Reinforcement Learning

Dynamic Programming

Prof. James Brusey

June 4, 2023

Why study Dynamic Programming?

- DP assumes perfect model and is computationally expensive
- Still important theoretically, though
- Basis for understanding

Expectation (reminder)

- Expected value is the average outcome of a random event given a large number of trials
- Where all possible values are equally likely, this is simply the mean of possible values
- Where each x_i occurs with probability $p(x_i)$, we have the sum

$$\mathbb{E}[x] = \sum_i p(x_i)x_i$$

- Quick quiz: what's the expected value of a fair, 6-sized dice?

Expectation with a subscript?

- When we have a subscript, that means according to that probability measure

$$\mathbb{E}_y[x] = \sum_i p_y(x_i) x_i$$

where p_y is a probability measure for x that might differ from the sampling probability.

Bellman optimality equation

- Dynamic programming is based on solving Bellman's optimality equation

$$v_*(s) = \max_a \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right]$$

- why is this equation so important and what do the variables mean?

Policy evaluation (prediction)

Evaluation asks:

- Given a policy π, what is the expected value v_π of each state s?

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

Policy evaluation (2)

- We can substitute for the G_t based on

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

or more simply

$$G_t = R_{t+1} + \gamma G_{t+1}$$

Policy evaluation (3)

We should get

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

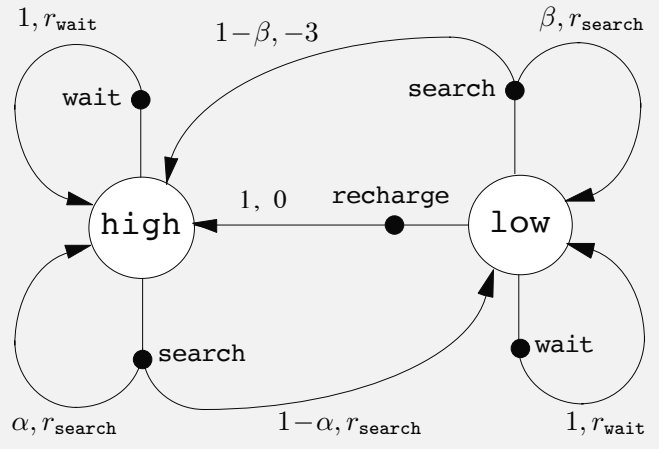
where

- $p(s', r|s, a)$ is the transition probability
- $\pi(a|s)$ is the probability of taking action a when in state s
- r is the immediate reward
- γ is the discount factor

Policy evaluation (4)

For this simple recycle robot, write down the set of equations for $v_\pi(s)$ assuming that π is a uniform distribution over actions.

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1-\alpha$	$r_{\texttt{search}}$
low	search	high	$1-\beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	-



Policy evaluation (5)

Here is the equation for $v_\pi(\text{high})$:

$$\begin{aligned} v_\pi(\text{high}) &= \frac{1}{2}(r_{\text{wait}} + \gamma v_\pi(\text{high})) + \\ & \quad \frac{1}{2}\Big(\alpha(r_{\text{search}} + \gamma v_\pi(\text{high})) + \\ & \quad (1-\alpha)(r_{\text{search}} + \gamma v_\pi(\text{low}))\Big) \end{aligned}$$

LP solution

Note that we have equations in the form:

$$v(a) = k_1 v(a) + k_2 v(b) + k_3$$

which can be converted into

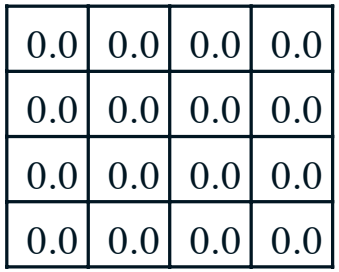
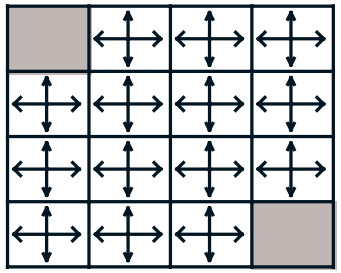
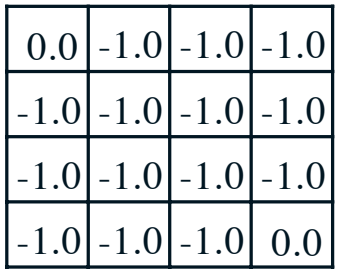
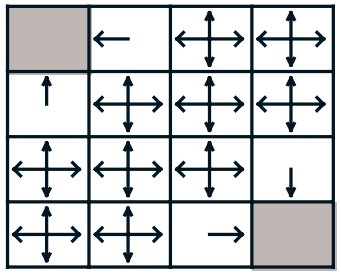
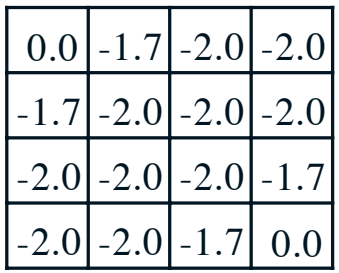
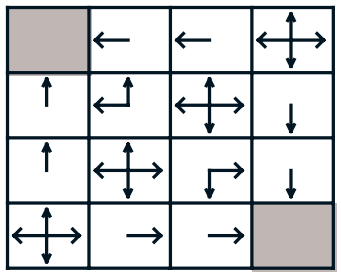
$$-k_3 = (k_1 - 1)v(a) + k_2 v(b)$$

or

$$\begin{pmatrix} -k_3 \\ \vdots \end{pmatrix} = \begin{bmatrix} (k_1 - 1) & k_2 \\ \vdots & \vdots \end{bmatrix} \mathrm{v}$$

Which can be solved by inverting the matrix. *Do try this at home!*

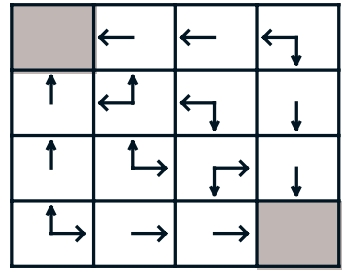
Iterative policy evaluation

	v_k for the random policy	greedy policy w.r.t. v_k
$k = 0$		 random policy
$k = 1$		
$k = 2$		

Iterative policy evaluation

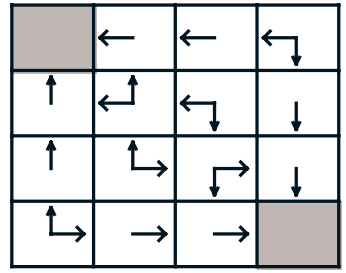
$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



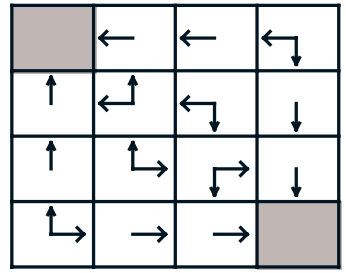
$k = 10$

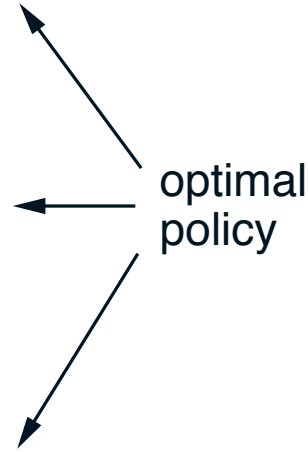
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0



$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0





Approximate methods

- It is also possible to find the solution *iteratively* by starting off with some guesses for v and updating according to the equations.
- This approach forms the basis of many other methods and for large MDPs we will never attempt using LP

Policy improvement

- Given that we can evaluate any policy, we can therefore *improve* it by updating the policy so that it takes the action that maximises the resulting value
- We define the expected value of taking an action a and then following π from then on

$$q_\pi(s, a) \doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$$

- which expands to

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

Policy improvement (2)

- If we can find some a for which $q_\pi(s, a) > v_\pi(s)$, then we can *improve* π by adjusting it to use a
- If we cannot find any improvement for all states then π must be optimal
- Discuss:
  - what aspects of MDPs are relied on to ensure this is true?

Policy iteration algorithm

S1: Initialisation

S2: Policy evaluation

- Loop:
  - $\Delta \leftarrow 0$
  - Loop for each $s \in S$:
    - $v \leftarrow V(s)$
    - $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
    - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  - until $\Delta < \theta$

S3: Policy Improvement

- *policy-stable* $\leftarrow 1$
- For each $s \in S$:
  - *old-action* $\leftarrow \pi(s)$
  - $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
  - If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow 0$
- If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Exercise: Policy iteration for action values

Revise the algorithm on the previous page to use action values q instead and thus find q_*.

Value iteration

- A problem with policy iteration is the need to perform policy evaluation to some accuracy before improving the policy.
- Value iteration skips finding the policy and just updates the value to the maximum value

$$v(s) \leftarrow \max_a \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s, A_t = a]$$

Value iteration

- A problem with policy iteration is the need to perform policy evaluation to some accuracy before improving the policy.
- Value iteration skips finding the policy and just updates the value to the maximum value

$$v(s) \leftarrow \max_a \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s, A_t = a]$$

- The algorithm terminates when the size of the updates to any $v(s)$ is smaller than some threshold

Phew!