The Reinforcement Learning Problem

Markov Decision Processes
Chapter 3

…or the problem we will try to solve for the rest of the course

Reminders

- One thought question about Chapter 3 due Tuesday 26th Jan (6:00am). Sutton & Barto 2nd Ed (2016)

- Mrinmoy, your AI/TA is back tomorrow
  - his office is LH 406
  - office hours: Tues 11am - 1pm
  - his job is to help you & answer your questions about the assignment (he will mark them)

- Questions not specifically about the assignment, course material?
  - come see me Friday **1pm- 3pm** LH 301h <<**TIME CHANGE**

Notes

- I was wrong about the UCB experiment in the book
- List of possible projects will be listed on canvas tomorrow
- A note about RL-glue

Today's outline

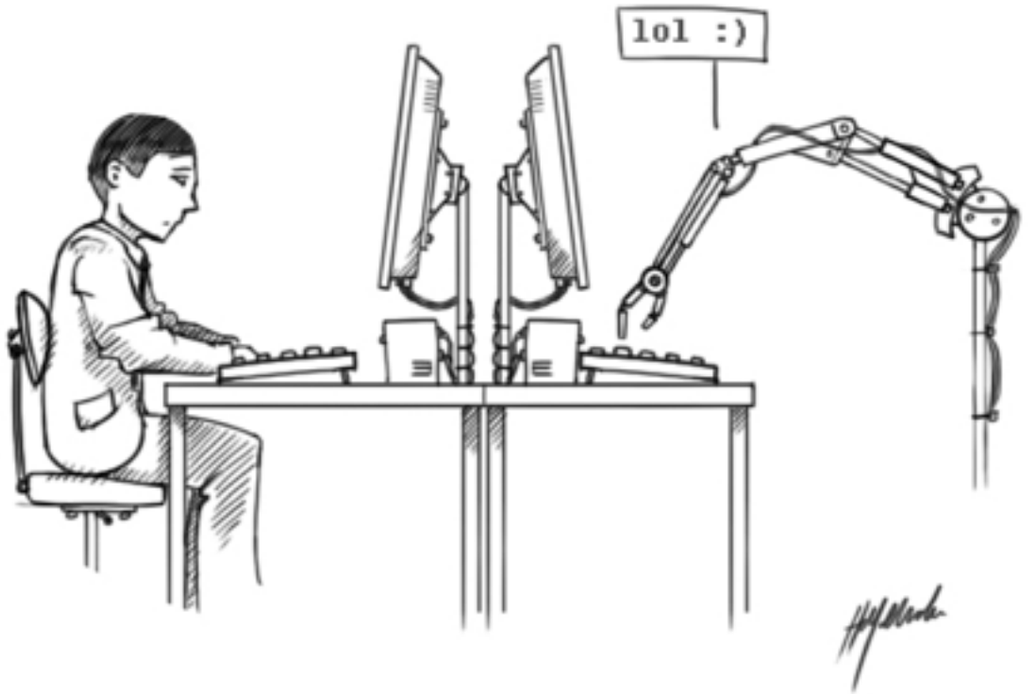
- More of your thought questions
- Beyond bandits: the full RL problem
- Goals, rewards, and returns
- Formalizing RL tasks as Markov Decision Processes
- The Markov property
- State-value functions, action-value functions
- Optimal value functions
- Bellman equations
- Lots of examples along the way

Some of your thought questions

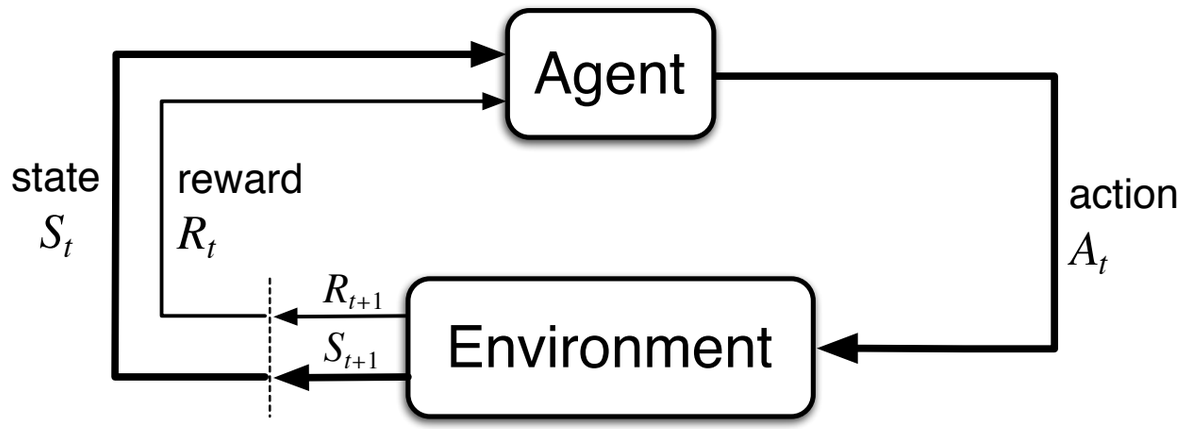
- How do we deal with state spaces that are too big to store in a table, or inputs that are continuous?
- Do reinforcement learning agents, interacting with the world ever overfit?
- Reinforcement learning is often applied to grid world like problems couldn't we use heuristic search methods like A*?

thought questions

- Could an RL agent pass the turning test?

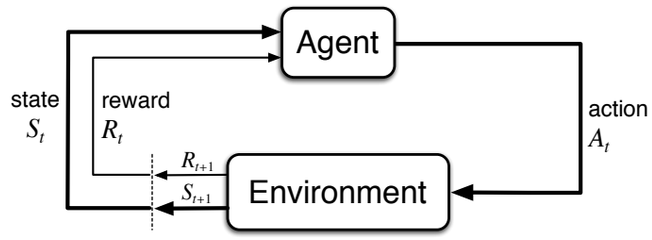


The agent-env interface, more formally



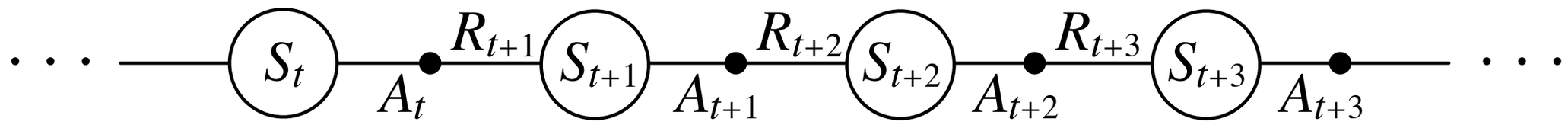
- Interaction occurs at discrete, time steps: t = 1,2,3,…
- Agent observes state on step t: $S_t \in \mathcal{S}$
- Agent produces an action on step t: $A_t \in \mathcal{A}(S_t)$
- *Environment* emits a Reward: $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$
- Environment, partially due to A_t transitions to new state: $S_{t+1} \in \mathcal{S}$

A stream of experience



- The agent-env interaction can produce an unending temporally correlated data stream:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, ...$$



The agent's policy

Policy at step $t = \pi_t =$

a mapping from states to action probabilities

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

Special case - *deterministic policies*:

$\pi_t(s) =$ the action taken with prob=1 when $S_t = s$

The agent learns a policy

- RL methods specify how the agent changes its policy as a result of it's experience: **learning**
- Informally, the agent's goal is to modify its policy to get as much reward in the long run
- Unlike the bandit task we have many states, and our policy isn't looking for one best action but the best action in every state
  - like facing a bandit task in every state
  - but choices in one state have long term consequences for future rewards and future states

Consider a robot

- The **actions** could be:
  - motor voltages, desired well velocities, higher level decisions like goto hallway

- The **states** could be:
  - low-level sensors like light and thermal sensors, X-Y-θ position, symbols encoding which room you are in

- **Rewards** could be:
  - motor-temperatures, time until reaching the hallway, provided by you

- In these examples the **time step** ranges from 100's of times a second, to minutes and may not fixed intervals

- The RL formalism is meant to be broad, to capture from low-level all the way to high-level abstract decision making

Our agent's subscribe to the reward hypothesis

> That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

- Can we think of counter examples?

Rewards

- The agent's goal is to maximize long term or cumulative reward
  - that is the agent's main goal in life
  - we call this goal-directed learning
- It is important we don't over-specify the reward
  - chess example in the book

Episodic Returns

- The agent seeks to maximized the **expected return**

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

- Here *T* denotes time when agent-env interaction ends in a special **termination state**
  - followed by a reset to a start state or distribution over state states
- In these situations the time breaks up into subsequences call **episodes** & we call the task **episodic**

Discounting and continuing tasks

- Another case there are no episodes, in these **continuing** tasks the interaction never ends; the defn: of return changes:

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

- $\gamma \in [0, 1)$ is called the discount factor
  - rewards are down-weighted exponentially with time
  - why do we need it?
- Rewards closer to now are worth more
  - dollar today more valuable than a dollar tomorrow …

Markov Decision Processes

- Roughly any RL task that has the Markov Property is an MDP
- A **finite MDP** is one where the S and A are finite
  - almost all theory is based on finite MDPs
- An MDP is defined by:
  - the state & action sets
  - one-step dynamics:

$$p(s', r|s, a) \doteq \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}$$

    - Given a state and action*:* probability of each possible next state and reward

The Markov Property

- By “state” we mean whatever information about the env that is available to the agent on each time step

- The state can be built up over-time

- Ideally, a state should summarize past sensations, retaining all **essential** information:

$$\mathbf{Pr}\{R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\} =$$

$$p(s', r|s, a) = \mathbf{Pr}\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\}$$

$$s\prime \in \mathcal{S}, r \in \mathcal{R}, \text{ and all histories } S_0, A_0, R_1, ..., R_t, S_t, A_t$$

- We call this the **Markov Property**

Markov Property

- If we have it, means we can predict future states and expected reward using just the current state and reward
  - do just as well as we would do if we stored the entire interaction history
  - similarly for selecting actions
- In practice we will write programs that encode the one-step dynamics implicitly. It is theoretical construct useful for defining Bellman equations

State

- The state is the information available to the agent on every time step, as defined by someone.
  - things like sensor readings on a robot, available at time t we will call **sensations**
- A person **defines** what the state is:
  - might be all the sensors readings
  - might be a history or window sensor readings (last 5 seconds)

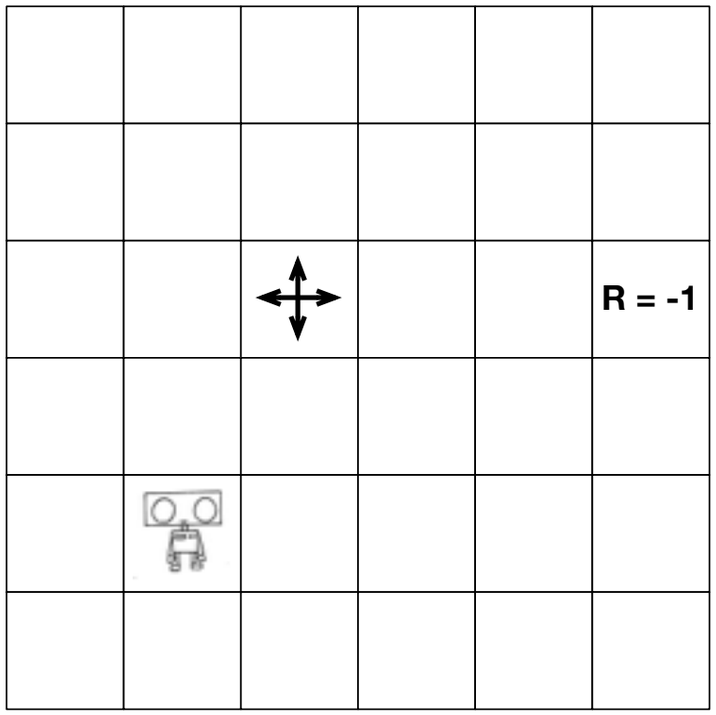
We are going to pay a game

- I will describe a problem to you
  - A decision making problem
- I will tell you **MY** definition of state, action & reward for the problem
- You tell me if its an MDP
  - if the problem has the Markov Property
  - this definition will help:

$$\mathbf{Pr}\{R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\} =$$

$$p(s', r|s, a) = \mathbf{Pr}\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\}$$

Lets play “is it an MDP?”



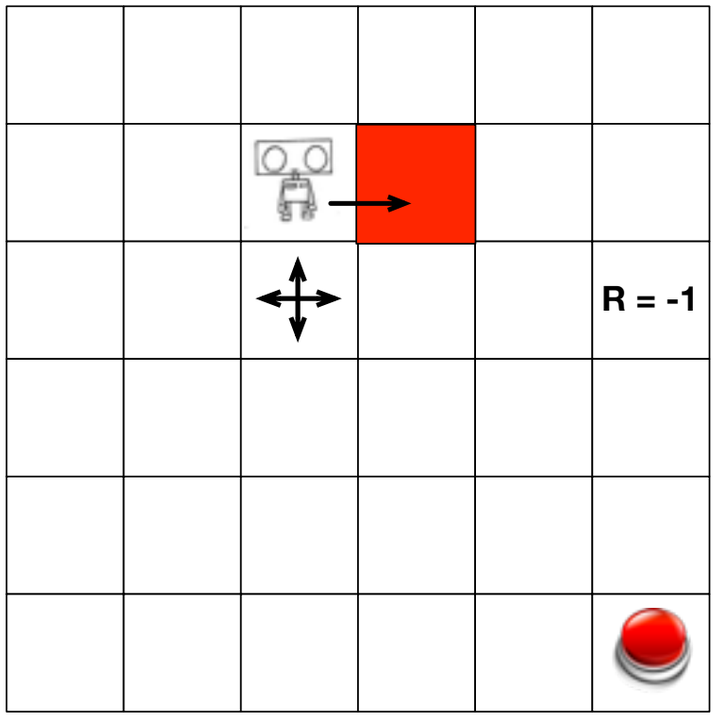
- deterministic transitions
- states are the cells
- no terminal states
- is it an RL task?
  - set of states, set of actions
- does it have the Markov property?

$$\mathbf{Pr}\{R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\} =$$

$$p(s', r|s, a) = \mathbf{Pr}\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\}$$

$$s\prime \in \mathcal{S}, r \in \mathcal{R}, \text{ and all histories } S_0, A_0, R_1, ..., R_t, S_t, A_t$$

How about this one. MDP?



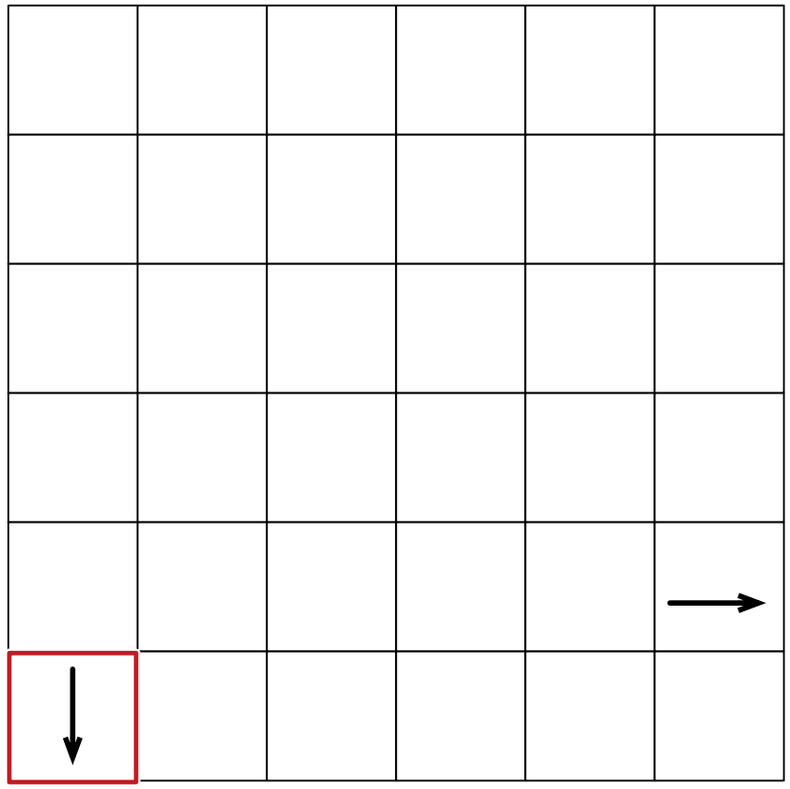
- one new action, **push button**
- if the button is pushed down, **and** later you enter **state 10**, the reward = **+10**, else -1
- state **does not include** if the button is pushed or not
- everything is the same
- does it have the Markov property?

$$\mathbf{Pr}\{R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\} =$$

$$p(s', r|s, a) = \mathbf{Pr}\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\}$$

$$s\prime \in \mathcal{S}, r \in \mathcal{R}, \text{ and all histories } S_0, A_0, R_1, ..., R_t, S_t, A_t$$

How about this one. MDP?



- state is **binary wall detector**
  - 1 iff facing a wall, 0 otherwise
  - orientation is not part of the state: 4 possible directions (NSEW)
- $R_t = 0$ on every transition
- two actions **forward** and **rotate** clockwise
  - forward action moves agent forward one step in the grid
- does it have the Markov property?
  - imagine you are the agent, you just have the wall detector

$$\mathbf{Pr}\{R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\} =$$

$$p(s', r|s, a) = \mathbf{Pr}\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\}$$

$$s\prime \in \mathcal{S}, r \in \mathcal{R}, \text{ and all histories } S_0, A_0, R_1, ..., R_t, S_t, A_t$$

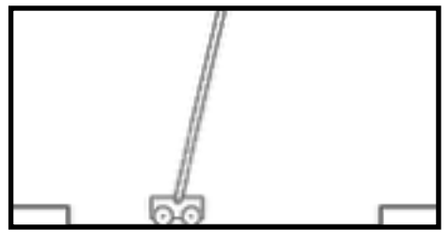
Markov does not mean we know everything there is to know about the world

- It just means our state does **not forget things** we have seen in the past that are relevant for the next state and reward

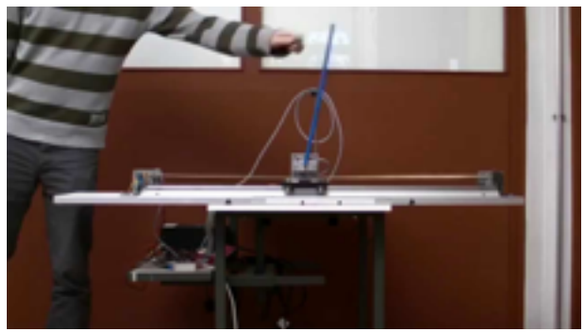
- Example: poker
  - knowing your cards, what you have discarded, the bets of the other players, your bets is Markov state. No **information** that was available on some time step was lost
  - You never had access to the other players cards so you could not have forgotten it

Sometimes we don't have Markov Property

- Cart-pole state: position & velocity of cart & angle & angular velocity of pole
  - in simulation this is Markov



- In a real physical system this is close but not Markov
  - sensors have noise and delay
  - physical phenomenon effect future states and rewards,
    - temperature of ball bearings
    - flexibility of pole etc etc

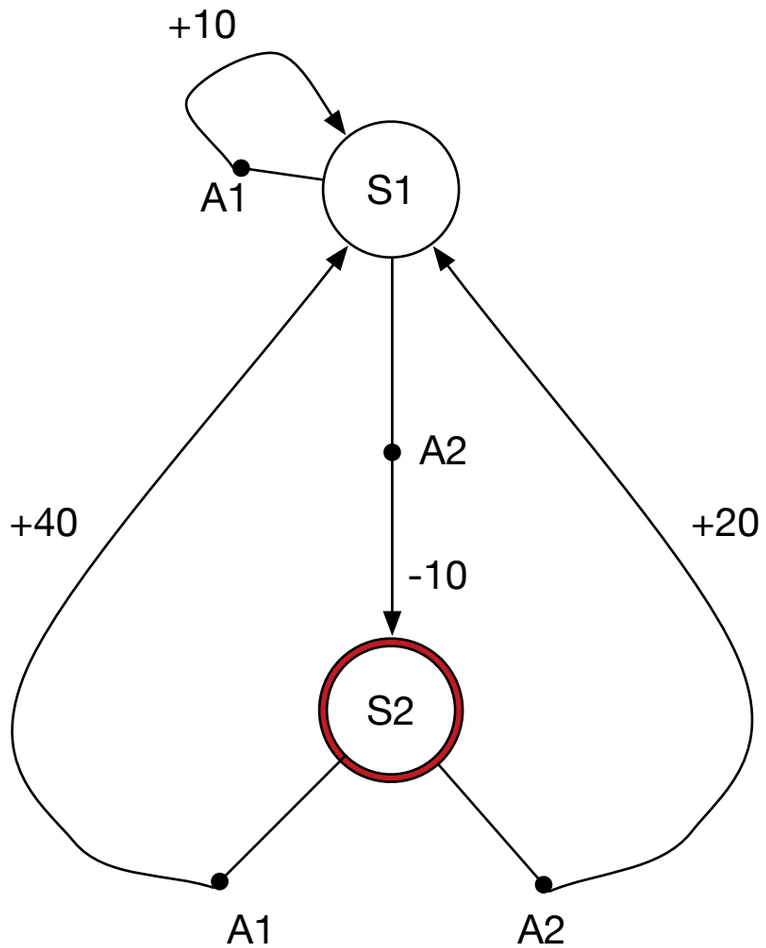


Our role

1. If someone gives us a problem
   - **they define** the dynamics, states, actions, rewards
   - **we** want to **identify** if the problem they gave us, the way the specified it is an MDP (with Markov property)
     - Because our **RL algorithms will work better** if the task is an MDP

2. If **we are formalizing** an RL task, we are designing the states, actions, rewards, & transition dynamics
   - we want to make sure our specification of the task is an MDP or close too it
   - why because it means RL algorithms will be more useful for its solution

MDP Example



- π(S1) = A2
  π(S2) = A1
  (deterministic policy)
- start in S2
- What is the interaction stream produced (states, actions, rewards)?

S2, A1, 40, S1, A2, -10, S2, A1, 40, …

Values & value functions

- The value of a state given a policy :

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \qquad v_\pi : \mathcal{S} \to \mathbb{R}$$

  - starting in s, if we select actions according to π, what is the expected return?

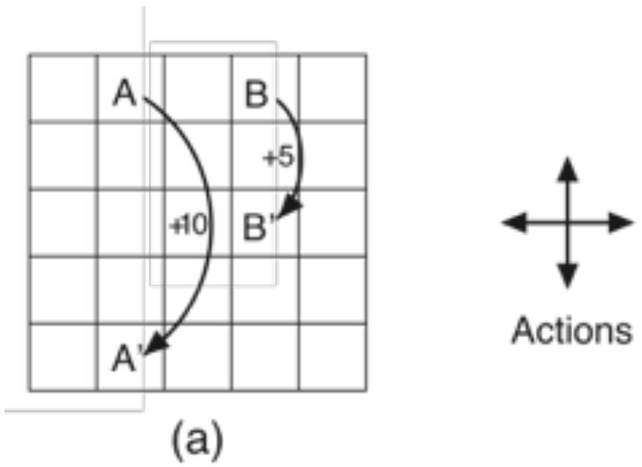
- The value of a state-action pair given a policy:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right] \qquad q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$

  - starting in s, if **first** select action a, then forever after select actions according to pi, what is the expected return?

Gridworld

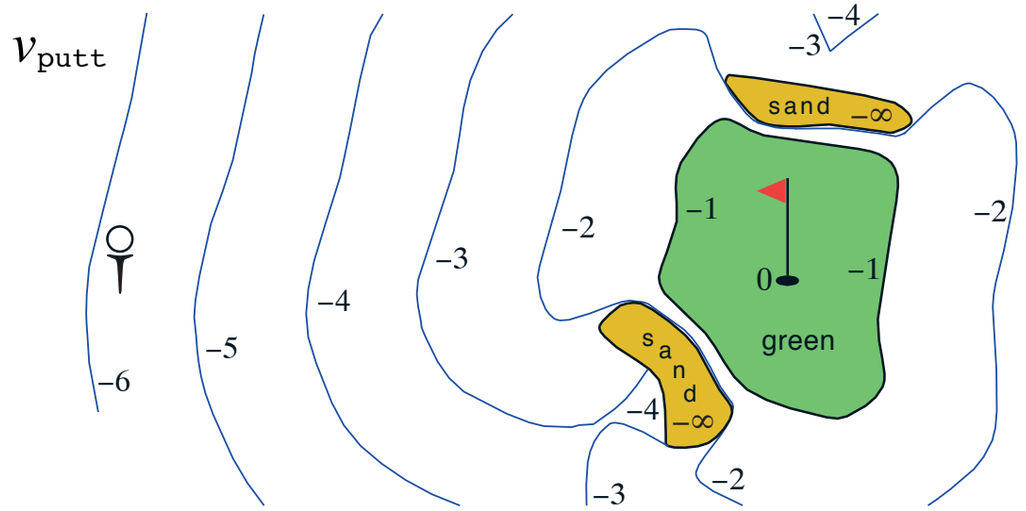
- Each cell is a state
- Actions: north, south, east, west
    - deterministic
- Cannot escape the world. Bump on walls, reward=-1
- Special transitions give +10, +5 rewards
- All other transitions reward = 0
- Continuing task



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

Golf

- State is ball location (grouped by contours of map)
- Reward is -1 for each hit
- Actions: putt and driver
- Episodic task
- Putt success:
  - zero in hole (terminal state)
  - one shot on green
  - can putt from one contour to another with one shot
  - sand trap is impossible to escape with putter



Optimal value functions

- The optimal value of a state, $v_*(s)$, is the state-value function corresponding to the optimal policy(s) denote by π^*:

$$v_\star : \mathcal{S} \to \mathbb{R}$$

- The state-action value function corresponding to π^*:

$$q_\star : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$

- What is an optimal policy?

Optimal value functions

- For finite MDPs, policies can be **partially ordered**:

$$\pi \geq \pi' \quad \text{if and only if } v_\pi(s) \geq v_{\pi'}(s) \text{ for all } s \in \mathcal{S}$$

- There are always one or more policies that are better than or equal to all others. Well call these **optimal policies**, π^*

- All optimal policies have the same optimal state-value functions:

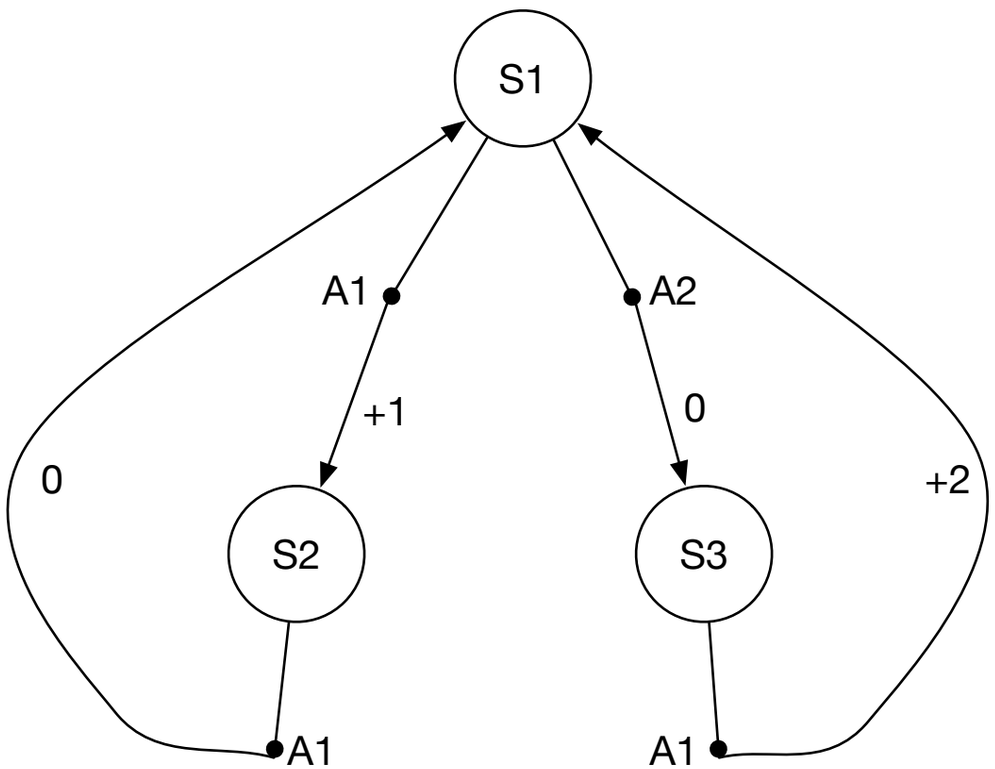
$$v_*(s) \doteq \max_\pi v_\pi(s) \qquad \text{for all } s \in \mathcal{S}.$$

- All optimal policies have the same optimal state-action value functions:

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a) \qquad \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}(s).$$

Another MDP Example

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

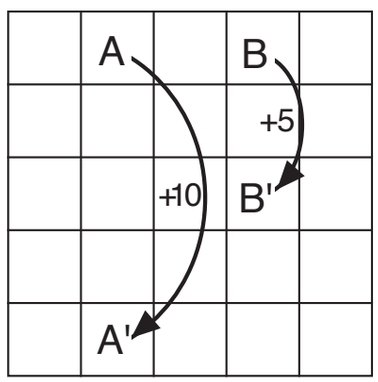


- What is the optimal policy if
- γ = 0?
- γ = 0.99
- γ = 0.5

Goal of learning maximize expected return by changing the policy

- One way to get an optimal policy:
  - act greedy with respect to v*
  - usually greedy means, myopic/short-sighted
  - magic of v_* is that it encodes long-term consequences!

- For example given v* and using a one step look-ahead we get long-term optimal actions



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) v_*

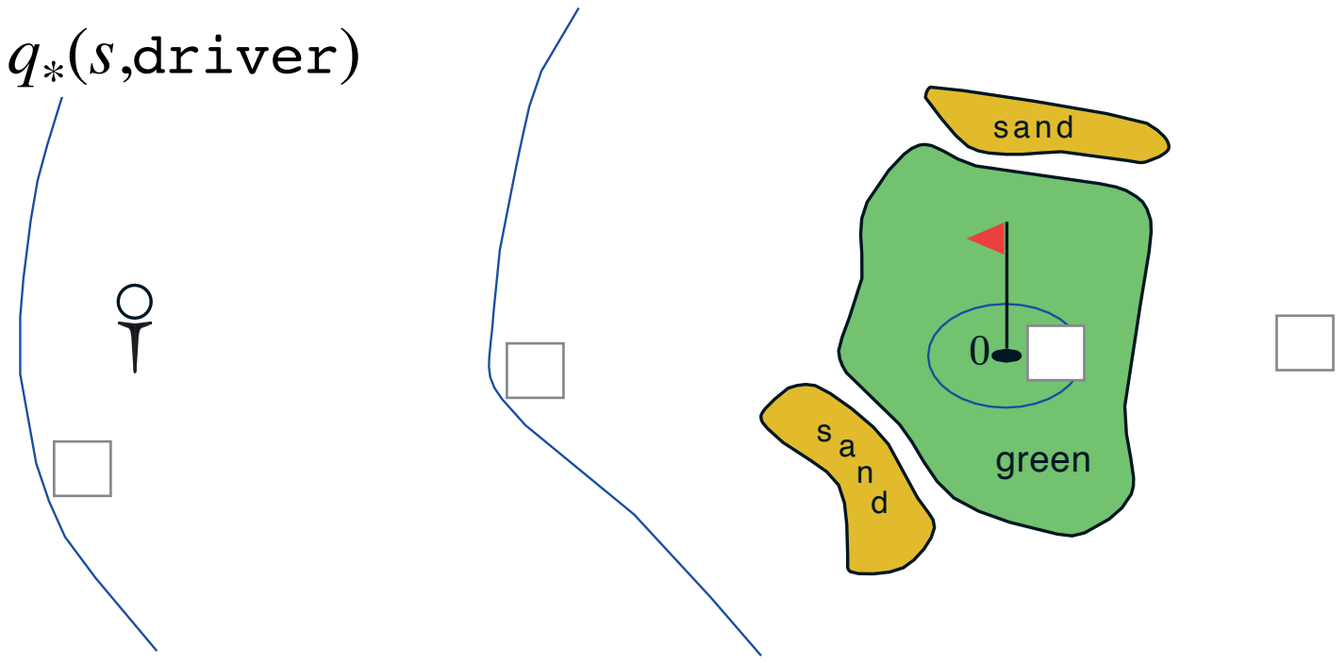
c) π_*

Any policy that is greedy wrt v_* is an optimal policy

- If someone, an oracle, gave you v* for some RL problem
- Then you can take actions according to one of the optimal policies. We don't have to know the optimal policy
- To do this we need v* and the one step dynamics
- Algorithm for acting optimally:
  1. one each step, in our current state: ***s***
  2. ask pr(s',r | **s**, a) what the possible next state is for each action in **s**: produces a list of possible next states $s'_1, s'_2, \ldots$
  3. for each possible next states query v_*
  4. pick the action that corresponds to the next state with max v_*

Optimal action-value function for golf

- We can hit the ball further with the drive, but with less accuracy;
  - only get in the hole in one shot on a small part of the green
- **What is the value of using the driver first (in each contour), then following π* (picking whichever is best) there after?**



Usage of value functions

	state values	action values
prediction	v_π	q_π
control	v_*	q_*

- Like the MDP dynamics, these are theoretical objects

- The things we actually learn are called our **approximate state & state-action value functions**, estimated from data

$$V_t(s) \qquad Q_t(s, a)$$

Optimal state-action value function

- Given q*, the agent does not need a one-step look-ahead, as we do with v*

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

- Most of our control (policy learning) algorithms will use approximate $Q_t(s,a)$ functions

Bellman equations for π

- The definition of the return is recursive:

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

- Therefore:

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

- Without expectation:

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

Set of equations

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

- This is a set of equations, one for each state
- Linear system of equations
- v_π is the unique solution to this system of equations
- **Key point:** the value of the start state is equal to the value of the expected next state + expected reward along the way

There is also a Bellman equation for the optimal state-value function

- The value of a state under π_* must equal the expected return for the best action best action from that state:

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

- No reference to any specific policy?

- In this case v_* is the unique solution, for this system of **non-linear** equations

Why do Bellman Equations Matter?

- They form the basis of learning algorithms:
  - approximating, computing v_π

Solving the RL problem

- We could solve the Bellman optimality equation, get q* and act greedy
- To do so we need:
  - pr(s',r | s,a) — one-step dynamics model
  - computation and storage
  - Markov property

- Could we do Computer Go this way?
  - perfect information game, we have a perfect model
  - current board configuration is Markov state
  - But, estimated that the # of possible games far exceeds the # of atoms in the universe & 10^170 states
  - Dynamic programing methods (Chapter 4) require computation that is polynomial in the # of states!

Approximation is our friend

- We don’t have a choice, we must approximate
  - Go is but one of many things that humans do well
  - our computation is always limited
- Thus we rarely learn *the optimal policy*
  - nevertheless a notion of optimality is useful designing algorithms and understanding their behavior theoretically

- RL methods can be thought of as approximately solving the Bellman Optimality Equation
  - using the relationship between values of successive states, v(s), v(s’)

Exercise

Exercise 3.9 In the gridworld example, rewards are positive for goals, negative for running into the edge of the world, and zero the rest of the time. Are the signs of these rewards important, or only the intervals between them? Prove, using (3.2), that adding a constant c to all the rewards adds a constant, v_c, to the values of all states, and thus does not affect the relative values of any states under any policies. What is v_c in terms of c and γ?

∞

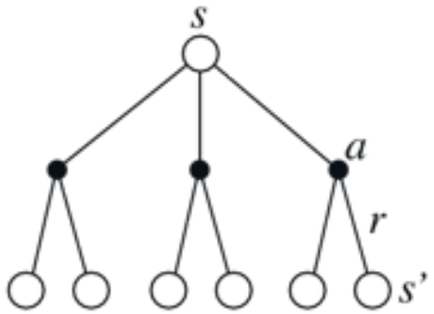
$$G_t = \sum_{k=0} \gamma^k (R_{t+k+1} + C)$$

Exercise 3.10 Now consider adding a constant c to all the rewards in an episodic task, such as maze running. Would this have any effect, or would it leave the task unchanged as in the continuing task above? Why or why not? Give an example.

Imagine a grid world where the goal is to reach the terminal state as fast as possible; Rt = -1

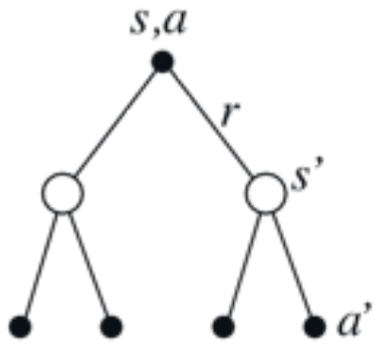
Backup diagrams

- I visualization tool for understanding and remembering the Bellman equations



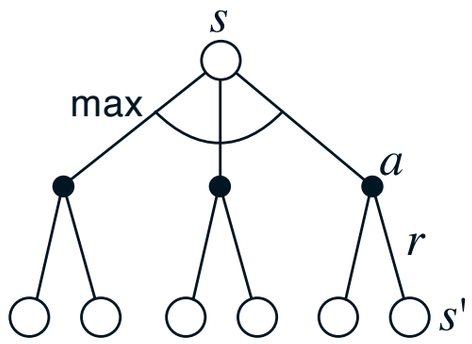
for v_π

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



for q_π

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



$$v_\star(s) = \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\star(s') \right]$$