

Reinforcement Learning – COMS4061A/7071A

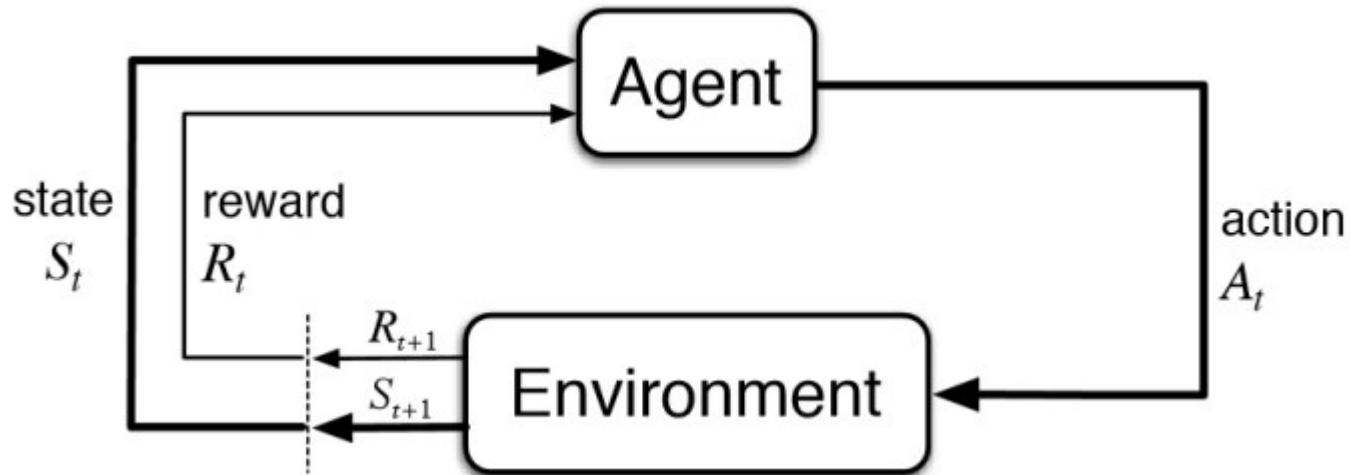
Markov Decision Processes

Prof. Benjamin Rosman

Benjamin.Rosman1@wits.ac.za / benjros@gmail.com

Based heavily on slides by Rich Sutton and Doina Precup

The RL problem

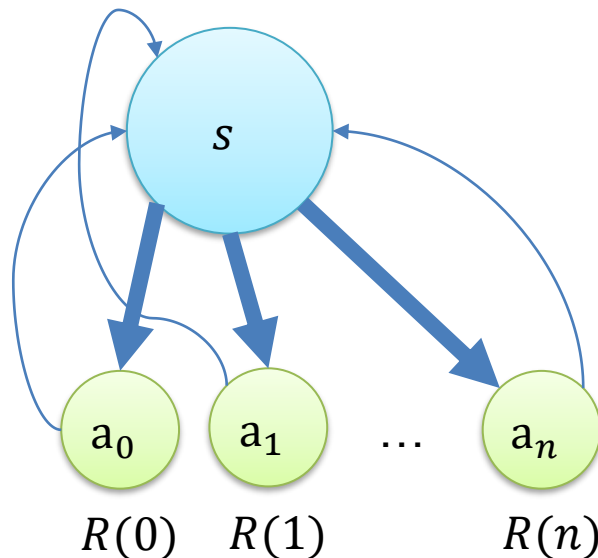


The RL problem is to find a **policy**: optimal actions a for each state s to maximise total rewards R

Last time: the bandit problem

Let's ignore **state** for now

- Assume there is **only one**
- We keep **choosing an action** $a \in A$ there
- This is the **bandit** problem



States

Now let's consider multiple **states** $s \in S$

What is a **state**?

- **Configuration** of the world that we care about
- Where we may want to **choose between actions**
- Whatever information is available to the agent
- Typically these **change while the agent acts**

e.g.

- Chess?
 - Configuration of chess board
- Robot in a maze?
 - (x,y)-position, door states, keys picked up
- Robot making coffee?
 - (x,y)-position, arm joint angles, position of cup, contents of cup, etc.

Dynamics

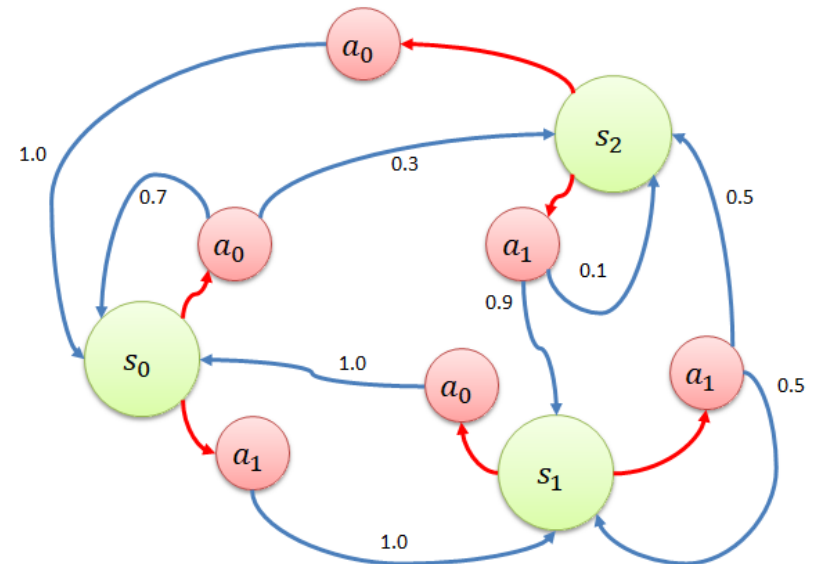
Actions allow us to change the current state of the world

Transition function:

$$P(s'|s, a) = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

Defines one-step stochastic dynamics

Sometimes represented as T



Reward functions



Specify **what we want to achieve**, not **how we want to achieve it**

Define a **reward function** as a feedback signal

$$r(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

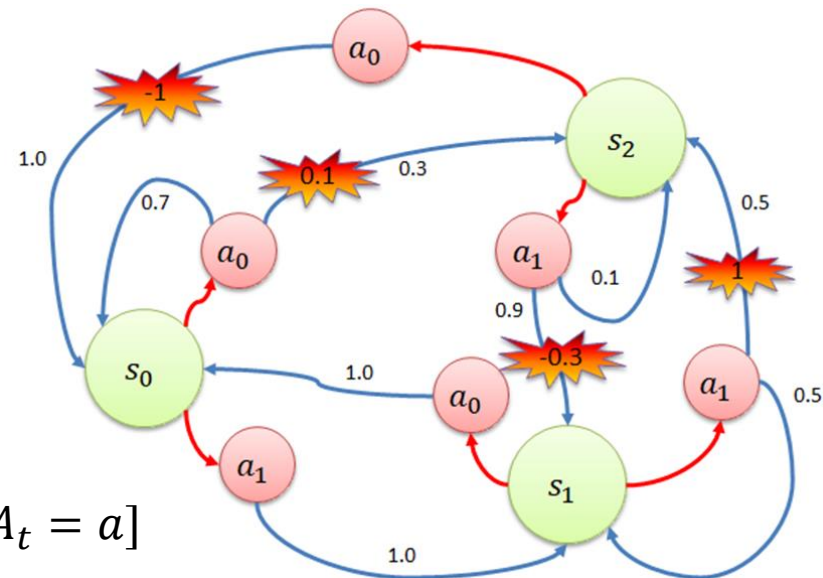
Sometimes $r(s, a, s')$

Scalar reward. Is this enough?

This is what we want to optimise
Suggests a cost function?

Combine with dynamics:

$$p(s', r | s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a]$$



What makes a good reward?

Chess?

- Win/lose/draw?
 - How sparse is this (how often do we get feedback)?
- Capturing the queen?
 - What behaviour does this give us?
- Taking pieces?
 - Will this give us what we want?

Self-driving car?

Trade-offs:

- Dense rewards
 - Faster to learn
- Sparse rewards
 - Less human bias and thinking about the solution



Returns

We really want to maximise **long-term future rewards**

Discounted return G_t at time t :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Discount rate: $\gamma \in [0,1]$

Discounting:

- Short-sighted or “myopic”: $\gamma \rightarrow 0$
- Far-sighted: $\gamma \rightarrow 1$

Tasks can be:

- **Episodic**
 - Natural episodes: plays of a game, trips through a maze, ...
 - Need terminal states
 - Often $\gamma = 1$
- **Continuing**
 - Task just goes on and on
 - $\gamma < 1$: otherwise can have infinite sums!

The Markov property

“The future is independent of the past given the present”

$$\mathbb{P}[S_{t+1} | S_1, A_1, S_2, A_2, \dots, S_t, A_t] = \mathbb{P}[S_{t+1} | S_t, A_t]$$

State contains **everything** we care about from the history

- Given the current state, we can throw away the history
- Why is this useful?
 - Computationally – this would grow the space over which we reason combinatorially

If we do need the history, we can just “roll” this into the state as a new variable

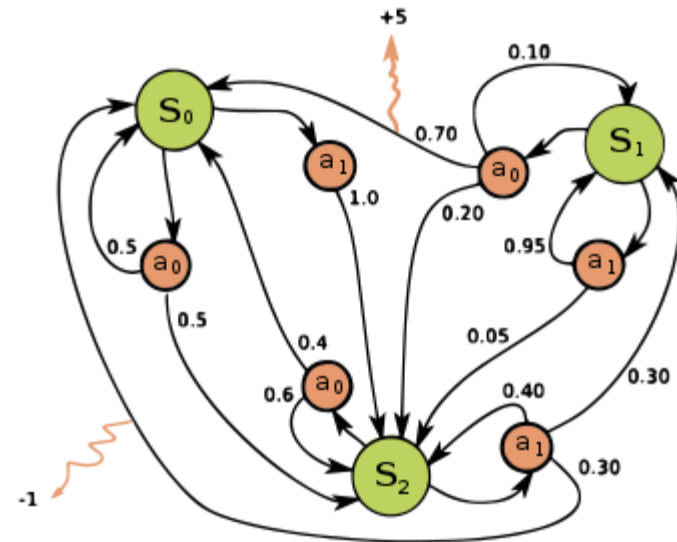
Markov decision processes

A **Markov decision process (MDP)** formally describes an environment for RL

Fully observable problem!

MDP = tuple $\langle S, A, P, R, \gamma \rangle$

- State space S
- Action space A
- Transition function P
- Reward function R
- Discount factor γ



A finite MDP is one where sets S and A are finite

Policies

How our agent chooses actions!

Mapping from states to action probabilities:

$$\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$$

Special case: deterministic policies:

$\pi(s) = a$ (action taken with probability 1)

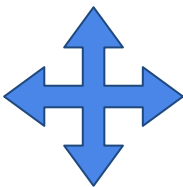
Want to find the policy that accumulates as much **total reward** (return) as possible

- Learning this **is** the RL problem

For any MDP, there always exists an optimal policy π_*

Example MDP: cleaning robot

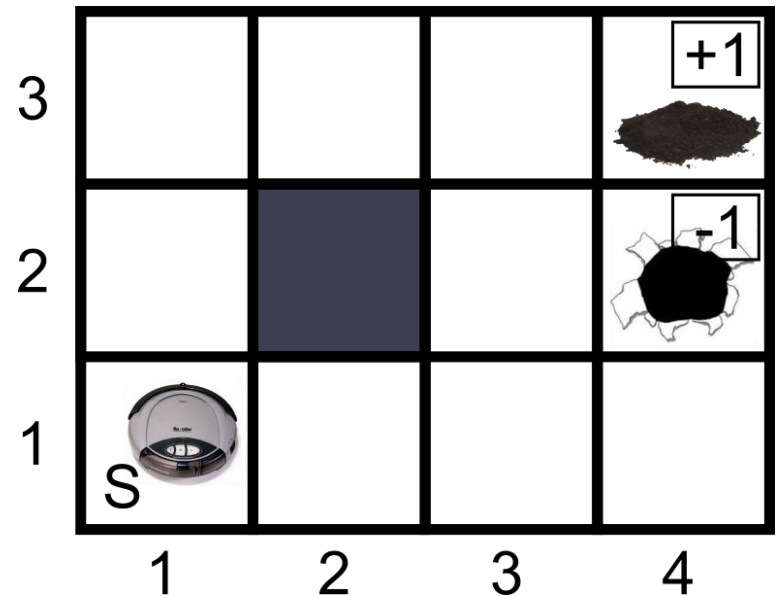
States: the 12-state grid

Actions: 

Dynamics: moving as expected

Reward:

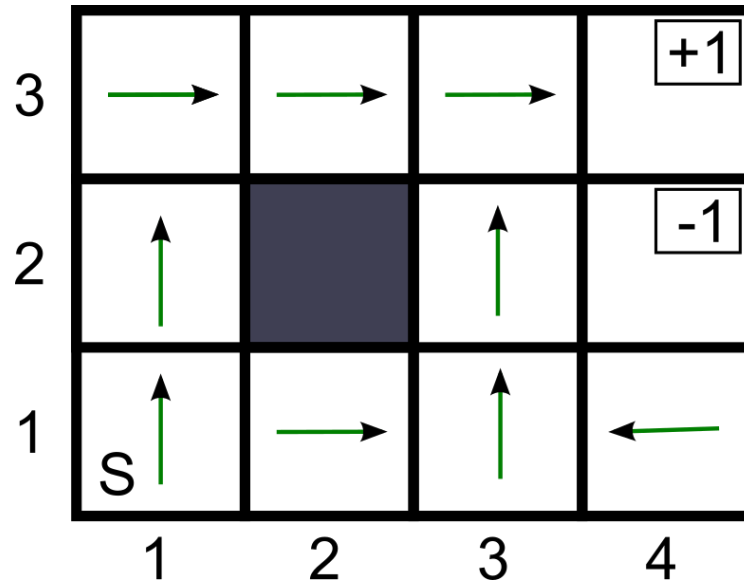
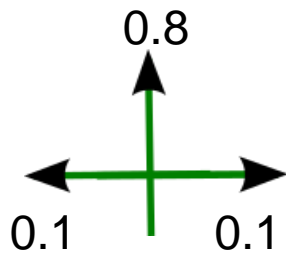
- +1 for finding dirt
- -1 for falling into hole
- -0.001 for every move



Episodic task: agent has repeated episodes of interaction (attempt at cleaning the room)

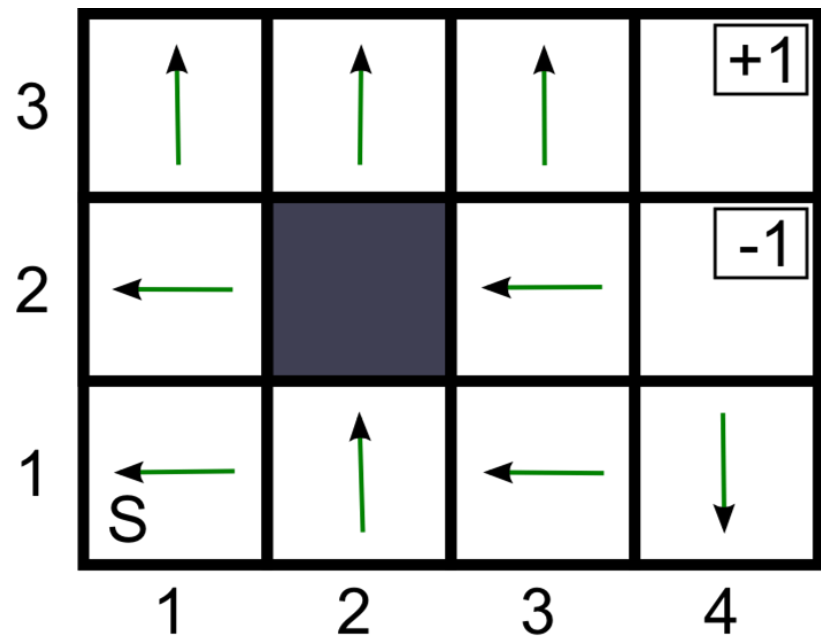
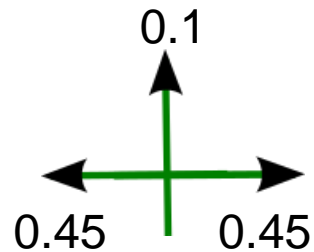
Optimal policy

Probability of movement if
intended direction was **up**



Optimal policy

Change action transitions to be very noisy



Example MDP: Chess

States:

all possible board configs

Actions:

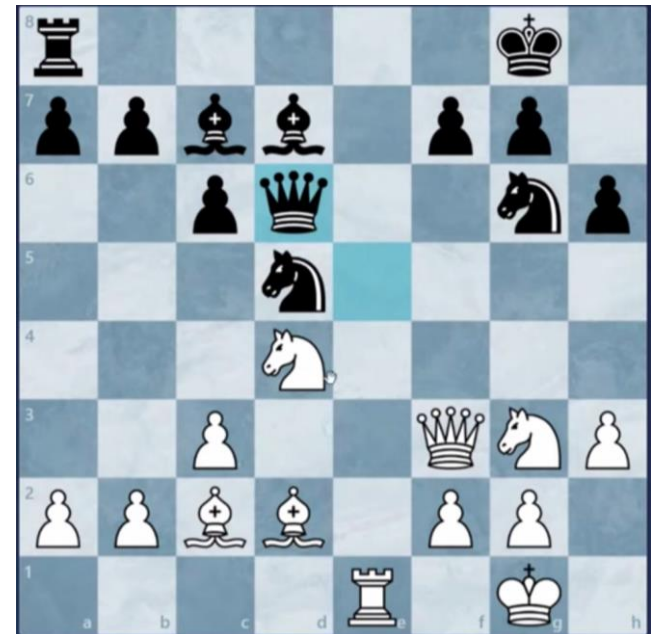
all legal moves

Dynamics:

piece moves as intended

Rewards:

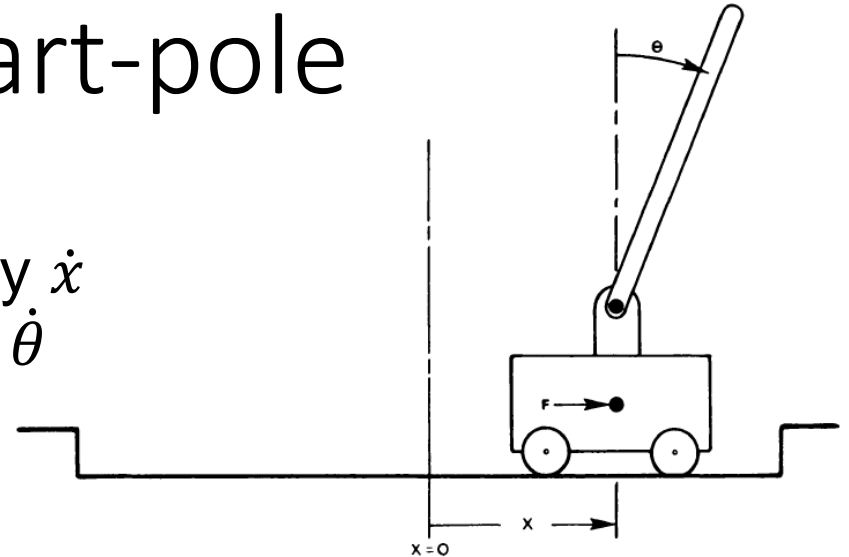
win = +1 / lose = -1



Example MDP: Cart-pole

State: cart position x , velocity \dot{x}
pole angle θ , velocity $\dot{\theta}$

Actions: force F_t



Dynamics:

$$\ddot{\theta}_t = \frac{g \sin \theta_t + \cos \theta_t \left[\frac{-F_t - m l \dot{\theta}_t^2 \sin \theta_t + \mu_c \operatorname{sgn}(\dot{x}_t)}{m_c + m} \right] - \frac{\mu_p \dot{\theta}_t}{m l}}{l \left[\frac{4}{3} - \frac{m \cos^2 \theta_t}{m_c + m} \right]}$$
$$\ddot{x}_t = \frac{F_t + m l [\dot{\theta}_t^2 \sin \theta_t - \ddot{\theta}_t \cos \theta_t] - \mu_c \operatorname{sgn}(\dot{x}_t)}{m_c + m}$$

Reward: +1 if upright

Optimising returns

If we care about **returns** rather than **rewards** when finding a policy, this is the thing we should be maximising!

But **we don't know the return** (the total reward we may receive in the future)!

So: can we estimate this?

Compute the **expected returns**.

These should be **conditioned on a policy**. Why?

Values are expected returns

The **value of a state**, given a policy:

$$v_{\pi}(s) = \mathbb{E}\{G_t | S_t = s, A_{t:\infty} \sim \pi\}, \quad v_{\pi} : S \rightarrow \mathbb{R}$$

i.e., the return if you start at state s and then follow π

The **optimal value of a state**:

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

Contrast against the
value of an action in a
MAB

The **value of a state-action pair**, given a policy:

$$q_{\pi}(s, a) = \mathbb{E}\{G_t | S_t = s, A_t = a, A_{t+1:\infty} \sim \pi\}, \quad q_{\pi} : S \times A \rightarrow \mathbb{R}$$

i.e., the return if you start at state s , take action a , and then follow π

The **optimal value of a state-action pair**:

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

Value functions

State-value function for policy π :

$$v_{\pi}(s) = \mathbb{E}_{\pi}\{G_t | S_t = s\} = \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right\}$$

Action-value function for policy π :

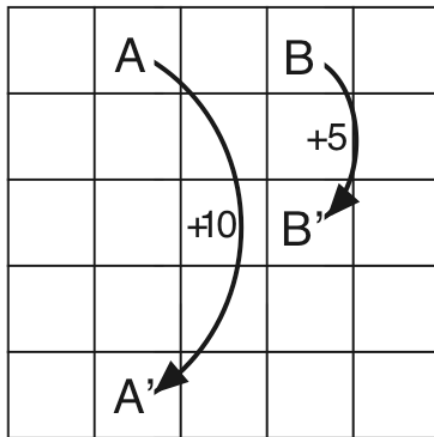
$$\begin{aligned} q_{\pi}(s, a) &= \mathbb{E}_{\pi}\{G_t | S_t = s, A_t = a\} \\ &= \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right\} \end{aligned}$$

Example value functions

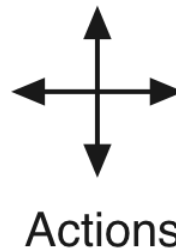
Deterministic actions

Trying to move off grid gives reward of -1

Reward = 0 otherwise, except at A and B



(a)



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

State-value function for
equiprobable random policy,
 $\gamma = 0.9$


Optimal value functions

- For finite MDPs, policies can be partially ordered:
 $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in S$
- There are always one or more policies that are better than or equal to all the others. These are **optimal policies**. We denote them all π_*
- Optimal policies **share the same optimal state-value function**:
$$v_*(s) = \max_{\pi} v_\pi(s) \quad \forall s \in S$$
- Optimal policies also **share the same optimal action-value function**:
$$q_*(s, a) = \max_{\pi} q_\pi(s, a) \quad \forall s \in S, a \in A$$
- This is the expected return for taking action a in state s and thereafter following an optimal policy.

Why value functions?

Any policy that is greedy with respect to v_* is an optimal policy.

Therefore, given v_* , one-step-ahead search produces the long-term optimal actions.



Which is the best neighbour?

What about optimal action-value functions?

Given q_* , the agent does not even have to do a one-step-ahead search:

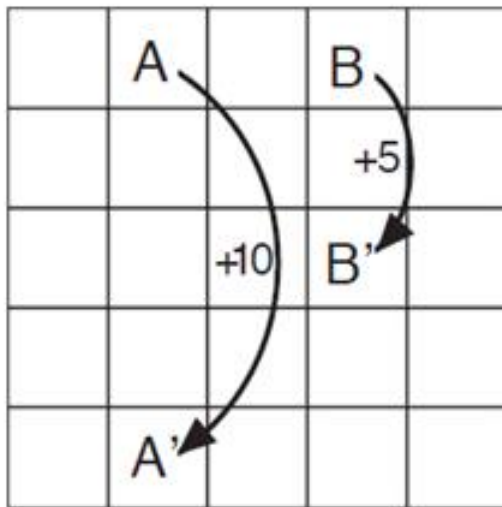
$$\pi_*(s) = \arg \max_a q_*(s, a)$$

Example optimal value functions

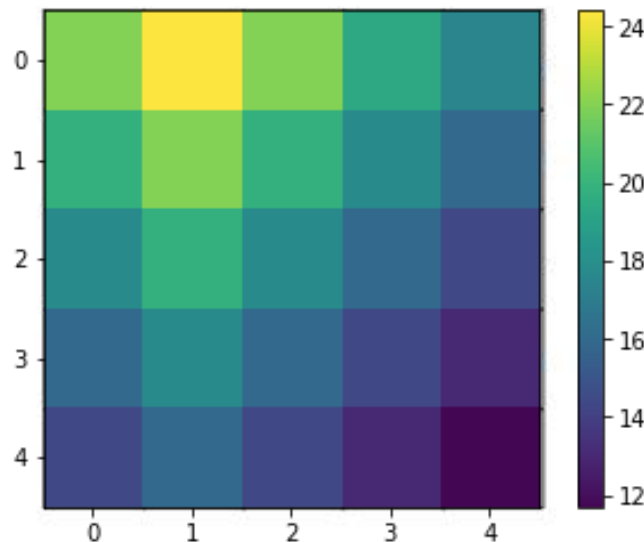
Deterministic actions

Trying to move off grid gives reward of -1

Reward = 0 otherwise, except at A and B

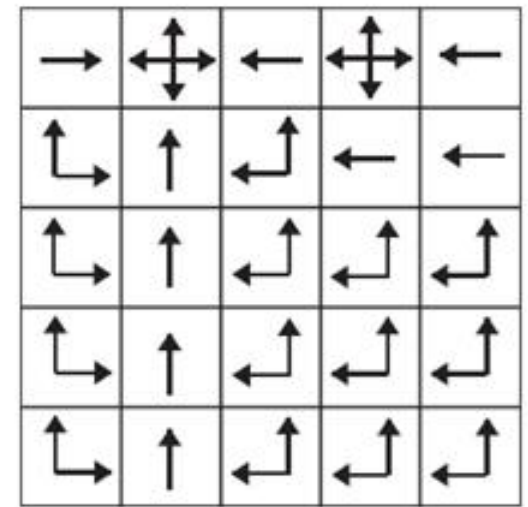


a) gridworld



b) v_*

Optimal policy:



c) π_*

How to compute a value function

Consider the return:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Immediate reward + discounted future return

So:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi\{G_t \mid S_t = s\} \\ &= \mathbb{E}\{R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\} \end{aligned}$$

Or, without the expectation operator:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_\pi(s')]$$

Recursive in v

This is called the **Bellman equation**

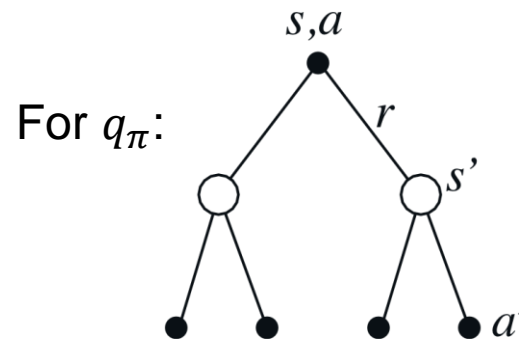
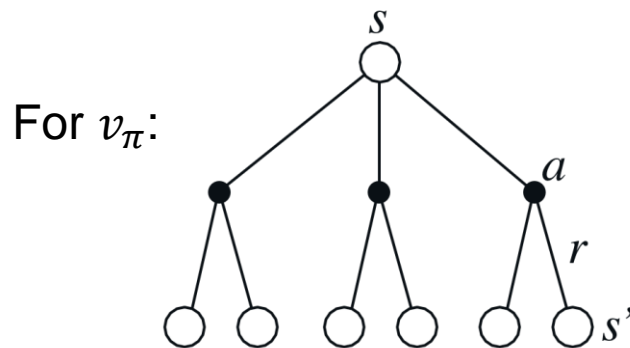
The Bellman equation

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_{\pi}(s')]$$

This is a **set of linear equations**: one for each state.

The value function for π is the unique solution.

Backup diagrams:

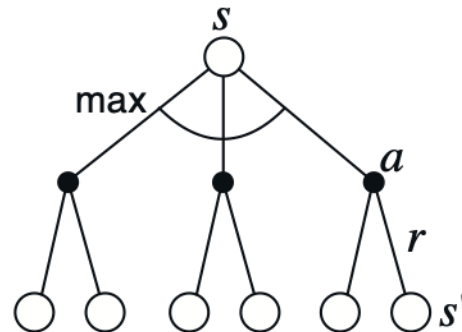


Bellman optimality equation for v_*

The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$\begin{aligned} v_*(s) &= \max_a q_{\pi_*}(s, a) \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')] \end{aligned}$$

The relevant backup diagram:



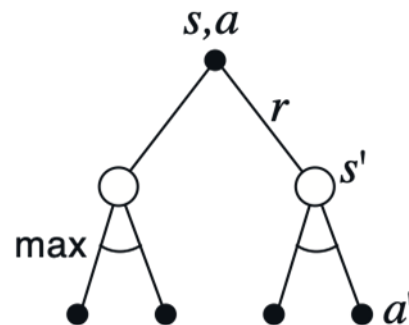
v_* is the unique solution to this system of nonlinear equations

Bellman optimality equation for q_*

Similarly,

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

The relevant backup diagram:



q_* is the unique solution to this system of nonlinear equations

Solving the Bellman optimality equation

Finding an optimal policy by solving the Bellman optimality equation requires:

- Accurate knowledge of environment dynamics
 - Transition and reward functions
- The Markov property
- Enough computational space and time

How much space and time?

- Polynomial in number of states
 - Next week!
- **But, number of states is often huge**
 - E.g. backgammon has about 10^{20} states

Usually have to approximate this

- **Many RL methods are just approximately solving the Bellman optimality equation**

POMDPs



What about if you can't see the entire state?
Partially Observable Markov Decision Process

$$\langle S, A, O, P, R, Z, \gamma \rangle$$

S is a finite set of states

A is a finite set of actions

O is a finite set of observations

P is the state transition function $P(s'|s, a) = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$

R is the reward function $R(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$

Z is an observation function $Z(o|s', a) = \mathbb{P}[O_{t+1} = o | S_{t+1} = s', A_t = a]$

$\gamma \in [0, 1]$ is the discount factor

Here we reason about the **probability of being in a particular state** (the belief) rather than the state itself

- This is a continuous distribution over the whole of S

Exercise

1. Gridworld domains are some of the most common in RL. Implement a 7x7 gridworld as an MDP:
 - Given a state and an action, you should be able to execute the action and return a new state and reward.
 - The actions are the cardinal directions and are deterministic. The states are just the cell the agent is currently in. Assume the third row from the top is all obstacles, except the right-most cell. The initial state is the bottom left, and the goal the top left.
 - There is no discounting. The reward for reaching the goal is 20, and -1 for each action.
2. Run a random agent on this domain that terminates after 50 steps.
3. Compute the optimal value function by hand and run a greedy agent using this value function.

By next week's lecture, submit on Moodle (groups of up to 4):

1. A bar graph of the returns accumulated by the two agents, averaged over 20 runs each.
2. A side-by-side illustration of a sample trajectory taken by each agent.
3. Your code