Reinforcement Learning

CMPSCI 383
Nov 29, 2011

Today's lecture

- Review of Chapter 17: Making Complex Decisions
  - Sequential decision problems
- The motivation and advantages of reinforcement learning.
- Passive learning
  - Policy evaluation
    - Direct utility estimation
    - Adaptive dynamic programming
    - Temporal Difference (TD) learning

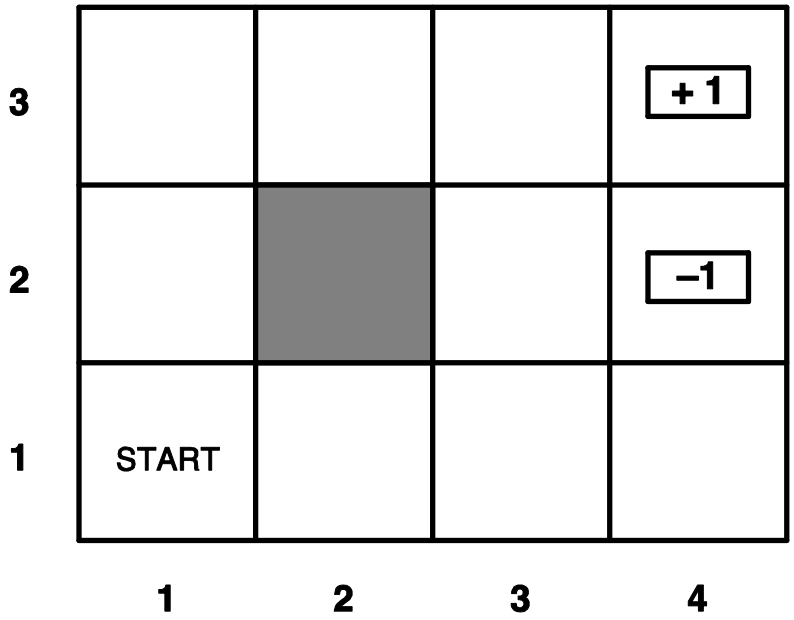
Check this out

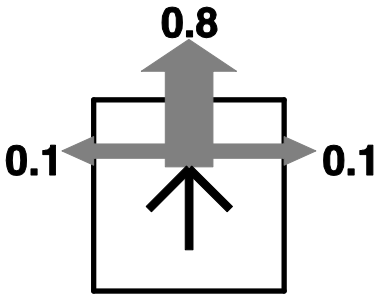
MIT's Technology Review has an in-depth interview with Peter Norvig, Google'sDirector of Research, and Eric Horvitz, a Distinguished Scientist at MicrosoftResearch about their optimism for the futureof AI.

http://www.technologyreview.com/computing/39156/

A Simple Example

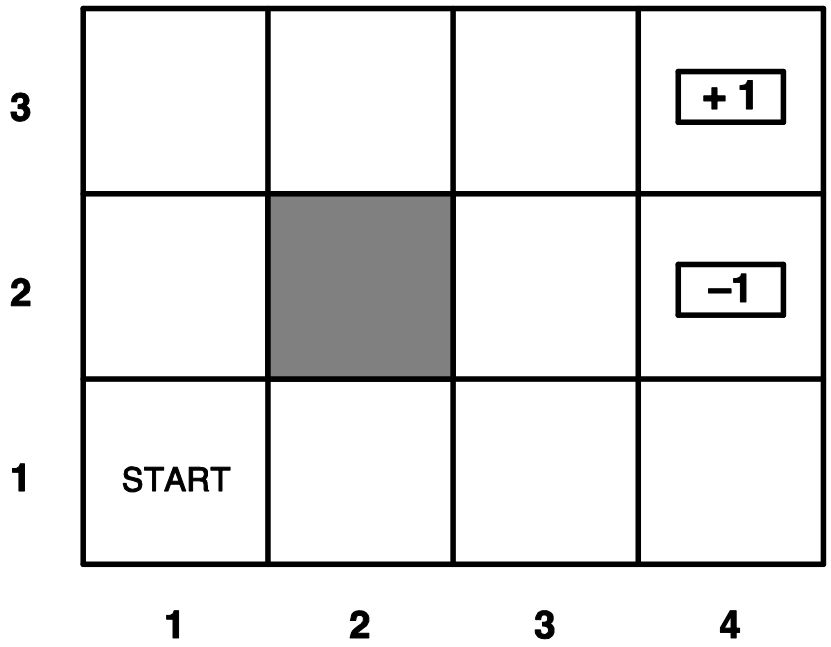
- “Gridworld” with 2 Goal states
- Actions: Up, Down, Left, Right
- Fully observable: Agent knows where it is

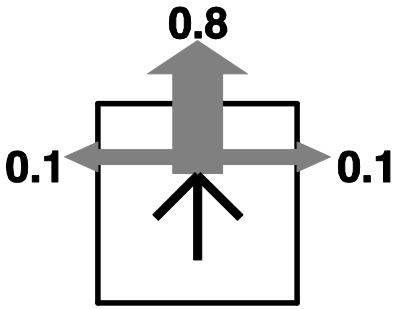




Transition Model

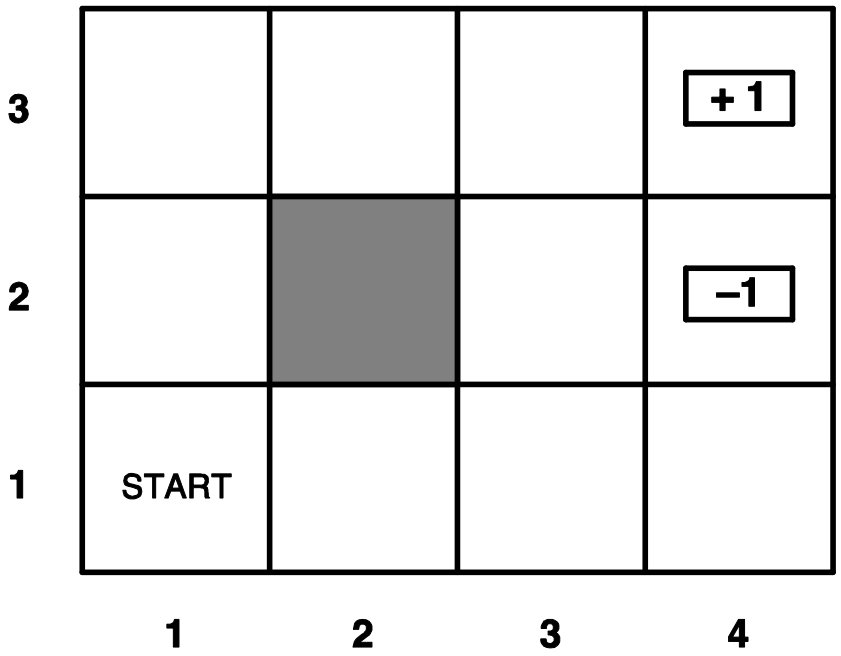
$$P(s' \mid s, a)$$

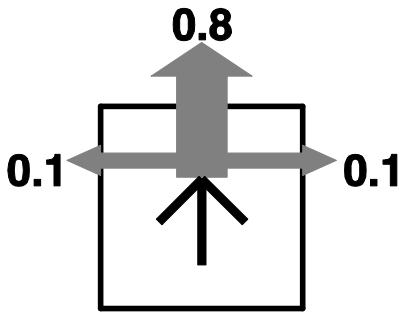




Markov Assumption

$$P(s' \mid s, a)$$





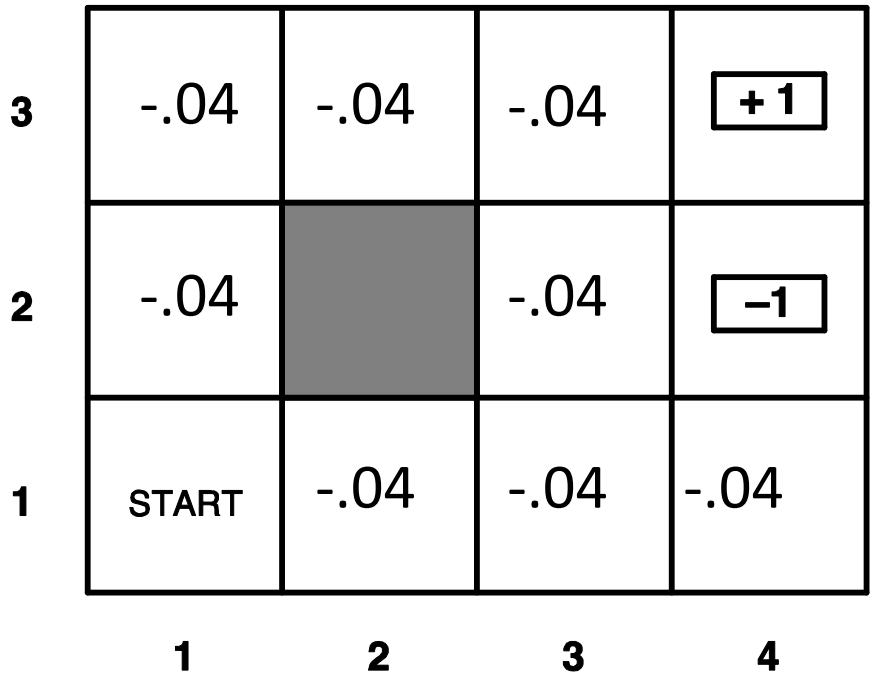
Agent's Utility Function

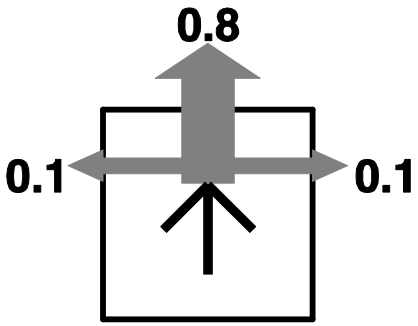
- Performance depends on the entire sequence of states and actions.
  - "Environment history"
- In each state, the agent receives a reward $R(s)$.
  - The reward is real-valued. It may be positive or negative.
- Utility of environment history = sum of reward received.

Reward function

$$P(s' \mid s, a)$$

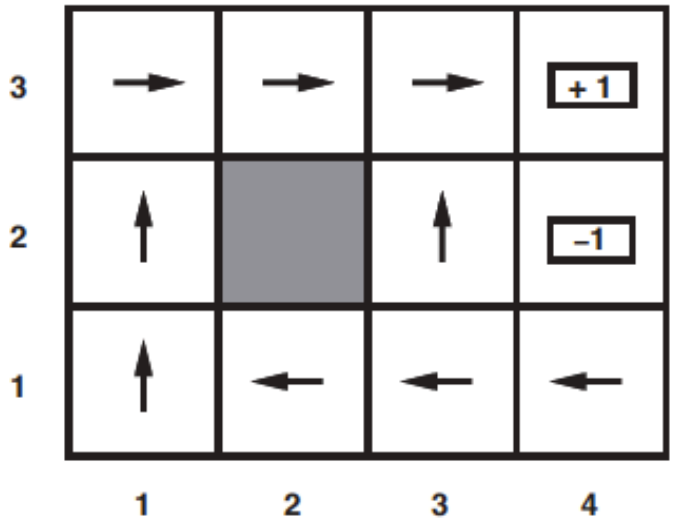
$$R(s)$$

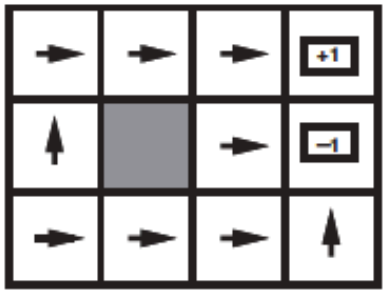




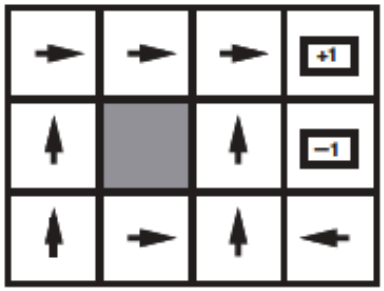
Decision Rules

- Decision rules say what to do in each state.
- Often called policies, π.
- Action for state s is given by $\pi(s)$.

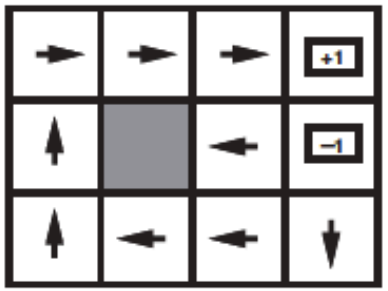




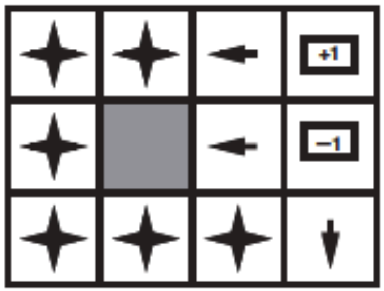
$R(s) < -1.6284$



$-0.4278 < R(s) < -0.0850$



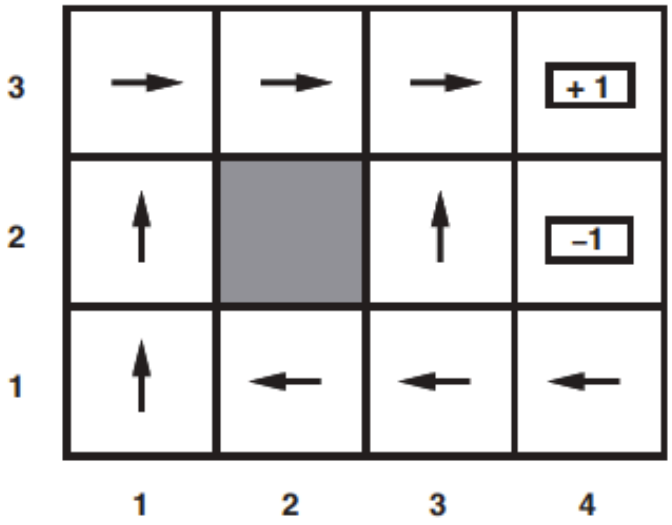
$-0.0221 < R(s) < 0$

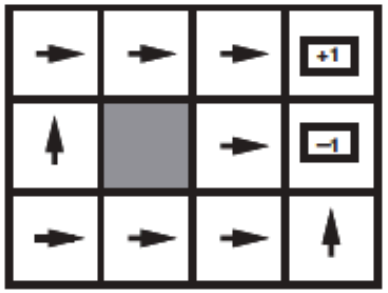


$R(s) > 0$

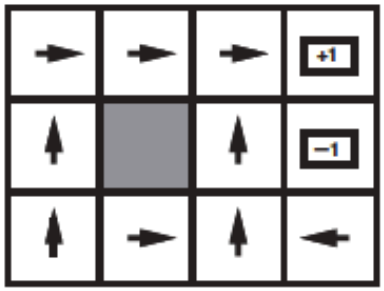
Our Goal

- Find the policy that maximizes the expected sum of rewards.
- Called an *optimal policy*.

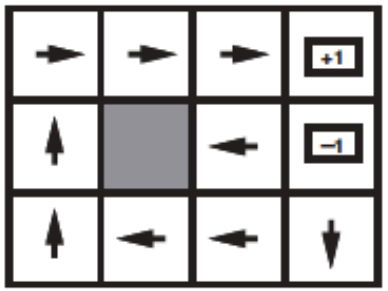




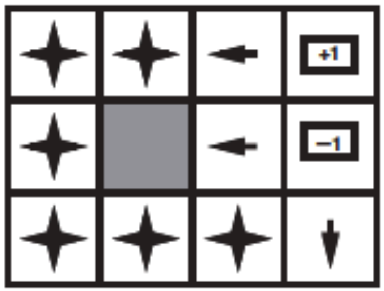
$R(s) < -1.6284$



$-0.4278 < R(s) < -0.0850$



$-0.0221 < R(s) < 0$



$R(s) > 0$

Markov Decision Process (MDP)

- M=(S,A,P,R)
- S = set of possible states
- A = set of possible actions
- P(s'|s,a) gives transition probabilities
- R = reward function

- Goal: find an optimal policy, π^*.

Finite/Infinite Horizon

- Finite horizon: the “game” ends after N steps.
- Infinite horizon: the “game” never ends
- “With a finite horizon, the optimal action in a given state could change over time.”
  - The optimal policy is *nonstationary*.
- With infinite horizon, the optimal policy is *stationary*.

Utilities over time

- Additive rewards:

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots$$

- Discounted rewards:

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$$

- Discount factor:

$$\gamma \in [0,1]$$

Discounted Rewards

- Would you rather have a marshmallow now, or two in 20 minutes?
- Infinite sums!

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots$$

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$$

Utility of States

- Given a policy, we can define the utility of a state:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} R(s_t)\right]$$

Policy Evaluation

- Finding the utility of states for a given policy.
- Solve a system of linear equations:

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

- An instance of a “Bellman Equation”.

Policy Evaluation

- The *n* linear equations can be solved in $O(n^3)$ with standard linear algebra methods.
- If $O(n^3)$ is still too much, we can do it iteratively:

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

Optimal Policy

$$\pi_s^* = \arg\max_{\pi} U^{\pi}(s)$$

- Optimal policy doesn't depend on what state you start in (for infinite horizon discounted case).
- Optimal policy: π^*
- True utility of a state:

$$U^{\pi^*}(s) = U(s)$$

"optimal value function"

Selecting optimal actions

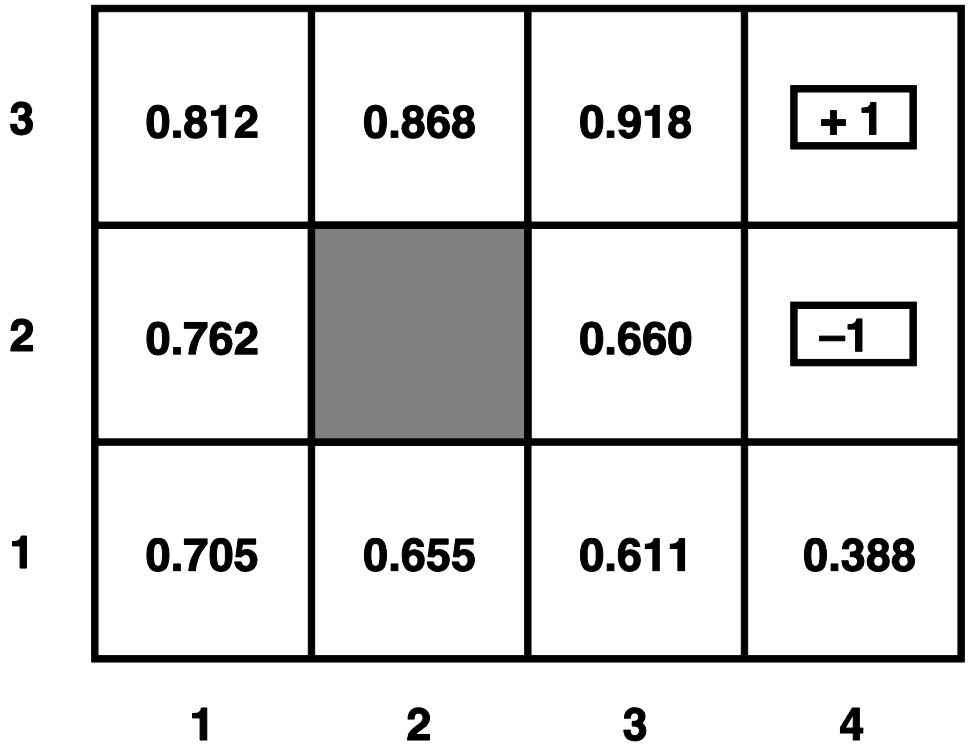
- Given the true $U(s)$ values, how can we select actions? (Maximum expected utility – MEU)

$$a_t = \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s_t, a) U(s')$$

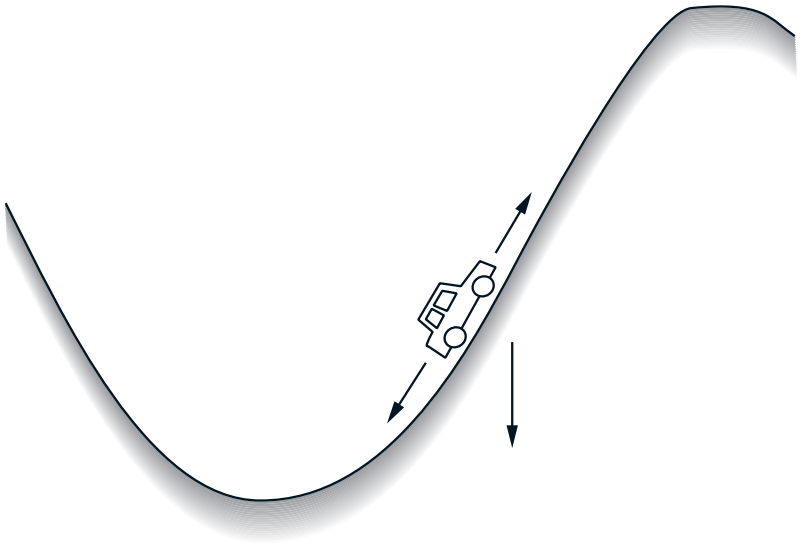
Utility and Rewards

- Utility = long term total reward from s onwards
- Reward = short term reward from s

Utility

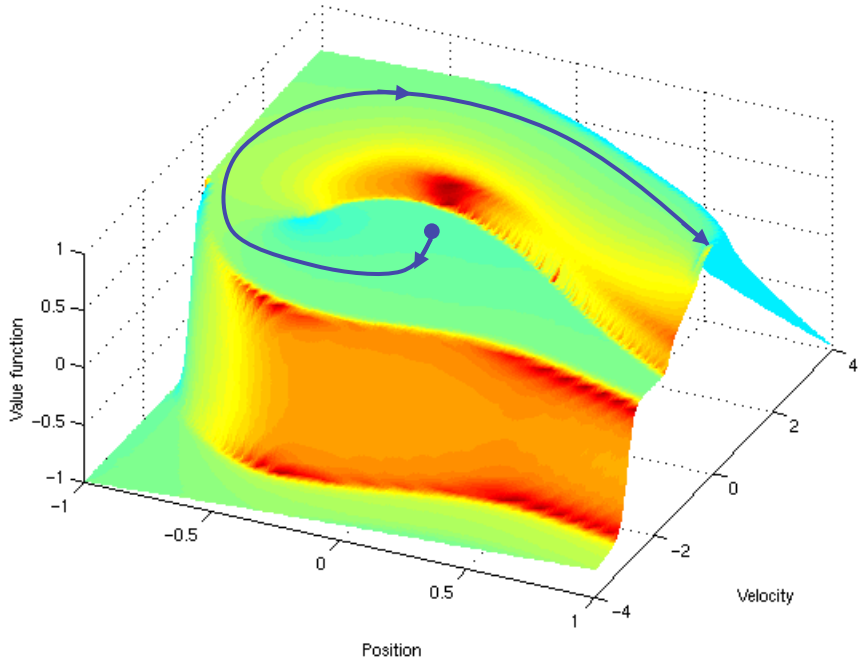


Value-Guided Optimal Control



Get to the top of the hill as quickly as possible (roughly)

Predicted minimum time to goal (negated)



Munos & Moore "Variable resolution discretization for high-accuracy solutions of optimal control problems", IJCAI 99.

Searching for Optimal Policies

- Bellman Equation

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

- If we write out the Bellman equation for all n states, we get n equations, with n unknowns: $U(s)$.
- We can solve this system of equations to determine the Utility of every state.

Value Iteration

- The equations are non-linear, so we can't use standard linear algebra methods.

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

- Value iteration: start with random initial values for each U(s), iteratively update each value to fit the fight-hand side of the equation:

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

Value Iteration

- The update is applied simultaneously to every state.
- If this update is applied infinitely often, we are guaranteed to find the true $U(s)$ values.
  - There is one unique solution
- Given the true $U(s)$ values, how can we select actions? (Maximum expected utility – MEU)

$$a_t = \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s_t, a) U_i(s')$$

Policy Iteration

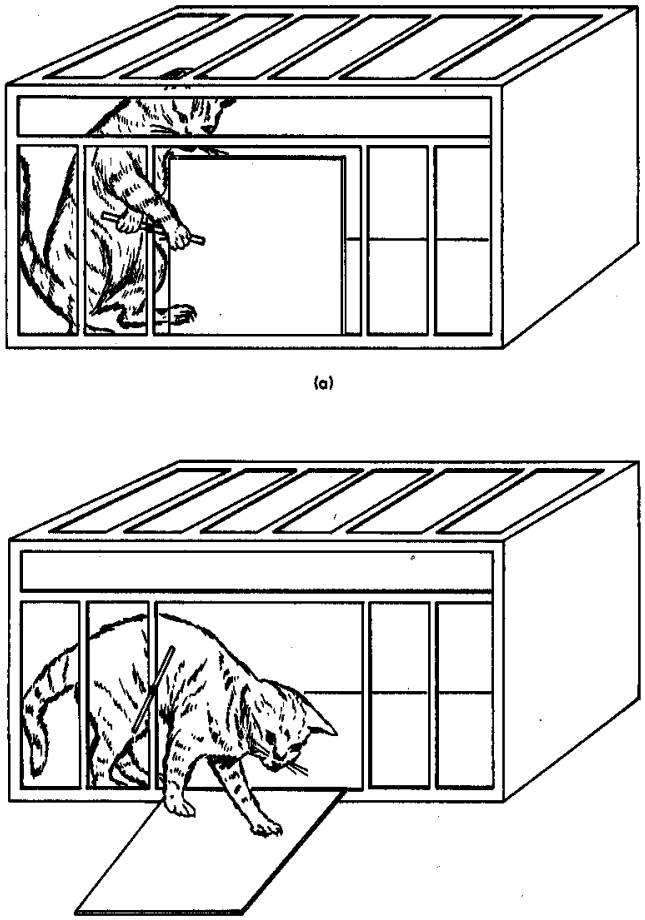
- Policy iteration interleaves two steps:
  - Policy evaluation: Given a policy, compute the utility of each state for that policy
  - Policy improvement: Calculate a new MEU policy
- Terminate when the policy doesn't change the utilities.
- Guaranteed to converge to an optimal policy

Asynchronous Policy Iteration

- We said the utility of every state is updated simultaneously. This isn't necessary.
- You can pick and subset of the states and apply either policy improvement or value iteration to that subset.
- Given certain conditions, this is also guaranteed to converge to an optimal policy.

Reinforcement Learning

Edward L. Thorndike (1874-1949)



Learning by “Trial-and-Error”

puzzle box

"Law of Effect"

"Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that when it recurs, they will be less likely to occur."

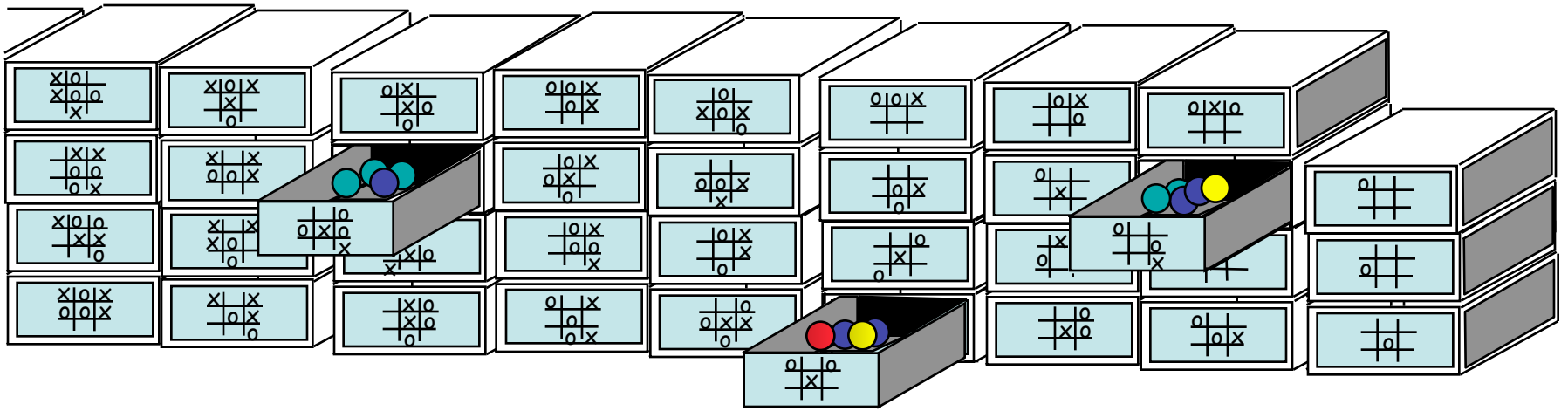
Edward Thorndike, 1911

RL = Search + Memory

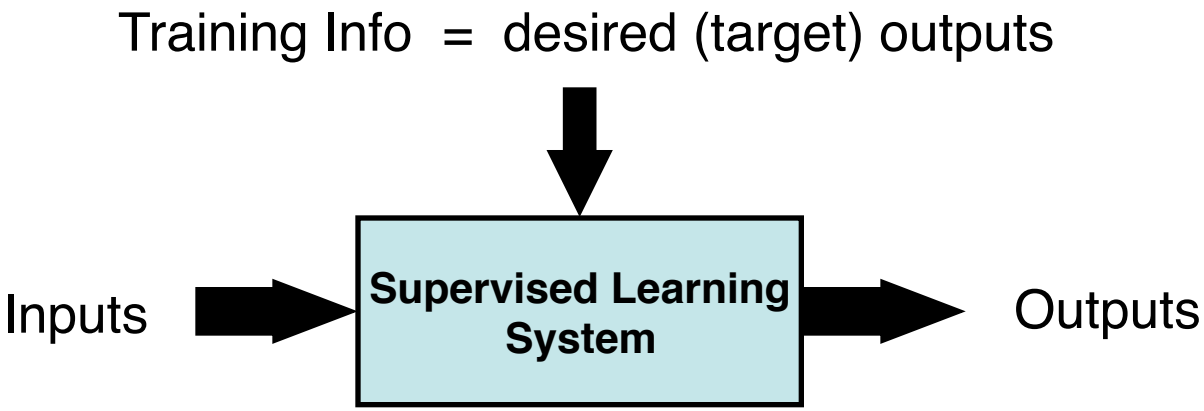
- **Search**: Trial-and-Error, Generate-and-Test, Variation-and-Selection
- **Memory**: remember what worked best for each situation and start from there next time

MENACE (Michie, 1961)

"Matchbox Educable Noughts and Crosses Engine"

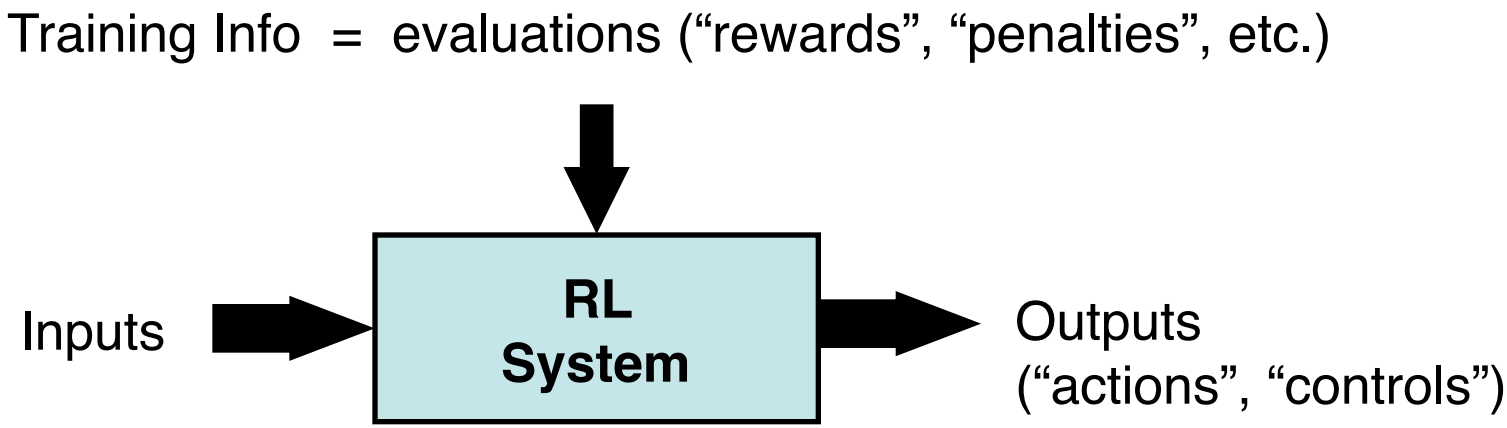


Supervised learning



Error = (target output – actual output)

Reinforcement learning



Objective: get as much reward as possible

Reinforcement learning

- Learning what to do—how to map situations to actions—so as to maximize a numerical reward signal
- Rewards may be provided following each action, or only when the agent reaches a goal state.
- The “credit assignment problem”.

Why reinforcement learning?

- More realistic model of the kind of learning that animals do.
- Supervised learning is sometimes unrealistic: where will correct examples come from?
- Environments change, and so the agent must adjust its action choices.

Stick to this kind of RL problem here:

- Environment is fully observable: agent knows what state the environment is in
- But: agent does not know how the environment works or what its actions do

Some kinds of RL agents

- Utility-based agent: learns utility function on states and uses it to select actions
  - Needs an environment model to decide on actions
- Q-learning agent: leans and action-utility functions, or Q-function, giving expected utility for taking each action in each state.
  - Does not need an environment model.
- Reflex agent: learn a policy without first learning a state-utility function or a Q-function

Passive versus active learning

- A **passive learner** simply watches the world going by and tries to learn the utility of being in various states.
- An **active learner** must also act using the learned information, and can use its problem generator to suggest explorations of unknown portions of the environment.

Passive learning

Given (but agent doesn't know this):

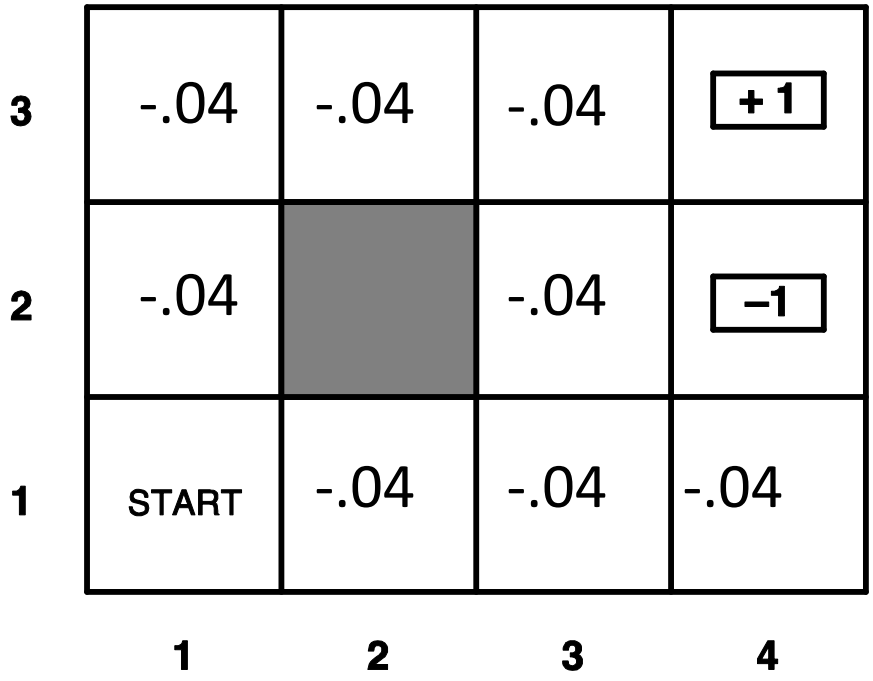
- A Markov model of the environment.
- States, with probabilistic actions.
- Terminal states have rewards/utilities.

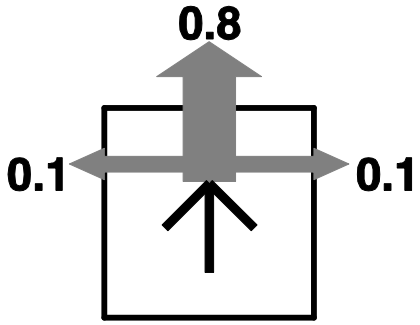
Problem:

- Learn expected utility of each state.

Note: if agent knows how the environment and its actions work, can solve the relevant Bellman equation (which would be linear).

Example





Percepts tell agent:
[State, Reward, Terminal?]

Learning utility functions

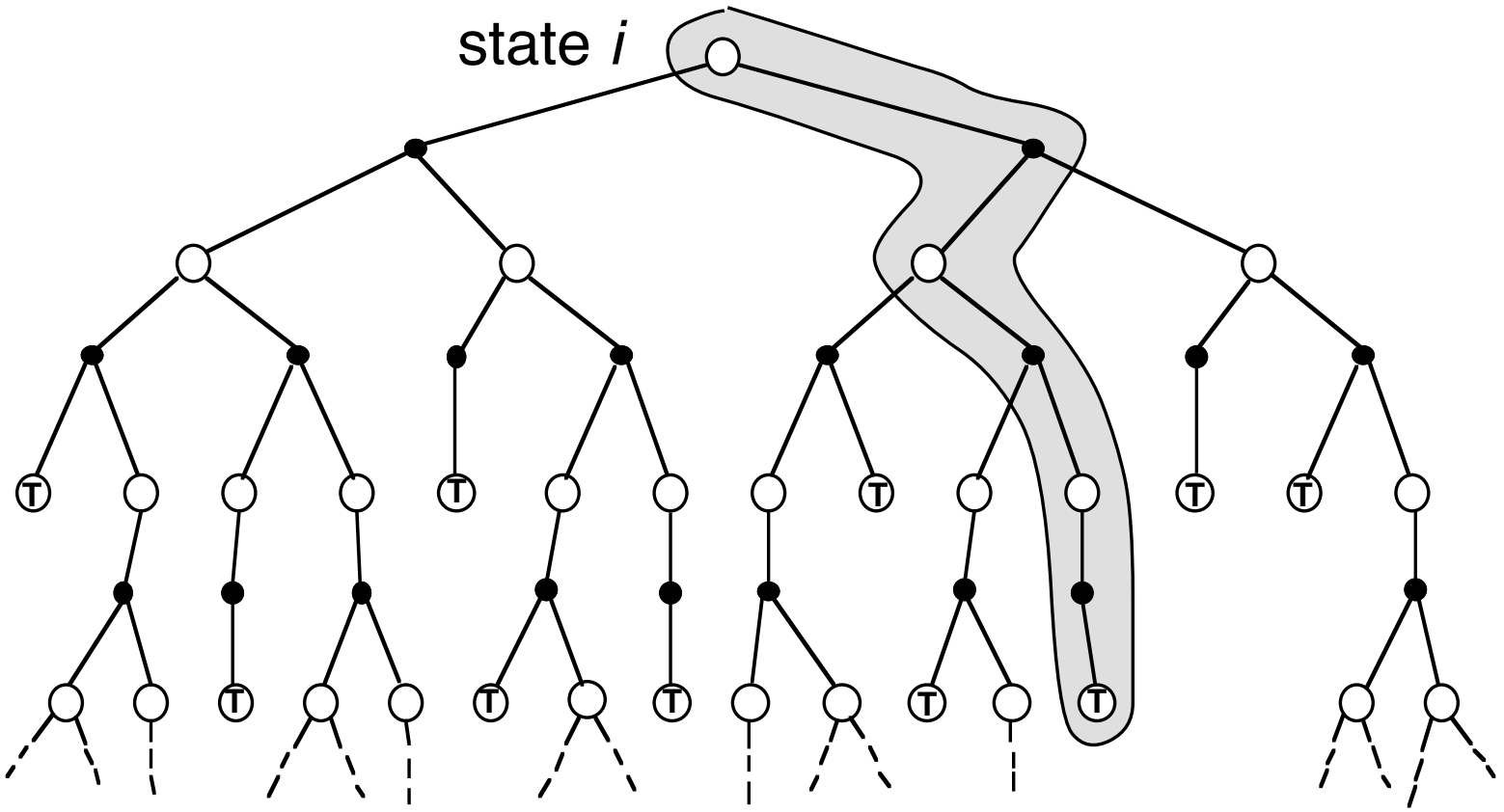
- A **training sequence** (or episode) is an instance of world transitions from an initial state to a terminal state.
- The **additive utility assumption**: utility of a sequence is the sum of the rewards over the states of the sequence.
- Under this assumption, the utility of a state is the expected **reward-to-go** of that state.

Direct Utility Estimation

- For each training sequence, compute the reward-to-go for each state in the sequence and update the utilities.
- This is just learning the utility function from examples.
- Generates utility estimates that minimize the mean square error (LMS-update).

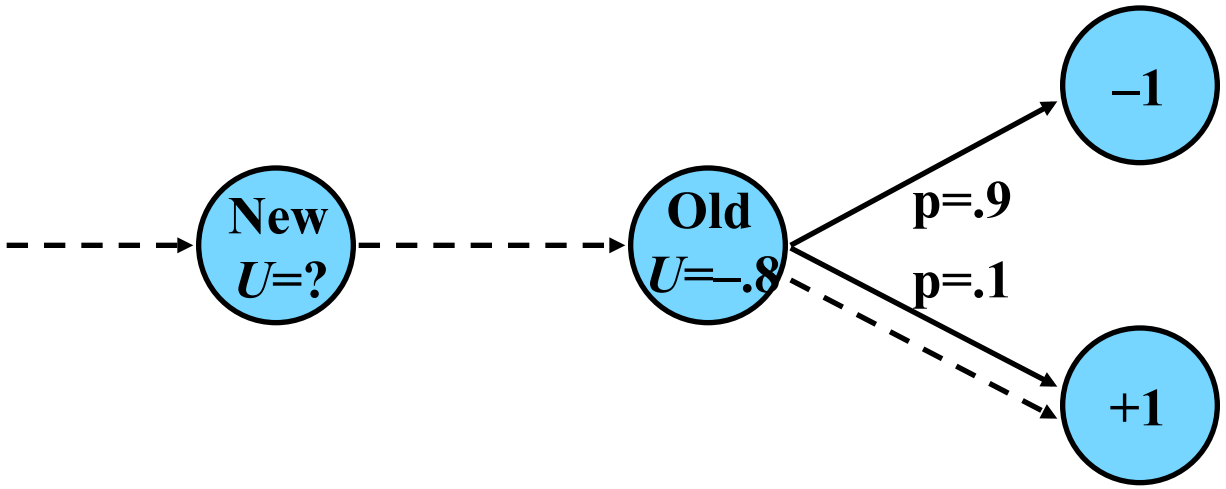
Direct Utility Estimation

$$U(i) \leftarrow (1-\alpha)U(i) + \alpha\, REWARD(\text{training sequence})$$



Problems with direct utility estimation

Converges slowly because it ignores the relationship between neighboring states:



Key Observation: Temporal Consistency

- Utilities of states are not independent
- The utility of each state equals its own reward plus the expected utility of its successor states:

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

The key fact:

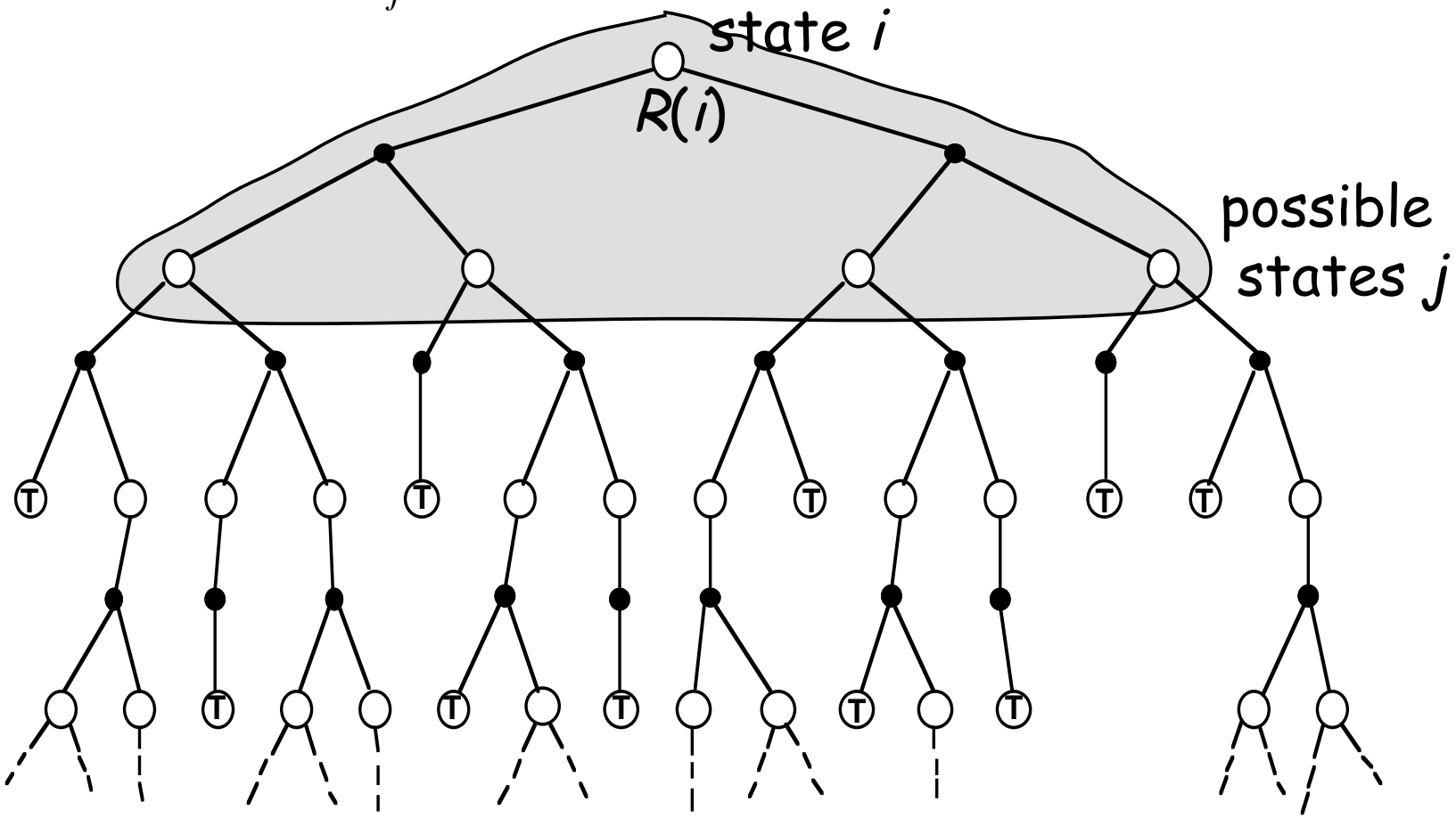
$$\begin{aligned} U_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots \\ &= r_{t+1} + \gamma\left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \cdots \right) \\ &= r_{t+1} + \gamma U_{t+1} \end{aligned}$$

Adaptive dynamic programming

- Learn a model: transition probabilities, reward function
- Do policy evaluation
  - Solve the Bellman equation either directly or iteratively (value iteration without the max)
- Learn model while doing iterative policy evaluation:
  - Update the model of the environment after each step. Since the model changes only slightly after each step, policy evaluation will converge quickly.

Adaptive dynamic prog (ADP)

$$U(i) \leftarrow R(i) + \sum_j P_{ij} U(j)$$



Passive ADP learning curves

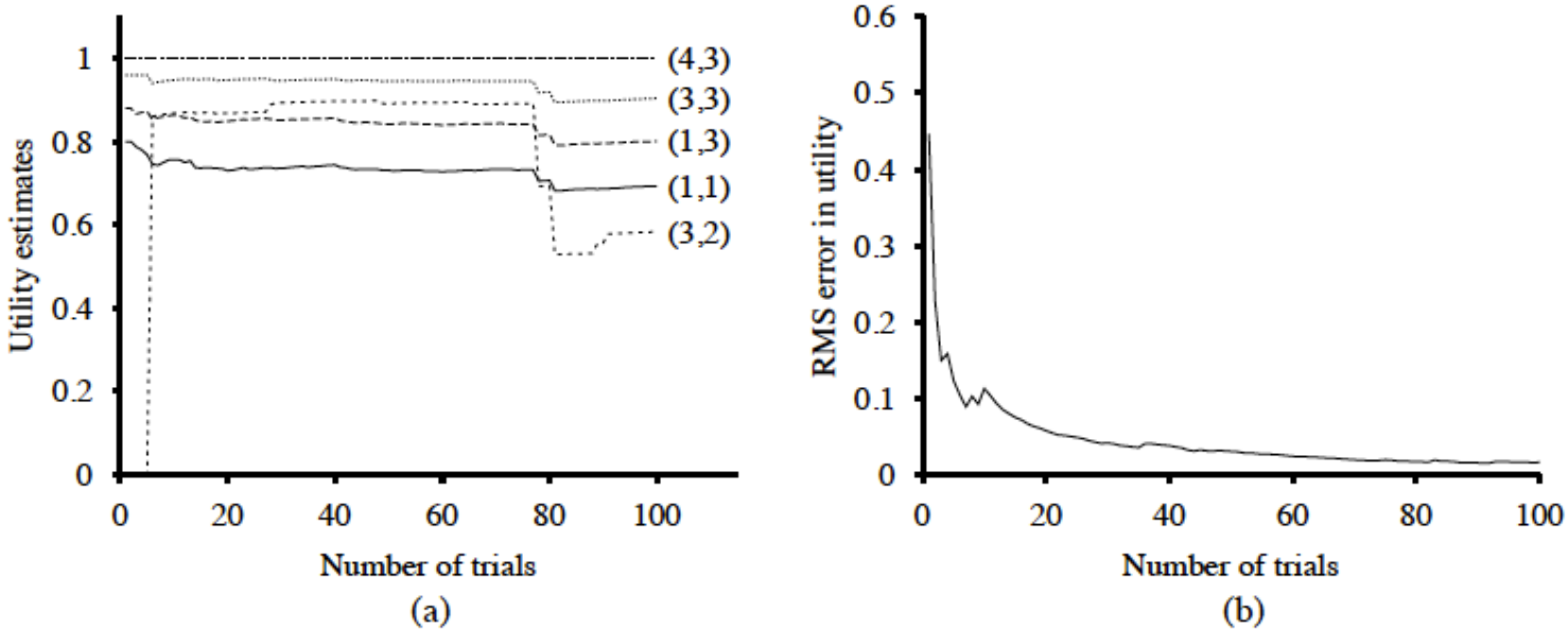
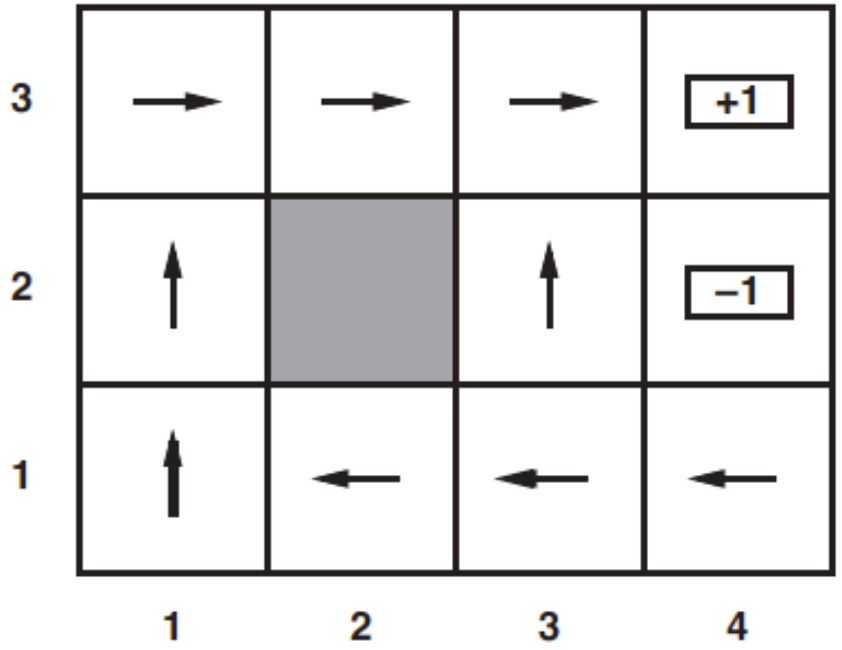


Figure 21.3 **FILES: .** The passive ADP learning curves for the 4×3 world, given the optimal policy shown in Figure 21.1. (a) The utility estimates for a selected subset of states, as a function of the number of trials. Notice the large changes occurring around the 78th trial—this is the first time that the agent falls into the -1 terminal state at (4,2). (b) The root-mean-square error (see Appendix A) in the estimate for $U(1,1)$, averaged over 20 runs of 100 trials each.

Exact utility values for the example



3	0.812	0.868	0.918	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Temporal difference (TD) learning

Approximate the constraint equations without solving them for all states.

Modify $U(i)$ whenever we see a transition from i to j using the following rule:

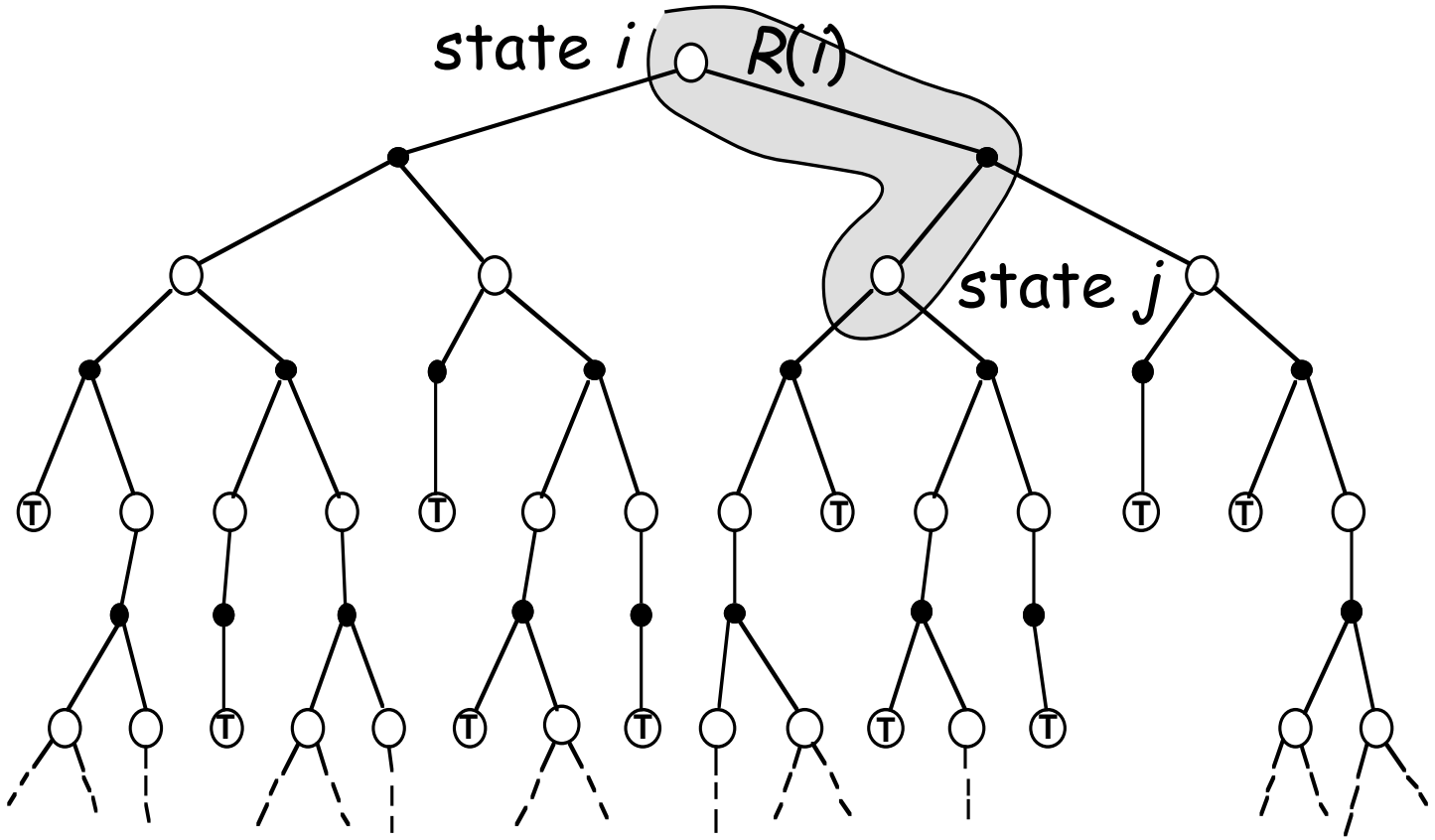
$$U(i) \leftarrow U(i) + \alpha\left[R(i) + U(j) - U(i)\right]$$

The modification moves $U(i)$ closer to satisfying the original equation.

Q. Why does it work?

TD learning contd.

$$U(i) \leftarrow (1-\alpha)U(i) + \alpha\left[R(i) + U(j)\right]$$



TD learning curves

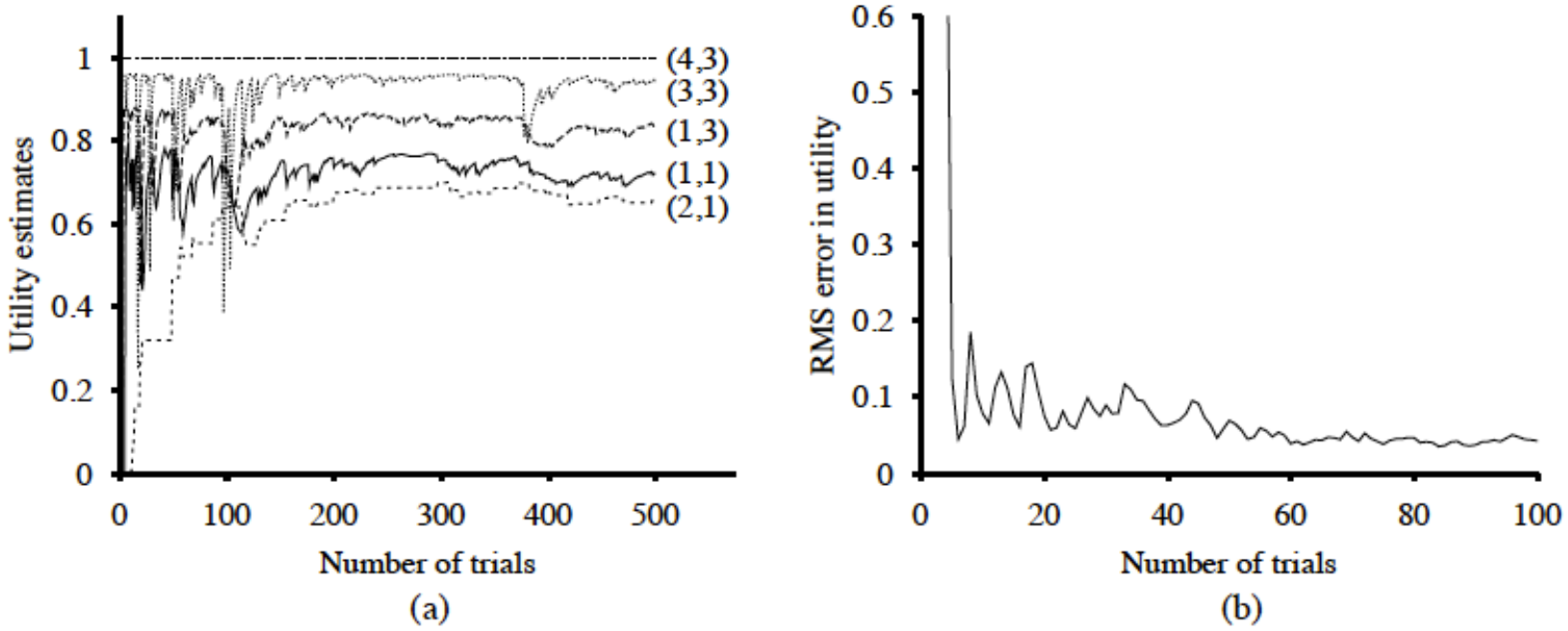


Figure 21.5 **FILES: .** The TD learning curves for the 4×3 world. (a) The utility estimates for a selected subset of states, as a function of the number of trials. (b) The root-mean-square error in the estimate for $U(1,1)$, averaged over 20 runs of 500 trials each. Only the first 100 trials are shown to enable comparison with Figure 21.3.

Next Class

- Active Reinforcement Learning
- Sec. 21.3

Next Class

- Reinforcement Learning
- Secs. 21.1 – 21.3