REINFORCEMENT LEARNING

Overview

- Last lecture looked at inductive learning
  - How to learn rules given examples of decisions.
- Supervised learning = examples of correct behavior.
- Often we don't have such examples.
- Just know when we succeed or fail.
- This is the domain of *reinforcement learning* (RL).

- First section of the lecture leans heavily on:

  R. Sutton and A. Barto, *Reinforcement learning*, MIT Press.

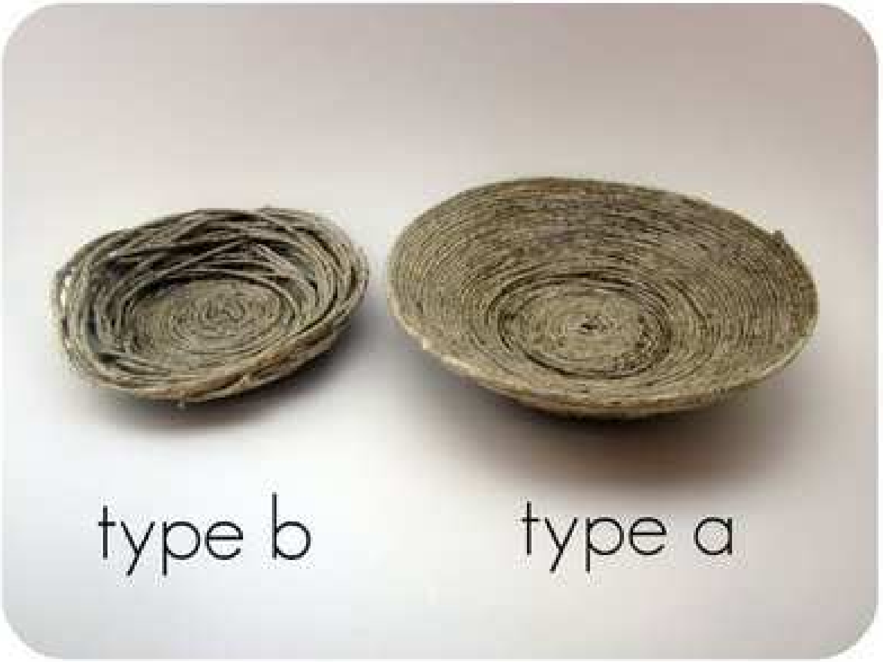
- A book I thoroughly recommend.

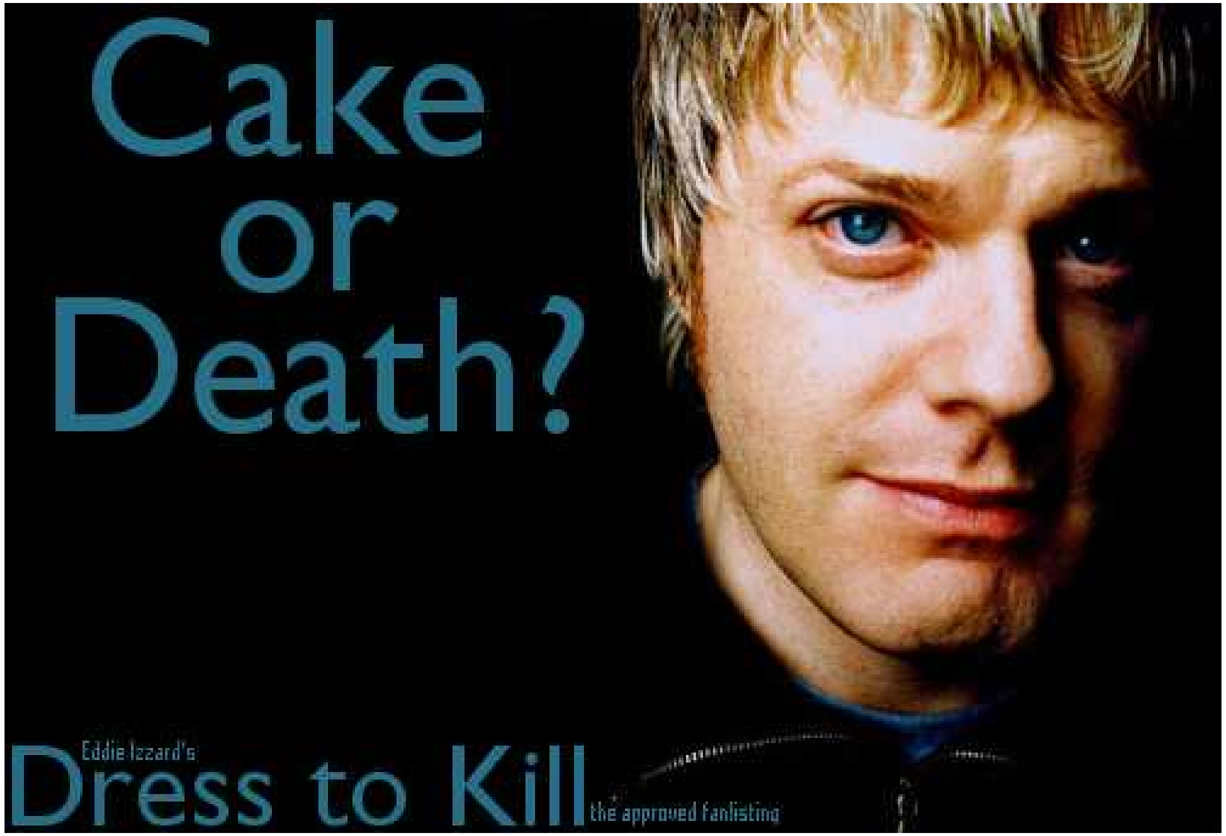
Example

- We'll start by playing a pair of slot-machines:

- Choice is between playing machine *A* and machine *B*.

- What do you choose?




Cake
or
Death?
Eddie Izzard's
Dress to Kill
the approved fanlisting

n-armed bandits

- How can we formally analyze this kind of situation?
  - *n*-armed bandit
- Classify problems as:
  - *Evaluatve* vs *instructive* feedback

    Evaluative depends totally on the action you take.

    Instructive (tells you the right action) does not.
  - *Associative* vs. *nonassociative* learning

    Assocative maps inputs to outputs and learns the best output for each input

    Nonassociative learns the one best output.
- *n*-armed bandit is nonassociative and has evaluative feedback.

- Choose repeatedly from one of n actions
  - Each choice is called a *play*
- After each play a_t , you get a reward r_t , where:

$$E\langle r_t | a_t \rangle = Q^*(a_t)$$

  These are unknown action values.
- Distribution of r_t depends only on a_t.
- Objective is to maximize the reward in the long term
  - Say over 1000 plays
- To solve the n-armed bandit problem, you must *explore* a variety of actions and then *exploit* the best of them

Exploration vs. exploitation

- Suppose you form *action value estimates*:

$$Q_t(a) \text{ which estimates } Q^*(a)$$

  which try to say what each action is worth at each point in time.

- The *greedy* action at t is

$$a_t^* = \arg\max_a Q_t(a)$$

- Picking a_t^* is exploitation.
- Picking $a_t \neq a_t^*$ is exploration.
- You can't exploit all the time; you can't explore all the time.
- You can never stop exploring; but you should always reduce exploring.

- Who knew The North Face were experts in reinforcement learning.

Action-value methods

- Methods that adapt action-value estimates and nothing else.
- For example suppose by the t-th play, action a had been chosen k_a times, producing rewards.

$$r_1, r_2, \ldots, r_{k_a}$$

then:

$$Q_t(a) = \frac{r1 + r2 + \ldots + r_{k_a}}{k_a}$$

- This is just the sample average of the reward.
- We have:

$$\lim_{k_a \to \infty} Q_t(a) = Q^*(a)$$

Aside

- We can compute the average reward incrementally.
- If Q_k is the average of the first k rewards and r_{k+1} is the $k+1$th reward, then

$$
\begin{aligned}
Q_{k+1} &= \frac{1}{k+1}\sum_{i=1}^{k+1} r_i \\
&= \frac{1}{k+1}\left(r_{k+1} + \sum_{i=1}^{k}\right) \\
&= \frac{1}{k+1}\left(r_{k+1} + kQ_k + Q_k - Q_k\right) \\
&= \frac{1}{k+1}\left(r_{k+1} + (k+1)Q_k - Q_k\right) \\
&= Q_k + \frac{1}{k+1}\left(r_{k+1} - Q_k\right)
\end{aligned}
$$

- Given an estimate of a reward for each action, we than have to decide what to do.
- Greedy action selection:

$$a_t = a_t^* = \arg\max_a Q_t(a)$$

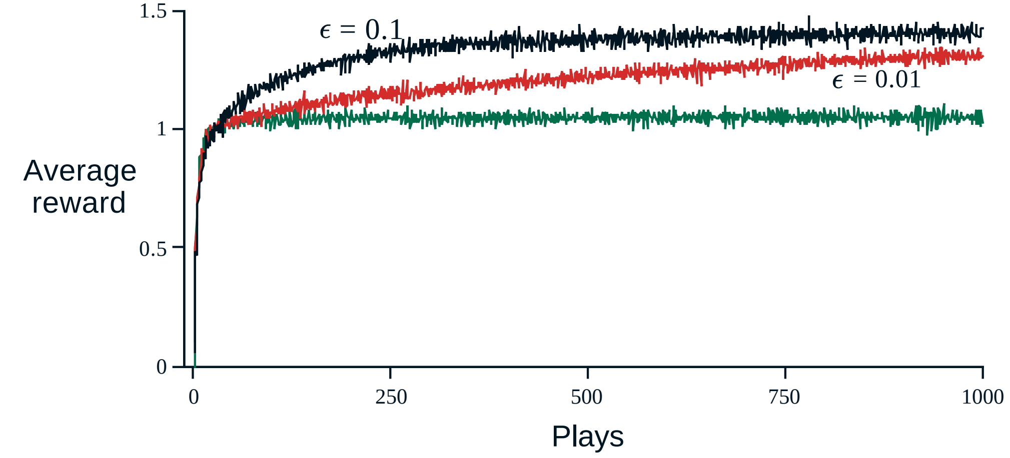
- ϵ-greedy action selection:

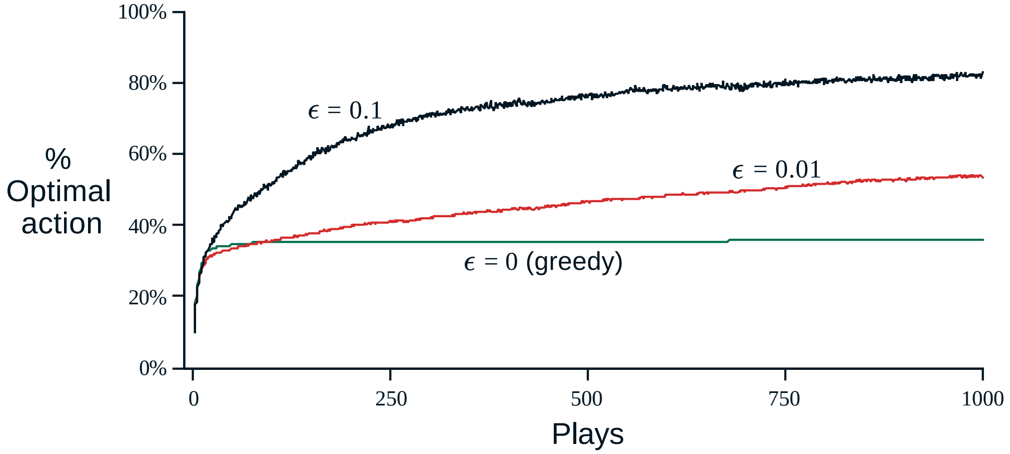
$$a_t = \begin{cases} a_t^* & \text{with probability } 1 - \epsilon \\ \textit{random action} & \text{with probability } \epsilon \end{cases}$$

- The simplest way to try to balance exploration and exploitation.

$\boxed{\text{10-armed bandit}}$

- $n = 10$ possible actions.
- Each $Q^*(a)$ is chosen randomly from a normal distribution: $\eta(0, 1)$.
- Each r_t is also normal $\eta(Q^*(a_t), 1)$.
- 1000 plays.
- Repeat the whole thing 2000 times and average the results.


1.5
1
0.5
0
Average reward
ϵ = 0.1
ϵ = 0.01
0
250
500
750
1000
Plays


100%
80%
60%
40%
20%
0%
% Optimal action
ϵ = 0.1
ϵ = 0.01
ϵ = 0 (greedy)
0
250
500
750
1000
Plays

- ϵ-greedy makes a random choice among the non-optimal actions.
- Sometimes, it is good not to pick actions with really poor outcomes.
- *Softmax* picks non-optimal actions based on their reward.

- Common to use the Gibbs distribution to pick the action.
- Chooses action a on the rth play with probability:

$$\frac{e^{\frac{Q_t(a)}{\tau}}}{\Sigma_{b=1}^{n} e^{\frac{Q_t(b)}{\tau}}}$$

  where τ is the *temperature*.
- When temperature is high, all actions are approximately equally likely.
- A temperature tends to 0, action selection tends to greedy selection.

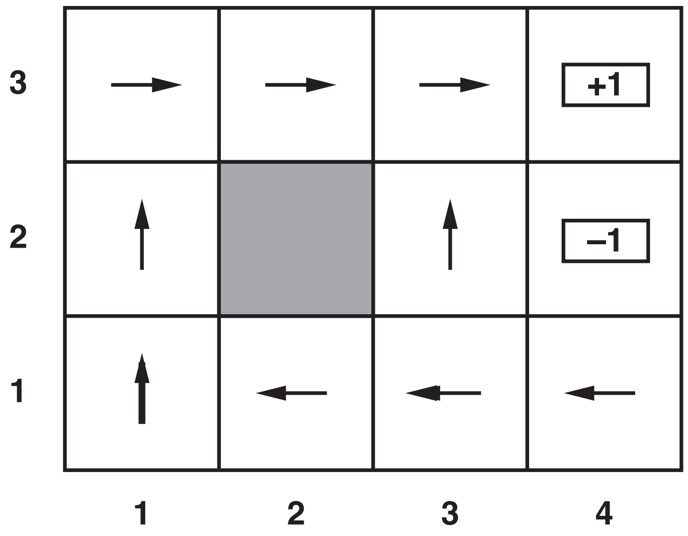
- Can vary τ over time — high to start, low as the agent thinks it is converging.
  - Look at received payoff

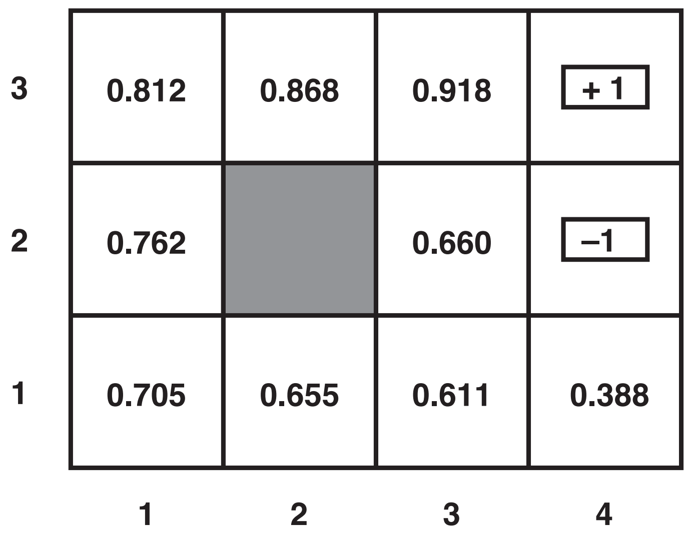
More complex scenarios

- The bandit model makes a key simplifying assumption
  - The agent is always in the same state.
- So we only have to learn about one action.
- In general, agents can be in multiple states, and the best action varies with state.
- In general, the agent faces an MDP.
  - But the parameters of the MDP are unknown.

Passive learning

- Remember this world which we solved as an MDP:





- In passive learning the agent's policy is fixed:
  - In state s it always executes $\pi(s)$.
- It has to learn the utility function $U^{\pi}(s)$.
- Comparing with the MDP case, the agent doesn't know the transition model:

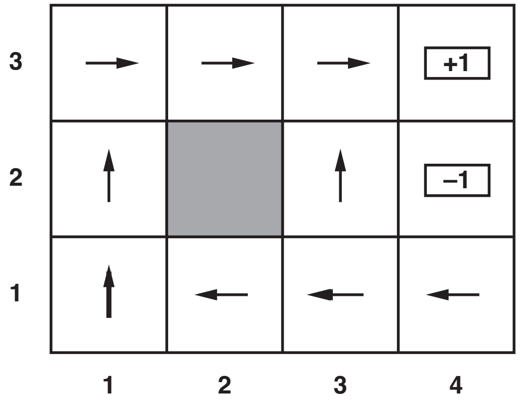
$$P(s'|s, a)$$

  and it doesn't know the reward function

$$R(s)$$

- How can it learn them?

- It learns them by carrying out runs through the environment.



- As ever, a run is a sequence of states and actions that continues until the agent reaches the terminal state:

$$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow$$

- Note that we have reward as well.

- As the agent moves it can calculate a sample estimate of $P(s'|s, \pi(s))$
  - Each time it moves it creates a new sample for one state.
- Each reward is a contribution to the computation of utility.

- We could estimate the utility of a state by the rewards generated along the run from that state.
  - Direct utility estimation.
- Thus a sample reward for $(1, 1)$ from the run above is the sum of the rewards all the way to a goal state.
- The same run will produce two samples for $(1, 2)$ and $(1, 3)$.
- You can do the calculation with or without discount.

Adaptive dynamic programming

- We can improve on the direct estimation by remembering the Bellman equation for a fixed policy:

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

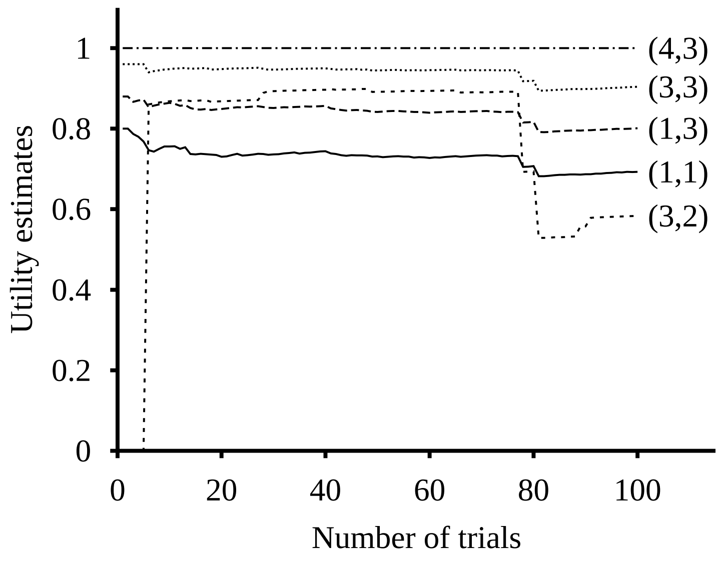
- The utility of a state is the reward for being in that state plus the expected discounted reward of being in the next state.
- This is the formula from page 33 of the notes for Lecture 8.

- Since we are using the fixed policy version of the Bellman equation we don't have the $\max$ that makes the original so hard to solve.
- Can just plug results into an LP solver
  - As we discussed when talking about policy iteration.
- Can also use value iteration, using:

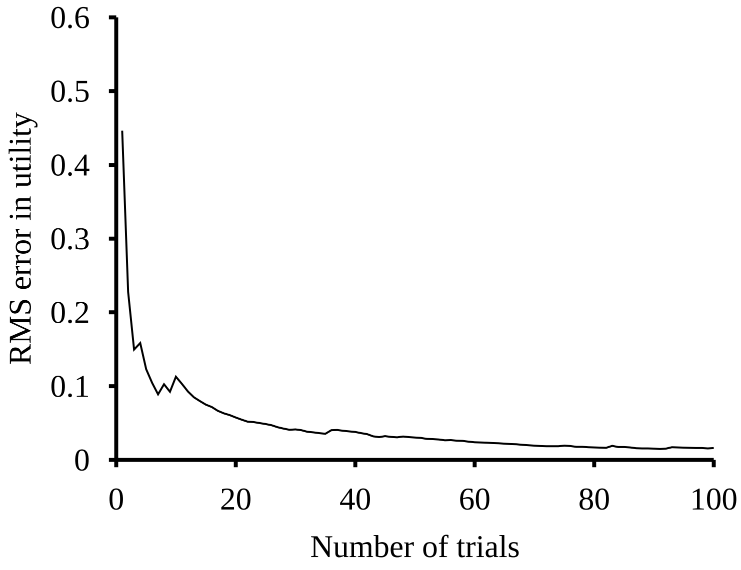
$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi(s))U_i(s')$$

  to update utilities.
- (We do this in policy iteration also — we just ran out of time to talk about it).

- Results:



Utility estimates
1
0.8
0.6
0.4
0.2
0
0
20
40
60
80
100
Number of trials
(4,3)
(3,3)
(1,3)
(1,1)
(3,2)



RMS error in utility
0.6
0.5
0.4
0.3
0.2
0.1
0
0
20
40
60
80
100
Number of trials

- Now, to get the utilities, the agent started with a fixed policy, so it always knew what action to take.
- It used this to get utilities.
- Having gotten the utilities, it could use them to choose actions.
  - Just picks the action with the best expected utility in a given state.
- However, there is a problem with doing this.
- What is it?

- Might not yet have experienced the bad effects of an action:

- Textbook uses the example of successfully running a red light.
- Of course, this kind of over-reliance on not-full-explored state/action spaces is what people do all the time.

- In addition, as the textbook points out, there are ways to get around this.
- There is no way to be sure that the action your reinforcement learner is picking doesn't have possible bad outcomes.
- But there are ways to try to mitigate the issue.

Active reinforcement learning

- The passive reinforcement learning agent is told what to do.
  - Fixed policy
- An active reinforcement learning agent must decide what to do.
- We'll think about how to do this by adapting the passive learner.
- We can use exactly the same approach to estimating the transition function.
  - Sample average of the transitions we observe.
- But computing utilities is more complex.

- When we had a policy, we could use the simple version of the Bellman equation:

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s))U^{\pi}(s')$$

- When we have to choose actions, we need to solve the full Bellman equation:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s, a)U(s')$$

  with its pesky $\max$.

- What to do?

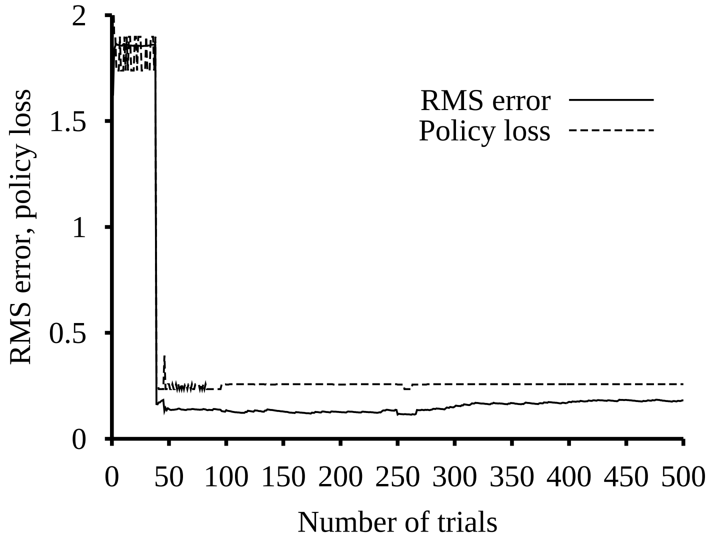
- Well, we know what to do, we use value iteration.
- At any stage, we can run:

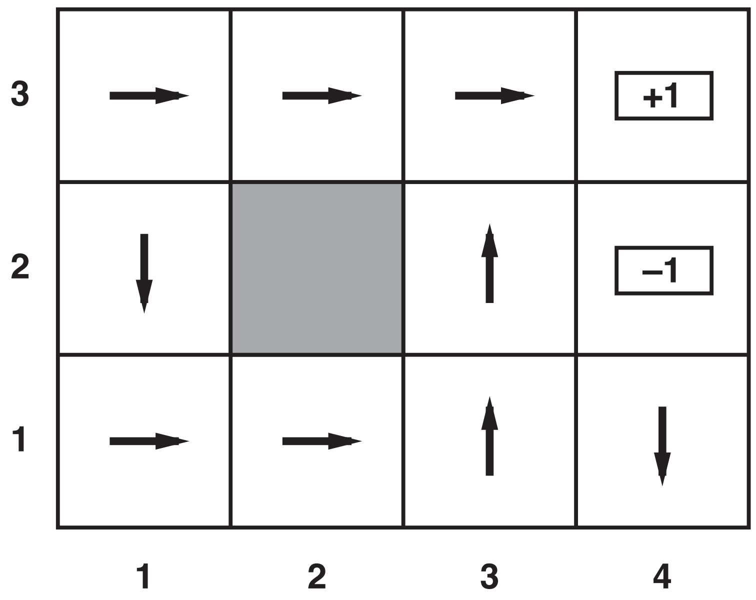
$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$$

to stability to compute a new set of utilities.

- Deciding what to do, what action to take, is the next issue.
- Normally after running value iteration we would choose the action with the highest expected utility.
  - Greedy agent
- This turns out not to be so great an idea.

- Typically a greedy agent will not learn the optimal policy:





- The issue is that once the agent finds a run that leads to a good reward, it tends to stick to it.
  - It stops exploring.
- To do better we can go back to the idea of ϵ-greedy exploration/exploitation.
  - As we saw earlier these can be slow.
- A better approach is to change the estimated utility assigned to states in value iteration.

- For example we can use:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} f\left(\sum_{s'} P(s'|s,a)U_i(s'), N(s,a)\right)$$

where $N(s,a)$ counts how many times we have done a in s, and $f(u,n)$ provides an exploration-happy estimate of the utility of a state.

- For example:

$$f(u,n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

R^+ is an optimistic reward, and N_e is the number of times we want the agent to be forced to pick an action in every state.

Q-learning

- Q-learning is a *model-free* approach to reinforcement learning.
  - It doesn't need to learn $P(s'|s,a)$.
- Revolves around the notion of $Q(s,a)$, which denotes the value of doing a in s.

$$U(s) = \max_a Q(s,a)$$

- We can write:

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) max_{a'} Q(s',a')$$

and we could do value-iteration style updates on this.

(Wouldn't be model-free)

- However, we can write the update rule as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

  and recalculate everytime that a is executed in s and takes the agent to s'.
- α is a learning rate, just like the learning rate in linear regression.
  - Controls how quickly we update the Q-value when we have new information.

Summary

- This lecture looked at reinforcement learning.
  - Learning when the agent receives periodic rewards and has to use these to figure out what to do.
- We started with bandit problems.
- Then we looked at multi-state problems, initially looking at learning the model when we had a fixed policy.
- We moved on to look at active learning, where the agent has to decide what to do.
- Finally we considered model-free learning.