = -1$ if $a$ would move agent off the grid, otherwise $R^a = 0$, except for actions from states A and B.

Diagram (b) shows the solution of the set of equations (6), for equiprobable (i.e. $\pi(s, \uparrow) = \pi(s, \downarrow) = \pi(s, \leftarrow) = \pi(s, \rightarrow) = .25$, for all $s$) random policy and $\gamma = 0.9$
Optimal Value functions

- For finite MDPs, policies can be partially ordered:
  \[ \pi \geq \pi' \iff V^\pi(s) \geq V^{\pi'}(s), \quad \forall s \in S \]

- There are always one or more policies that are better than or equal to all the others. These are the optimal policies, denoted \( \pi^* \).

- The Optimal policies share the same
  - optimal state-value function: \( V^*(s) = \max_\pi V^\pi(s), \quad \forall s \in S \) and
  - optimal action-value function:
    \[ Q^*(s, a) = \max_\pi Q^\pi(s, a), \quad \forall s \in S \text{ and } a \in A \]
The value of a state under an optimal policy must equal the expected return for the best action from that state:

\[ V^*(s) = \max_{a \in A(s)} Q^*(s, a) \]

\[ = \max_{a} E_{\pi^*} \{ R_t | s_t = s, a_t = a \} \]

\[ = \max_{a} E_{\pi^*} \{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, a_t = a \} \]

\[ = \max_{a} E_{\pi^*} \{ r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a \} \quad (7) \]

\[ = \max_{a \in A(s)} \sum_{s'} P_{ss'}^a [ R_{ss'}^a + \gamma V^*(s') ] \quad (8) \]
Bellman optimality equation for $Q^*$

- Analogously to $V^*$, we have:

$$Q^*(s, a) = \mathbb{E}\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a\}$$

$$= \sum_{s'} P_{ss'}^a [R_{ss'} + \gamma \max_{a'} Q^*(s', a')]$$

- $V^*$ and $Q^*$ are the unique solutions of these systems of equations.
From optimal value functions to policies

- Any policy that is greedy with respect to $V^*$ is an optimal policy.
- Therefore, a one-step-ahead search yields the long-term optimal actions.
- Given $Q^*$, all the agent needs to do is set $\pi^*(s) = \arg \max_a Q^*(s, a)$.

![Diagram](image-url)

a) gridworld  

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>+10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A'</td>
<td></td>
<td></td>
<td>+5</td>
</tr>
</tbody>
</table>

b) $V^*$

<table>
<thead>
<tr>
<th></th>
<th>22.0</th>
<th>24.4</th>
<th>22.0</th>
<th>19.4</th>
<th>17.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.8</td>
<td>22.0</td>
<td>19.8</td>
<td>17.8</td>
<td>16.0</td>
<td></td>
</tr>
<tr>
<td>17.8</td>
<td>19.8</td>
<td>17.8</td>
<td>16.0</td>
<td>14.4</td>
<td></td>
</tr>
<tr>
<td>16.0</td>
<td>17.8</td>
<td>16.0</td>
<td>14.4</td>
<td>13.0</td>
<td></td>
</tr>
<tr>
<td>14.4</td>
<td>16.0</td>
<td>14.4</td>
<td>13.0</td>
<td>11.7</td>
<td></td>
</tr>
</tbody>
</table>

c) $\pi^*$
Knowledge and Computational requirements

- Finding an optimal policy by solving the Bellman Optimality Equation requires:
  - accurate knowledge of environment dynamics,
  - the Markov Property.

- Tractability:
  - polynomial in number of states (via dynamic programming)...
  - ...but number of states is often very large (e.g., backgammon has about $10^{20}$ states).
  - So approximation algorithms have a role to play

- Many RL methods can be understood as approximately solving the Bellman Optimality Equation.
These notes are based on [Sutton and Barto, 1998]. For a comprehensive formal treatment of MDPs and RL (under the name of “Neuro-dynamic programming” see [Bertsekas and Tsitsiklis, 1996].

*Neuro-Dynamic Programming.*
Athena Scientific, Belmont.

*BOXES: An experiment in adaptive control.*

*Reinforcement Learning: An Introduction.*
MIT Press, Cambridge, MA.