

# Reinforcement Learning

## Ch. 21 (sections 1-3)

# Last time

- Markov Decision Processes
- Calculating the optimal policy

# Today

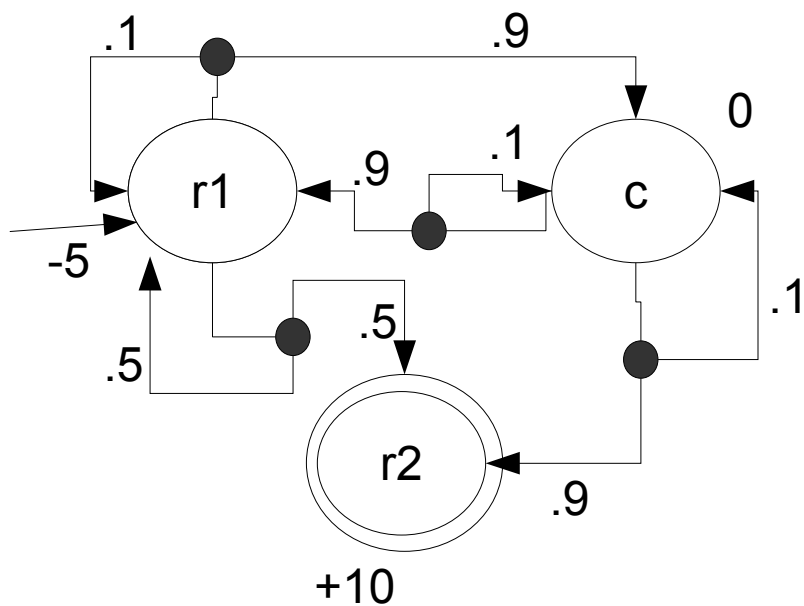
- Example of value iteration and policy iteration
- What happens when the transition model (and maybe the reward function) is unknown
- Reinforcement learning
  - “Imagine playing a new game whose rules you don't know; after 100 or so moves, your opponent announces: “You lose.” This is reinforcement learning in a nutshell.” (AIMA p.764)

# Value iteration: calculating $\pi^*$

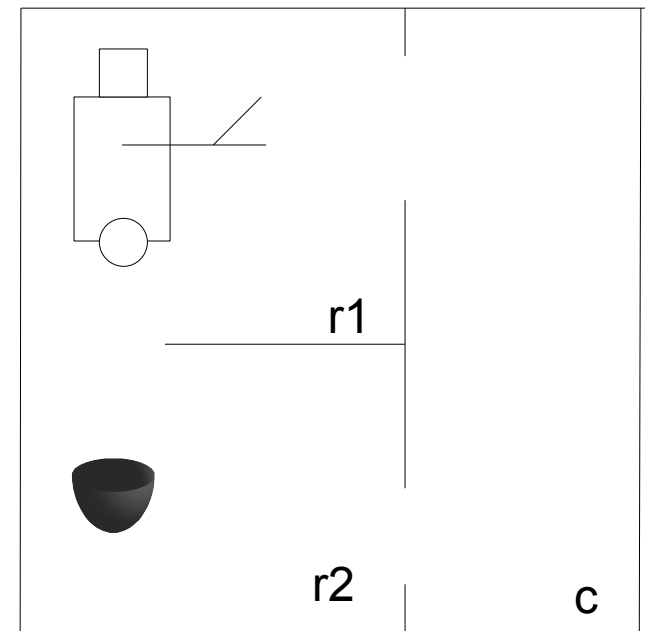
Starting with random utilities,

$$U_{t+1}(s) := R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_t(s')$$

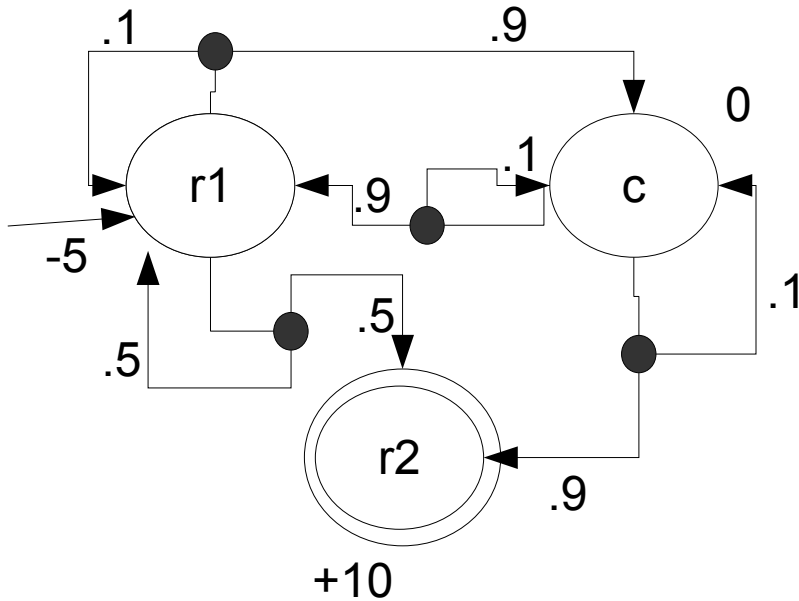
... until utilities converge to an equilibrium (differences below a small threshold number)



self-transitions out of r1 and c not shown  
(success with probability 1)



# Value iteration example



Starting with random utilities:

$$U_0 = \langle 2, -3, 1 \rangle$$

apply:

$$U_{t+1}(s) := R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_t(s')$$

$$T(r1, a, s') = \begin{matrix} 1 & 0 & 0 \\ .5 & .5 & 0 \\ .1 & 0 & .9 \end{matrix}$$

$$T(c, a, s') = \begin{matrix} .9 & 0 & .1 \\ 0 & .9 & .1 \\ 0 & 0 & 1 \end{matrix}$$

# Policy iteration: calculating $\pi^*$

- If one action is clearly better than all others, the actual state values need not be precise
- Policy iteration
  - 1. **policy evaluation**: given  $\pi_i$ , calculate  $U_i = U^{\pi_i}$
  - 2. **policy improvement**: calculate new policy  $\pi_{i+1}$  that maximizes expected utility using one-step lookahead based on  $U_i$

# Dynamic programming

- Value iteration is an algorithm in the dynamic programming (DP) family: recursive methods to find optimal solutions for problems with *overlapping* subproblems
- DP formulation:
  - 1. Break down the problem into smaller subproblems
  - 2. Solve those with DP recursively
  - 3. Compose optimal solution out of optimal sub-solutions
- DP examples we have already seen:
  - logic: forward chaining, backward chaining with caching of sub-goals
  - Bayes Nets: variable elimination (reuse factors)
  - DBNs: filtering, smoothing (forward-backward), and Viterbi algorithms (reuse messages)

# Limitations

- Assumes full knowledge of the process (transition model and reward function)
- Intractable for large state-spaces
  - e.g., backgammon has  $10^{50}$  states, which would require  $10^{50}$  Bellman equations in  $10^{50}$  variables

# Reinforcement learning

- Solving the MDP problem with no knowledge of transition models and reward functions

# Reinforcement/reward signals

- Learning to perform a goal-based task (e.g. finding your way out of a maze):  $R = 1$  if goal achieved
- Learning to crawl:  $R =$  forward progress
- Learning to play a game that keeps score (e.g., ping-pong or scrabble):  $R =$  score difference

# Assumptions

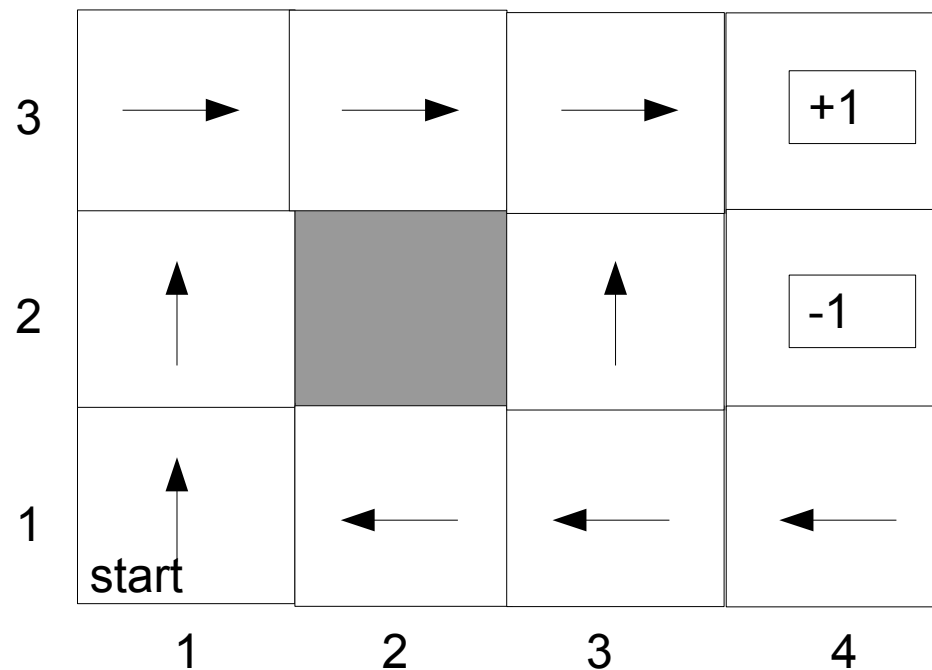
- Fully observable MDP environment with stationary laws
- The agent doesn't know how it works (no transition or reward model)
- The agent can act in the environment, but doesn't know what the actions will do (no transition model). Actions can have probabilistic outcomes.
  - stochastic simulation

# Agents

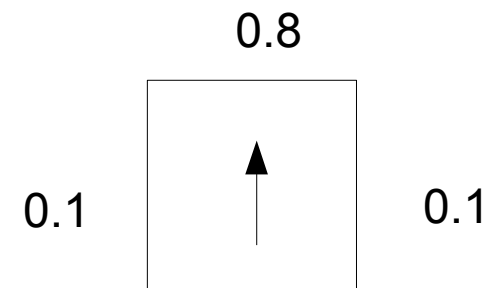
- Utility-based agent
  - learns a utility function on states and a transition model
- Q-learning agent
  - learns an action-value function or Q-function to compare values of available actions directly, without a transition model

# Passive reinforcement learning

- A simpler problem: the agent policy is fixed, but it needs to learn the utility function of this policy and the transition model of the environment



All non-terminal states have a reward of -0.04.



Transition model

# Trials

- The agent executes a set of trials in the environment
  - start in  $s_0$
  - execute fixed policy  $\pi$  until a terminal state is reached
- A trial looks like this:

(1,1)-.04, (1,2)-.04, (1,3)-.04, (1,2)-.04, (1,3)-.04, (2,3)-.04,  
(3,3)-.04, (4,3)+1

- Utility if policy  $\pi$  is followed (to be learned):

$$U^\pi(s) = \sum_{t=0}^{\infty} \gamma^t R(s_t), \quad \gamma = 1 \quad (\text{finite-horizon MDP})$$

# Direct utility estimation

- Each trial provides a *sample* of the utility for each state

trial: (1,1)-.04, (1,2)-.04, (1,3)-.04, (1,2)-.04, (1,3)-.04, (2,3)-.04,  
(3,3)-.04, (4,3)+1

sample utility for (3,3):  $-.04+1 = .96$

two samples for (1,3):  $-.04-.04-.04+1=.88$  and  $\dots=.8$

sample for (1,1):  $.72$

- Estimate utility by keeping a table of state utility values, where each value is a running average of samples seen so far
- In the limit of infinitely many trials, table values will converge to the true utility function
- This completely ignores the dependencies of states!

# Adaptive dynamic programming

- Learn transition model as you go along
  - e.g., by tallying frequencies of state transitions and using those as probability estimates
- Solve corresponding MDP with a dynamic programming method (value or policy iteration)

trial: (1,1)-.04, (1,2)-.04, (1,3)-.04, (1,2)-.04, (1,3)-.04, (2,3)-.04,  
(3,3)-.04, (4,3)+1

$\pi(1,1)=up$ , so  $T((1,1),up,(1,2))=1$

$\pi(1,3)=up$ , so  $T((1,3),up,(1,2))=.5$  and  $T((1,3),up,(2,3))=.5$

Estimates will converge to probabilities as num. samples increases

# Temporal-difference (TD) learning

- ADP still requires solving  $N$  Bellman equations (intractable for large  $N$ )
- Instead: use observed state transitions to adjust utility estimate of observed states such that they agree with the Bellman equations, but without solving the equations.

trial: (1,1)-.04, (1,2)-.04, (1,3)-.04, (1,2)-.04, (1,3)-.04, (2,3)-.04, (3,3)-.04, (4,3)+1

Suppose current estimates for  $U^\pi(1,3)=0.81$  and  $U^\pi(2,3)=0.9$

If the transition (1,3) to (2,3) occurred all the time, we would expect  $U^\pi(1,3) = -.04 + U^\pi(2,3) = 0.86$

So our current estimate is too low, and we can adjust it upward

# TD-learning: update rule

Adjustment (slight) in the direction of temporal difference

$$U^\pi(s) := U^\pi(s) + \alpha (R(s) + \gamma U^\pi(s') - U^\pi(s))$$

... whenever  $s$  transitions to  $s'$  in the trial

$\alpha$  is the learning rate (free parameter), which must decay in order for  $U$  to converge to equilibrium values

No model needed!

TD takes more trials and has more variability than ADP, but is much more simple computationally

# Active reinforcement learning

- Agent must decide on a policy as well as learning utilities
- Can learn transition probabilities for all actions same way as in ADP
- Since Bellman equations defined for optimal actions, pick action that maximizes current estimated utility at each step
- Will that work?

# Exploration vs. exploitation

- Greedy agent will only very occasionally learn the optimal policy
- Why? Because learned transition estimates are not the same as MDP
- Actions do more than provide rewards
- There is utility in improving the model by trying currently suboptimal actions
- $n$ -armed bandits
- Optimal exploration is hard to solve exactly, but reasonable exploration strategies are GLIE: greedy in the limit of infinite exploration
- Stochastic policies: e.g.,  $\epsilon$ -greedy

→	→	→	+1
↓		↑	-1
→	→	↑	↓

From AIMA: policy learned by a greedy ADP agent

# Alternative to $\epsilon$ -greedy: optimistic utility prior

- Idea: give more weight to actions you haven't tried often, but avoid those believed to have low utility
- Agent behavior: initially, assume the world is full of wonderful rewards all over the place (optimistic utility  $U+(s)$  for state  $s$  and  $N(a,s)$  = number of times action  $a$  was executed in state  $s$ )
- If using ADP with value iteration, the update equation becomes:

$$U+(s) := R(s) + \gamma \max_a f(\sum_{s'} T(s,a,s') U+(s'), N(a,s)),$$

... where  $f(u,n)$  is the exploration function, e.g.

$$f(u,n) = R+ \text{ if } n < N_e \text{ and } f(u,n) = u \text{ otherwise}$$

... where  $R+$  is an optimistic estimate of the best possible reward in any state

# Q-learning

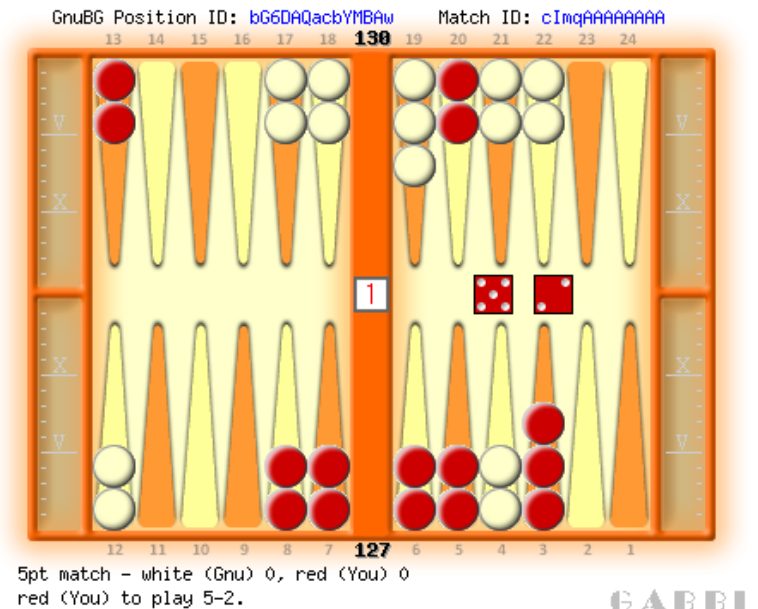
- Active TD learning: if using state utility update rule, need a transition model...
  - Instead, use action-value functions  $Q(a,s): U(s) = \max_a Q(a,s)$
  - A TD agent using Q-values does not need a transition model (model-free RL method)
  - Update rule (similar to TD-learning for utilities):  
$$Q(a,s) := Q(a,s) + \alpha(R(s) + \gamma \max_{a'} Q(a',s') - Q(a,s))$$
- ... where  $\alpha$  is the (decaying) learning rate

# Which is better?

- Model-based learning (ADP) or model-free learning (Q-learning)?
- Model-based: computationally expensive but fewer trials required for learning
- Model-free: simple, but takes more trials and is more variable
- A fundamental AI question: sometimes the world is its own best model, sometimes knowledge-building is required

# Famous RL agents: TD-gammon

- Gerry Tesauro's backgammon player (1992) used TD learning with some extensions to learn from self-play
- Reward given at the end of the game
- Originally states = raw board positions (no backgammon knowledge) : competitive among computer programs after 200,000 games against itself
- With added backgammon-related features: competitive with top world human players after 300,000 games against itself



# Limitations: partial observability

- It can be shown that all these methods can fail arbitrarily when the MDP is only partially observable
- Partial observability: evidence variables (like in HMMs) can be observed but the state is not known
- Solutions for Partially Observable MDPs (POMDPs) may be based on
  - direct search in policy space (cf. Simulated annealing, GAs, MCMC)
  - belief state filtering methods

# Summary

- Reinforcement learning is looking for an optimal policy when the agent doesn't have a model of the environment
- Environments are assumed discrete, stochastic, fully observable, stationary and having the Markov property
- Agent can do RL by learning the utility functions
  - Direct estimation
  - ADP
  - TD-learning
- Agent can do RL by learning action-value functions directly
- If decisions must be made (policy not fixed) there is an exploration-exploitation tradeoff

# Project proposal due next Wed in class

- Final project composition: proposal (5%), presentation (5%), final report (20%)
- Proposal (1 page):
  - What you will do (problem statement)
  - Why you should do it (motivation)
  - How you will go about it (approach)
  - Simple timeline (you have 5.5 weeks including next week)
- Presentation matters and will be taken into account
  - Use word-processing or LaTeX
  - Use professional-quality language and structure; think business plan or grant proposal
- The sooner you hand it in, the sooner I will give you feedback