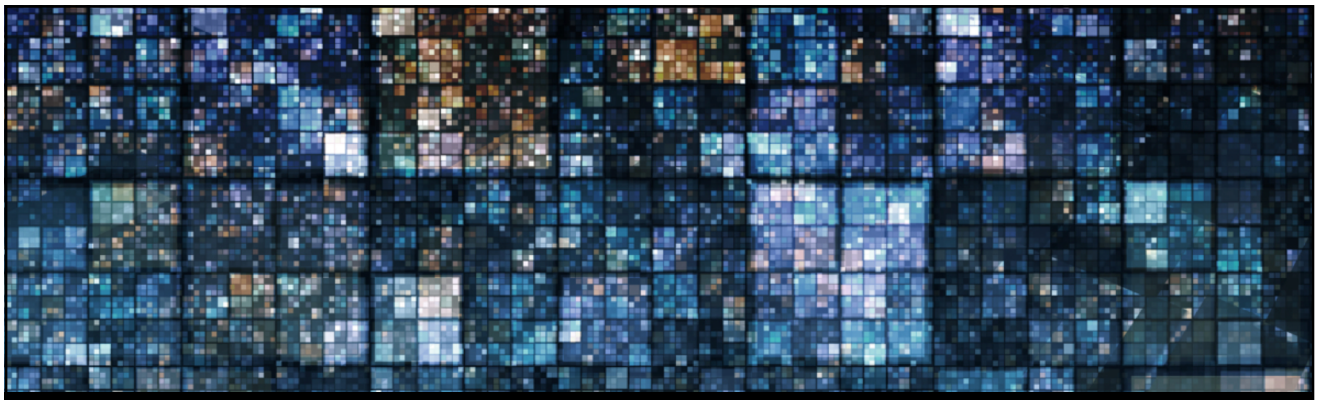
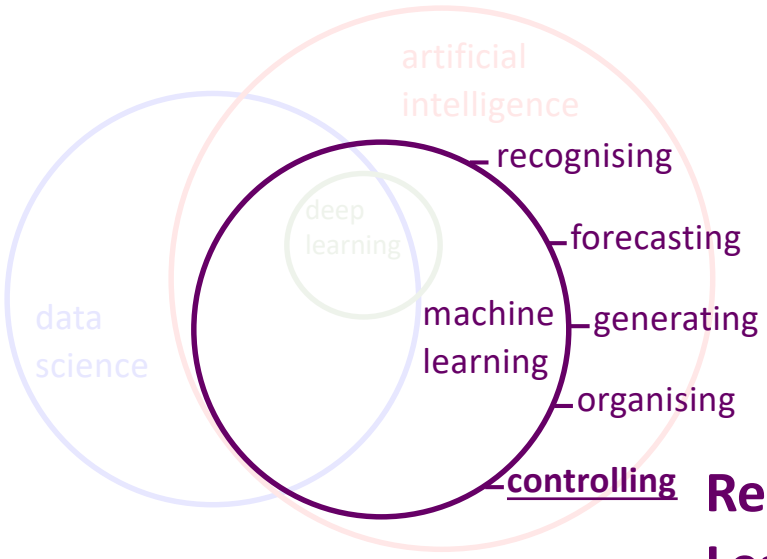


COMP47590
Advanced Machine Learning
RL - MDPs

Dr. Brian Mac Namee

Information

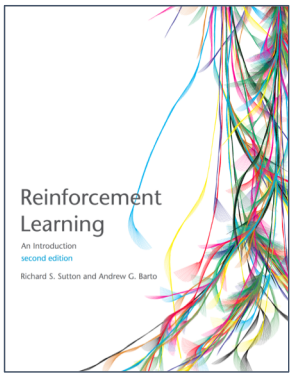
Email:	Brian.MacNamee@ucd.ie
Course Materials:	All material posted on UCD CS moodlehttps://csmoodle.ucd.ie/moodle/course/view.php?id=663 Enrolment key **UCDAdvML2019**



Reinforcement Learning

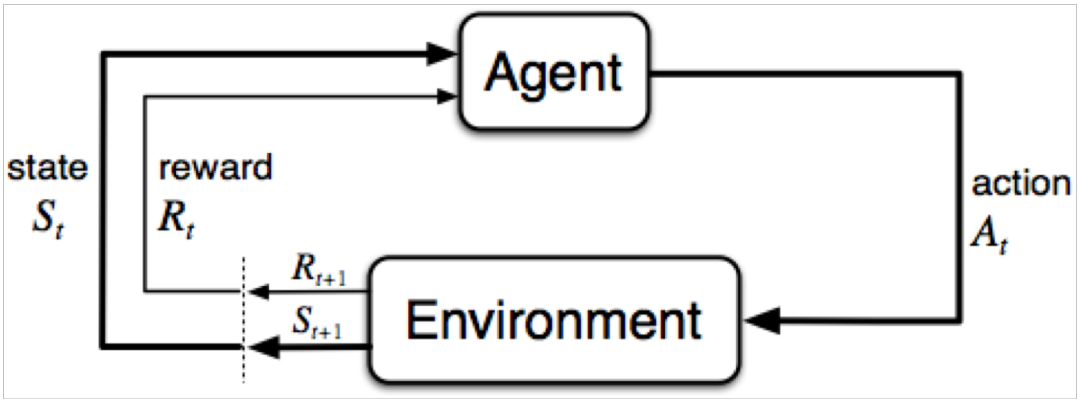
Reference Book

Much of the content in this section will follow "Reinforcement Learning An Introduction", 2nd Edition, Richard S. Sutton and Andrew G. Barto, MIT Press, 2018 (to appear)



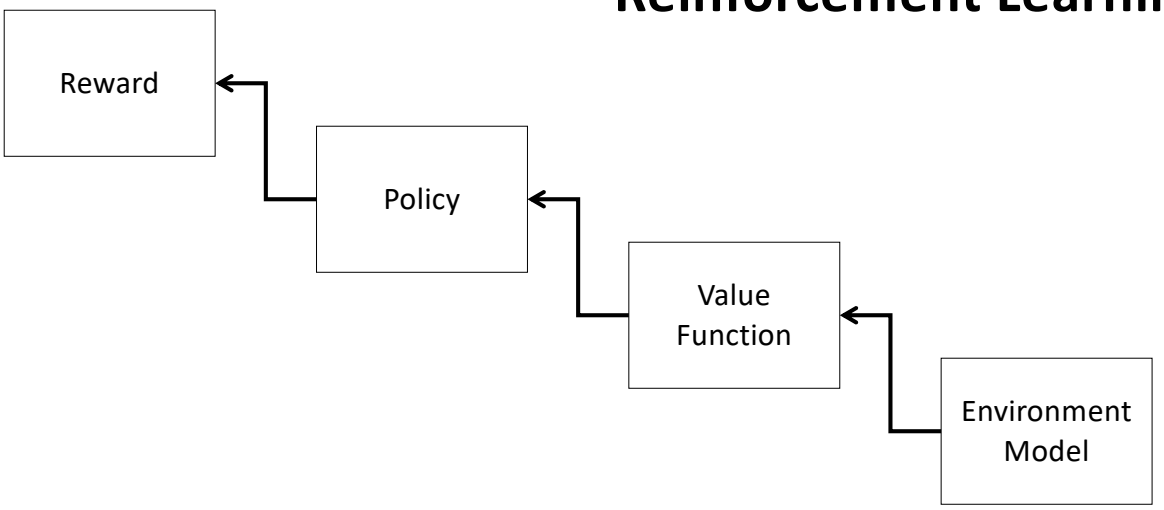
A legitimate online version of this book is available at:

www.incompleteideas.net/book/the-book-2nd.html



Reinforcement Learning: An Introduction , Second edition, in progress
Richard S. Sutton and Andrew G. Barto, MIT Press, 2017
www.incompleteideas.net/book/the-book-2nd.html

Reinforcement Learning



k-armed Bandit Problems

We introduced *k*-armed bandit problems as a simplified example of reinforcement learning

- agent: the arm puller
- environment: the bandit machine
- a policy: greedy or ε-greedy action selection
- a reward signal: R_t
- a value function: $Q_t(a)$
- a model of the environment: not needed

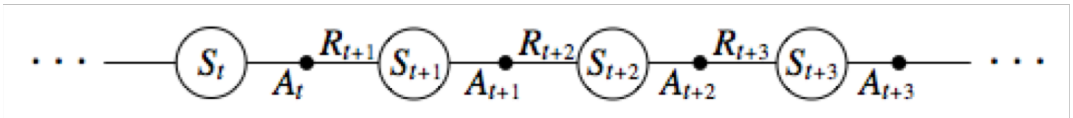
Finite Markov Decision Processes

We introduced finite Markov decision processes as a fundamental idea in reinforcement learning

- the agent and environment interact at each of a sequence of discrete time steps, t=0,1,2,3,....2
- at each time step the agent receives some representation of the environment's state, $S_t \in S$,
- based on S_t the agent selects an action, $A_t \in A(s)$
- One time step later the agent receives a numerical reward, R_{t+1}, and finds itself in a new state, S_{t+1}

Finite Markov Decision Processes

The MDP and agent together give rise to a sequence or trajectory that begins like this:



Reinforcement Learning: An Introduction , Second edition, in progress
Richard S. Sutton and Andrew G. Barto, MIT Press, 2017
www.incompleteideas.net/book/the-book-2nd.html

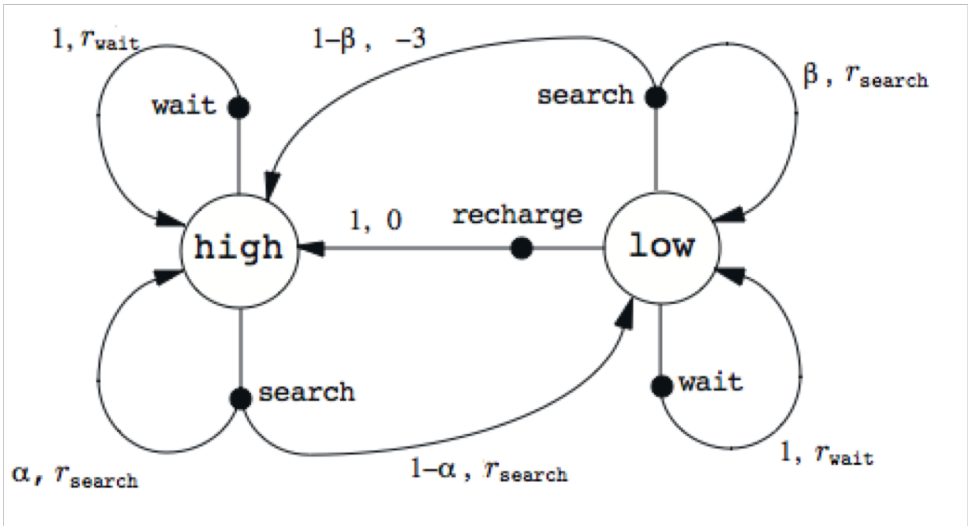
Finite Markov Decision Processes

For particular values of these random variables, $s' \in S$ and $r \in R$, there is a probability of those values occurring at time t, given particular values of the preceding state and action:

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s)$$

Example: Recycling Robot



Reinforcement Learning: An Introduction , Second edition, in progress
Richard S. Sutton and Andrew G. Barto, MIT Press, 2017
www.incompleteideas.net/book/the-book-2nd.html

The Markov Property

By "the state" at step *t*, we means whatever information is available to the agent at step *t* about its environment

- The state can include immediate "sensations," highly processed sensations, and structures built up over time from sequences of sensations
- State representation design is an important part of building reinforcement learning systems

The Markov Property

Ideally, a state should summarize past sensations so as to retain all "essential" information, this is the **Markov Property**:

$$\begin{aligned}
&\mathbf{Pr}\{R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\} \\
&\quad = \mathbf{Pr}\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\} \\
&\quad = p(s', r|s, a)
\end{aligned}$$

What's Missing? A Policy

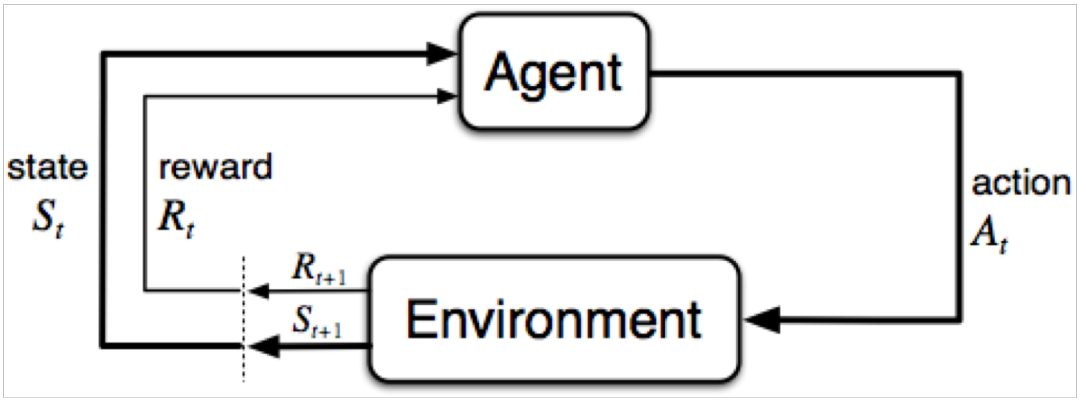
Policy at step t = π_t =

a mapping from states to action probabilities

$\pi_t(a \mid s)$ = probability that $A_t = a$ when $S_t = s$

Reinforcement learning methods specify how the agent changes its policy as a result of experience

GOALS, REWARDS & RETURNS



Reinforcement Learning: An Introduction , Second edition, in progress
Richard S. Sutton and Andrew G. Barto, MIT Press, 2017
www.incompleteideas.net/book/the-book-2nd.html

The Reward Hypothesis

Roughly, the agent's goal in reinforcement learning is to get as much reward as it can over the long run

This leads to the **reward hypothesis**:

That 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)

1

Goals and Rewards

Is a scalar reward signal an adequate notion of a goal?

- Maybe not, but it is surprisingly flexible and effective

Well designed goals and rewards:

- Should specify **what** to achieve, not **how** to achieve it
- A goal must be outside the agent's direct control
- The agent must be able to measure success according to the goal explicitly and frequently during its lifespan

1

Goals And Rewards

What would be sensible rewards for the following:

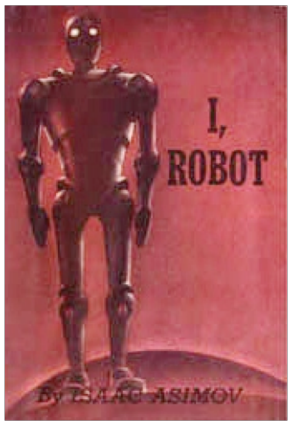
- A robot trying to find its way through a maze?
- Our rubbish cleaning robot?
- A chess playing agent?
- A robot trying to learn to walk?
- An automated taxi driving robot?
- An automated controller for the lunar landing game?

I, Robot

Isaac Asimov's 1950 short story collection "I, Robot" explores the design of goals and rewards

Features the three rules of robots:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.



Rewards and Returns

The objective in RL is to find a policy that chooses actions in particular states so as to maximize the expected long-term future reward

In general we try to maximize *expected return* which is defined as a function over the sequence of rewards received

Rewards and Returns

The simplest definition for expected return is simply the sum of rewards in the time steps following the current one

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T$$

where R_T is a final time step

Episodic Versus Continuing Tasks

Many tasks can be defined as having definite episodes

- A play of a game
- A trip through a maze
- An attempt to balance a pole

Other tasks do not have easily defined episodes:

- A pole balancing robot that doesn't drop the pole!
- Our rubbish cleaning robot

Rewards and Returns

Simply summing rewards does not work for continuous tasks - returns would tend towards infinity!

Instead we use a definition of **discounted return**:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Immediate rewards count for more than distant rewards

Rewards and Returns

γ	Reward sequence	Return
0.5	1 0 0 0…	
0.5	0 0 2 0 0 0…	
0.9	0 0 2 0 0 0…	
0.5	-1 2 6 3 2 0 0 0…	

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

γ	γ^2	γ^3	γ^4	γ^5
0.5	0.25	0.125	0.0625	0.03125
0.9	0.81	0.729	0.6561	0.59049

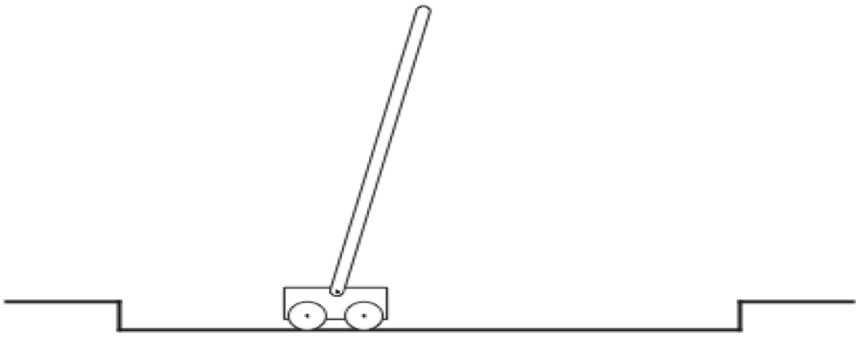
Rewards and Returns

γ	Reward sequence	Return
0.5	1 0 0 0…	1
0.5	0 0 2 0 0 0…	0.5
0.9	0 0 2 0 0 0…	1.62
0.5	-1 2 6 3 2 0 0 0…	2

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

γ	γ^2	γ^3	γ^4	γ^5
0.5	0.25	0.125	0.0625	0.03125
0.9	0.81	0.729	0.6561	0.59049

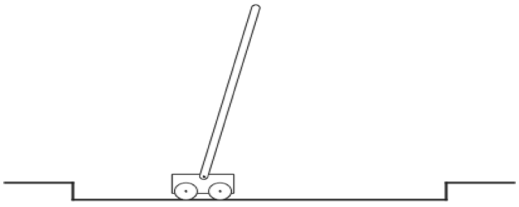
An Example: Pole Balancing



Avoid **failure:** the pole falling beyond a critical angle or the cart hitting end of track

Reinforcement Learning: An Introduction , Second edition, in progress
Richard S. Sutton and Andrew G. Barto, MIT Press, 2017
www.incompleteideas.net/book/the-book-2nd.html

An Example: Pole Balancing

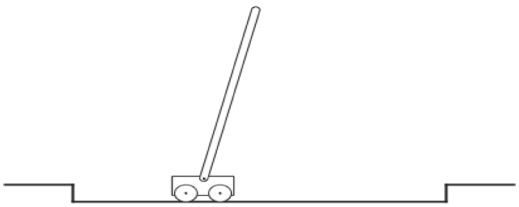


As an **episodic task** where episode ends upon failure:

Reinforcement Learning: An Introduction , Second edition, in progress
Richard S. Sutton and Andrew G. Barto, MIT Press, 2017
www.incompleteideas.net/book/the-book-2nd.html

2

An Example: Pole Balancing



As an **episodic task** where episode ends upon failure:

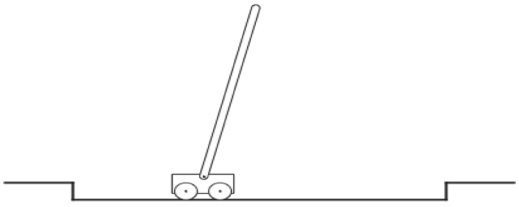
reward = +1 for each step before failure

$\Rightarrow$ return = number of steps before failure

Return is maximized by avoiding failure for as long as possible.

Reinforcement Learning: An Introduction , Second edition, in progress
Richard S. Sutton and Andrew G. Barto, MIT Press, 2017
www.incompleteideas.net/book/the-book-2nd.html

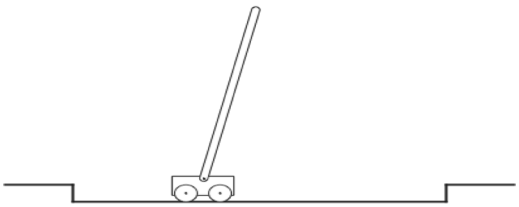
An Example: Pole Balancing



As an **continuing task** with discounted return:

Reinforcement Learning: An Introduction , Second edition, in progress
Richard S. Sutton and Andrew G. Barto, MIT Press, 2017
www.incompleteideas.net/book/the-book-2nd.html

An Example: Pole Balancing



As an **continuing task** with discounted return:

$$\text{reward} = -1 \text{ upon failure; } 0 \text{ otherwise}$$

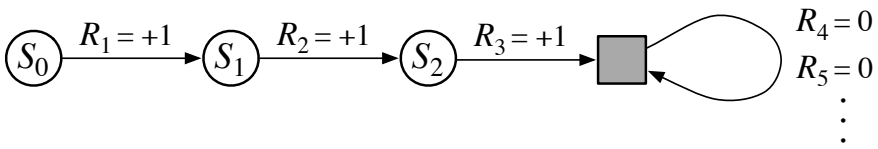
$$\Rightarrow \text{return} = -\gamma^k, \text{ for } k \text{ steps before failure}$$

Return is maximized by avoiding failure for as long as possible

Reinforcement Learning: An Introduction , Second edition, in progress
Richard S. Sutton and Andrew G. Barto, MIT Press, 2017
www.incompleteideas.net/book/the-book-2nd.html

Unifying Episodic & Continuous Cases

We can invent an **absorbing state** that episodic tasks fall into once an episode is complete



Allows us always use discounted expected returns summing to infinity

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

POLICIES & VALUE FUNCTIONS

Policies & Value Functions

Almost all reinforcement learning algorithms involve estimating **value functions** of **states** that estimate *how good* it is for the agent to be in a given state

- We can also define value functions in terms state-action pairs

The notion of "how good" is defined in terms of future rewards or **expected return**

But expected returns depend on the actions that an agent will take and so value functions are always defined with respect **policies**

Policies & Value Functions

A **policy** is a mapping from states to probabilities of selecting each possible action in that state

- If the agent is following policy π at time t, then $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$

Reinforcement learning methods specify how the agent's policy is changed as a result of its experience.

Policies & Value Functions

The value of a state s under a policy π is the expected return when starting in state s and taking actions following π thereafter:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$$

$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

Policies & Value Functions

We can similarly define the value of taking an action a in state *s* under policy π as:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

Policies & Value Functions

Similar to the case with expected return the state value function and action value function for a policy can be defined recursively:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a)\Big[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s']\Big] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r \mid s, a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S}, \end{aligned}$$

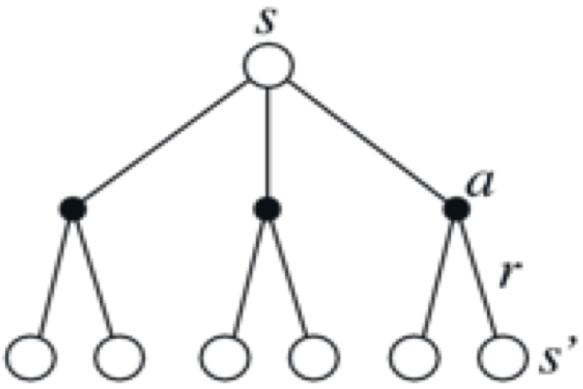
Policies & Value Functions

Similar to the case with expected return the state value function and action value function for a policy can be defined recursively:

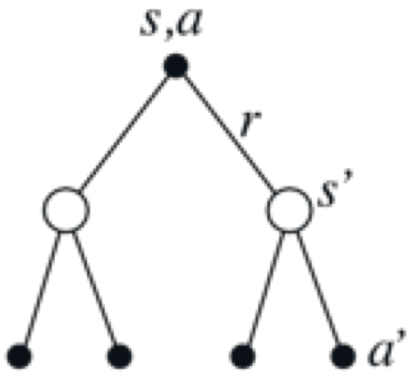
$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},
\end{aligned}
$$

Referred to as the **Bellman equation** for v_π

Backup Diagrams



for v_π



for q_π

OPTIMAL POLICIES

Optimal Policies

A policy π is is better than or equal to another policy π' if its expected return is greater than or equal to that of π' for all states

There is always at least one policy that is better than or equal to all other policies - this is the **optimal policy**

Optimal Policies

All optimal policies share the same **optimal state-value function**

$$v_*(s) = \max_\pi v_\pi(s)$$

And the **optimal action-value function**

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

Optimal Policies

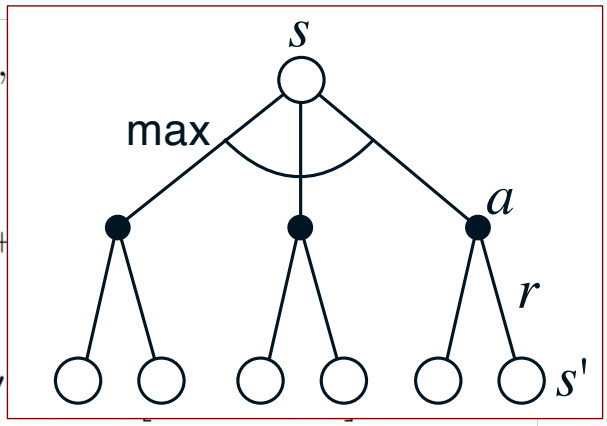
This gives rise to the **Bellman optimality equation** for state value:

$$\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_*(s')\big].
\end{aligned}$$

Optimal Policies

This gives rise to the **Bellman optimality equation** for state value:

$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+} \\
&= \max_a \mathbb{E}[R_{t+1} \\
&= \max_a \sum_{s',r} p(s'
\end{aligned}
$$



Optimal Policies

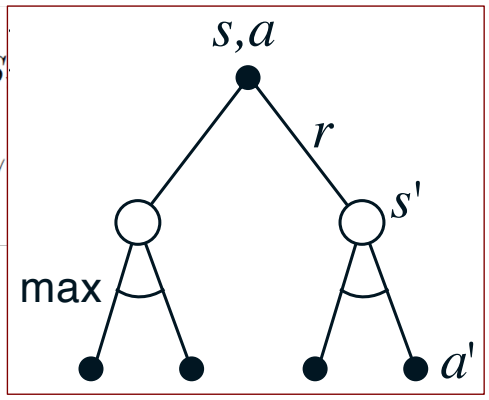
And for action value:

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\middle|\; S_t = s, A_t = a\right] \\
&= \sum_{s',r} p(s', r \,|\, s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right].
\end{aligned}
$$

Optimal Policies

And for action value:

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S'\right.$$
$$= \sum_{s',r} p(s',r\,|\,s,a)\left[r + \gamma\right.$$



Optimal Policies

For finite MDPs, the Bellman optimality equation for v_π has a unique solution independent of the policy which can be found by solving the set of equations (one equation per state)

The set of equations for q_π can similarly be solved

The **optimal policy** is a policy that assigns non-zero probability only to the optimal actions

Solving the Bellman Optimality Equation

Finding an optimal policy by solving the Bellman Optimality Equation requires the following:

- accurate knowledge of environment dynamics;
- we have enough space and time to do the computation;
- the Markov Property.

How much space and time do we need?

- polynomial in number of states,
- BUT, number of states is often huge (e.g., backgammon has about 10^{20} states).

Many RL methods can be understood as approximately solving the Bellman Optimality Equation.

4

SUMMARY

Summary

We have expanded our discussion of MDPs and how they can be used to frame reinforcement learning problems

We introduced the ideas of

- expected return
- policies
- optimal policies
- value functions

Questions