CS 461: Machine Learning
Lecture 8

Dr. Kiri Wagstaff
wkiri@wkiri.com

Plan for Today

- Review Clustering

- Homework 4 Solution

- Reinforcement Learning
  - How different from supervised, unsupervised?
- Key components
- How to learn
  - Deterministic
  - Nondeterministic

Review from Lecture 7

- Unsupervised Learning
  - Why? How?
- K-means Clustering
  - Iterative
  - Sensitive to initialization
  - Non-parametric
  - Local optimum
  - Rand Index
- EM Clustering
  - Iterative
  - Sensitive to initialization
  - Parametric
  - Local optimum

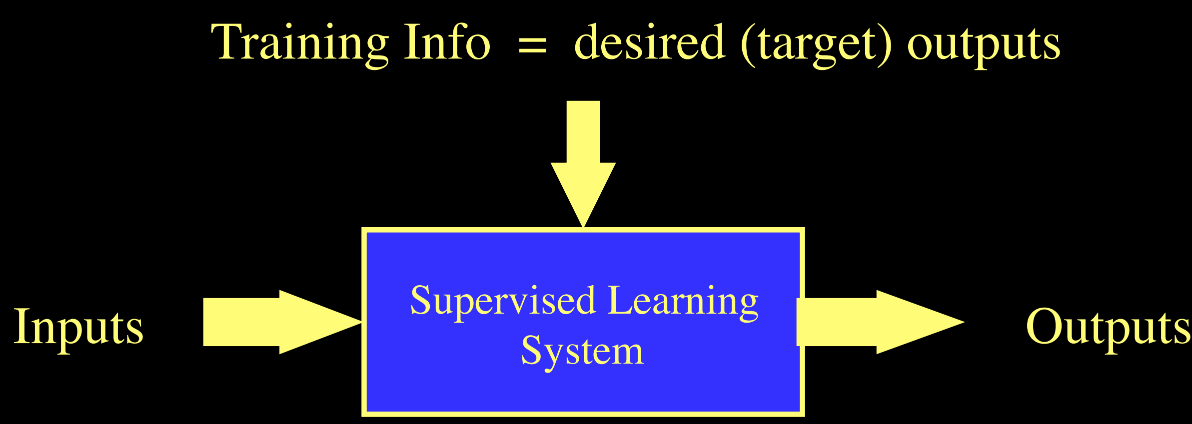
Reinforcement Learning

Chapter 16

What is Reinforcement Learning?

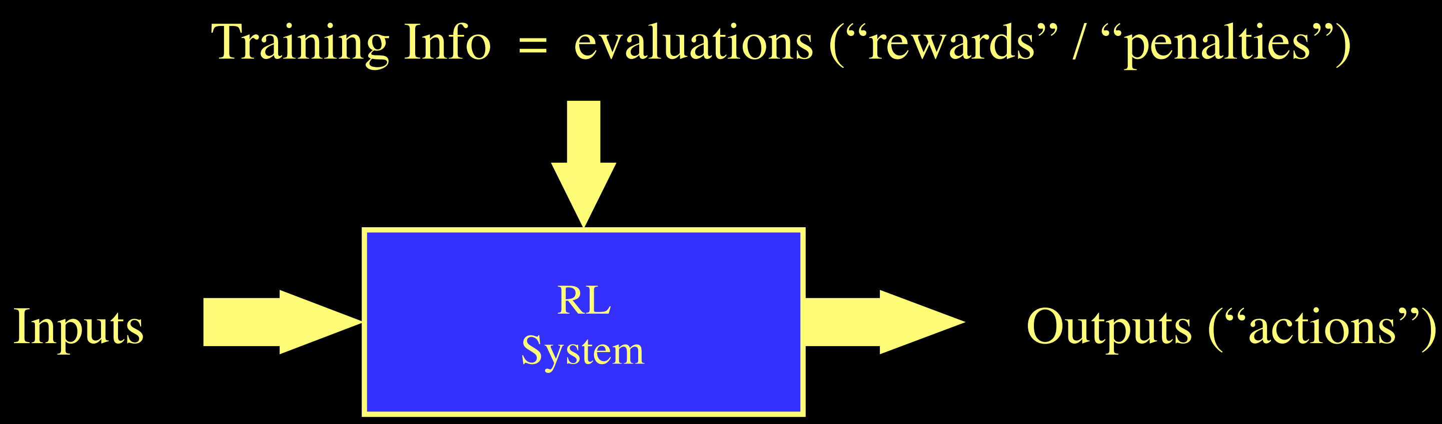
- Learning from interaction
- Goal-oriented learning
- Learning about, from, and while interacting with an external environment
- Learning what to do—how to map situations to actions—so as to maximize a numerical reward signal

Supervised Learning



$$\text{Error} = (\text{target output} - \text{actual output})$$

Reinforcement Learning



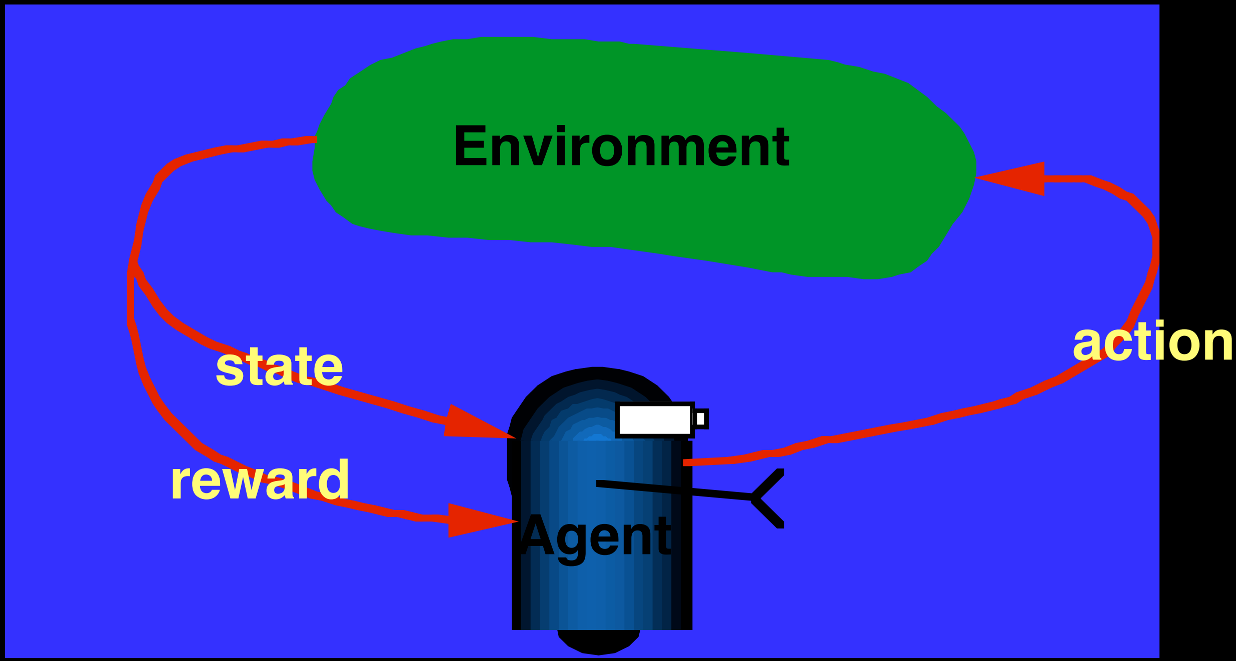
Objective: get as much reward as possible

Key Features of RL

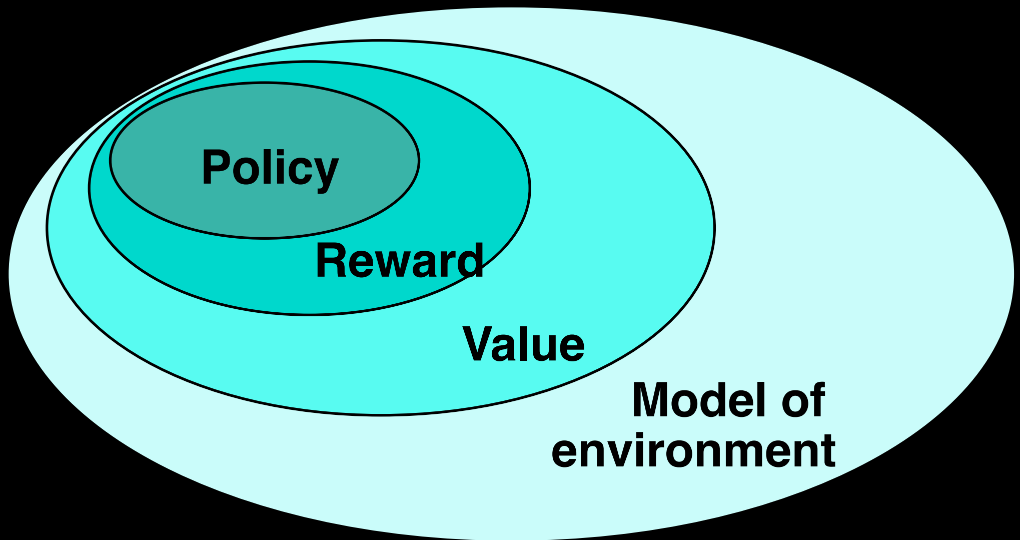
- Learner is not told which actions to take
- Trial-and-Error search
- Possibility of delayed reward
  - Sacrifice short-term gains for greater long-term gains
- The need to *explore* and *exploit*
- Considers the whole problem of a goal-directed agent interacting with an uncertain environment

Complete Agent (Learner)

- Temporally situated
- Continual learning and planning
- Object is to *affect* the environment
- Environment is stochastic and uncertain

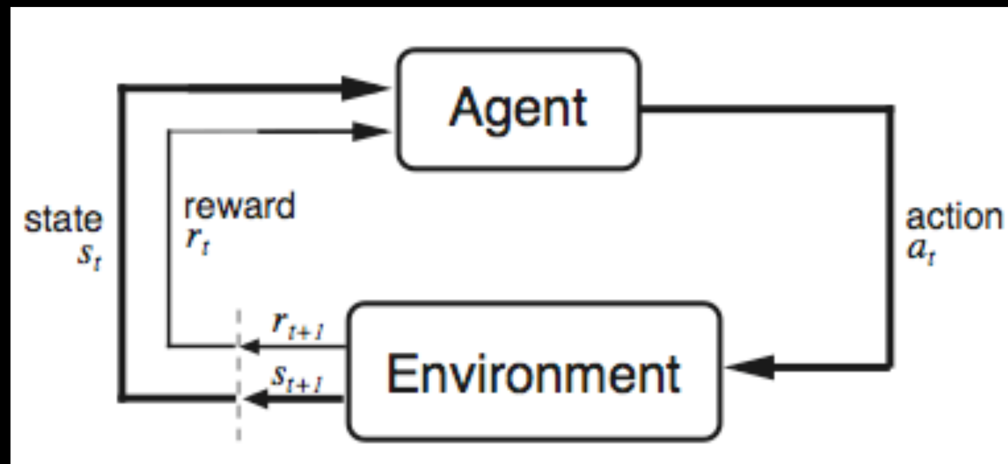


Elements of an RL problem



- Policy: what to do
- Reward: what is good
- Value: what is good because it *predicts* reward
- Model: what follows what

The Agent-Environment Interface



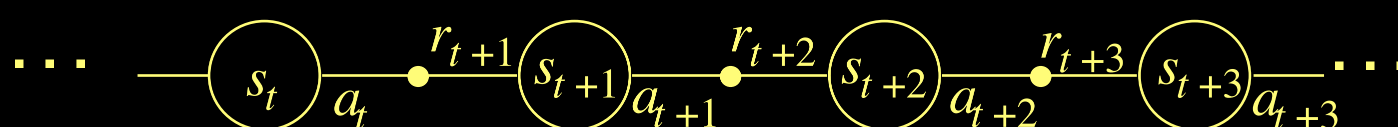
Agent and environment interact at discrete time steps: $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

produces action at step t: $a_t \in A(s_t)$

gets resulting reward: $r_{t+1} \in \Re$

and resulting next state: s_{t+1}



 [R. S. Sutton and A. G. Barto]

Elements of an RL problem

- s_t : State of agent at time t
- a_t: Action taken at time t
- In s_t, action a_t is taken, clock ticks and reward r_{t+1} is received and state changes to s_{t+1}
- Next state prob: $P(s_{t+1} \mid s_t, a_t)$
- Reward prob: $p(r_{t+1} \mid s_t, a_t)$
- Initial state(s), goal state(s)
- Episode (trial) of actions from initial state to goal

The Agent Learns a Policy

Policy at step t, π_t :

a mapping from states to action probabilities

$\pi_t(s,a) =$ probability that $a_t = a$ when $s_t = s$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience.
- Roughly, the agent's aim is to get as much reward as it can over the long run.

Goals and Rewards

- Goal state specifies what we want to achieve, not how we want to achieve it
  - "How" = policy
- Reward: scalar signal
  - Surprisingly flexible
- The agent must be able to measure success:
  - Explicitly
  - Frequently during its lifespan

Returns

Suppose the sequence of rewards after step t is :

$$r_{t+1}, r_{t+2}, r_{t+3}, \ldots$$

What do we want to maximize?

In general,

we want to maximize the **expected return**, $E\{R_t\}$, for each step t.

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze.

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T,$$

where T is a final time step at which a terminal state is reached, ending an episode.

Returns for Continuing Tasks

Continuing tasks: interaction does not have natural episodes.

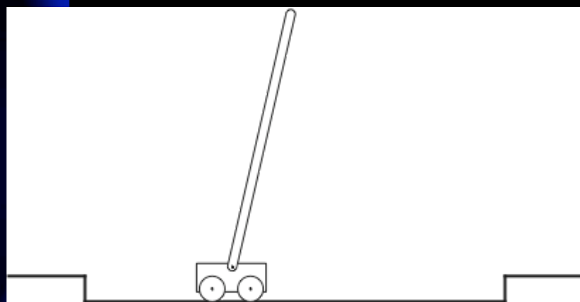
Discounted return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where $\gamma, 0 \leq \gamma \leq 1$, is the **discount rate**.

shortsighted $0 \leftarrow \gamma \rightarrow 1$ farsighted

An Example



Avoid failure: the pole falling beyond a critical angle or the cart hitting end of track.

As an episodic task where episode ends upon failure:

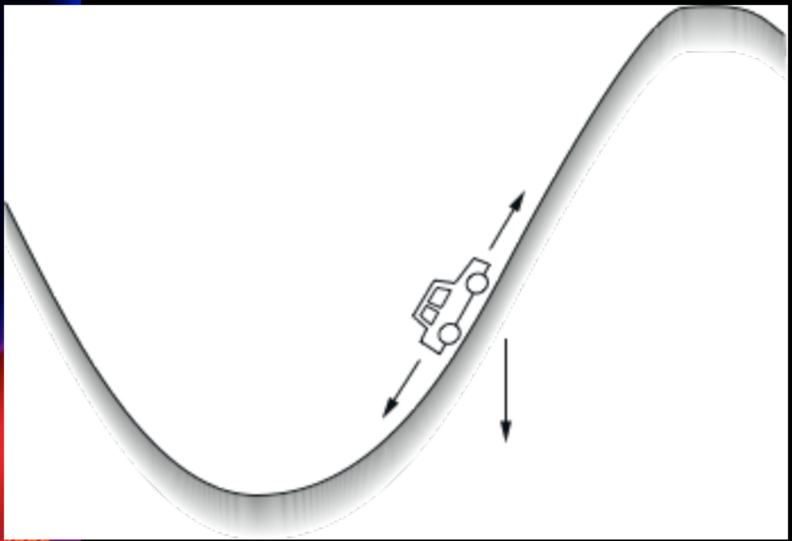
$$\text{reward} = +1 \text{ for each step before failure}$$
$$\Rightarrow \quad \text{return} = \text{number of steps before failure}$$

As a continuing task with discounted return:

$$\text{reward} = -1 \text{ upon failure; } 0 \text{ otherwise}$$
$$\Rightarrow \quad \text{return} = -\gamma^{k}, \text{ for } k \text{ steps before failure}$$

In either case, return is maximized by avoiding failure for as long as possible.

Another Example



Get to the top of the hill as quickly as possible.

reward = -1 for each step where **not** at top of hill

$\Rightarrow$ return = $-$ number of steps before reaching top of hill

Return is maximized by minimizing number of steps reach the top of the hill.

Markov Decision Processes

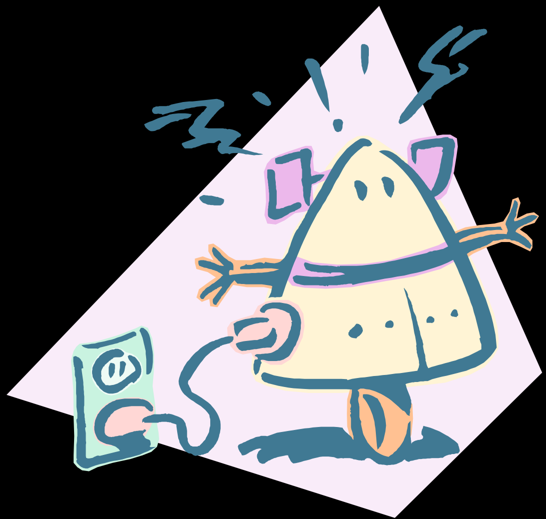
- If an RL task has the Markov Property, it is a Markov Decision Process (MDP)
- If state, action sets are finite, it is a finite MDP
- To define a finite MDP, you need:
  - state and action sets
  - one-step "dynamics" defined by transition probabilities:

$$P_{ss'}^{a} = \Pr\left\{s_{t+1} = s' \mid s_t = s, a_t = a\right\} \quad \text{for all } s, s' \in S,\ a \in A(s).$$

  - reward probabilities:

$$R_{ss'}^{a} = E\left\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\right\} \quad \text{for all } s, s' \in S,\ a \in A(s).$$

An Example Finite MDP



Recycling Robot

- At each step, robot has to decide whether it should
  - (1) actively search for a can,
  - (2) wait for someone to bring it a can, or
  - (3) go to home base and recharge.
- Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: `high`, `low`.
- Reward = number of cans collected

Recycling Robot MDP

$S = \{\texttt{high}, \texttt{low}\}$

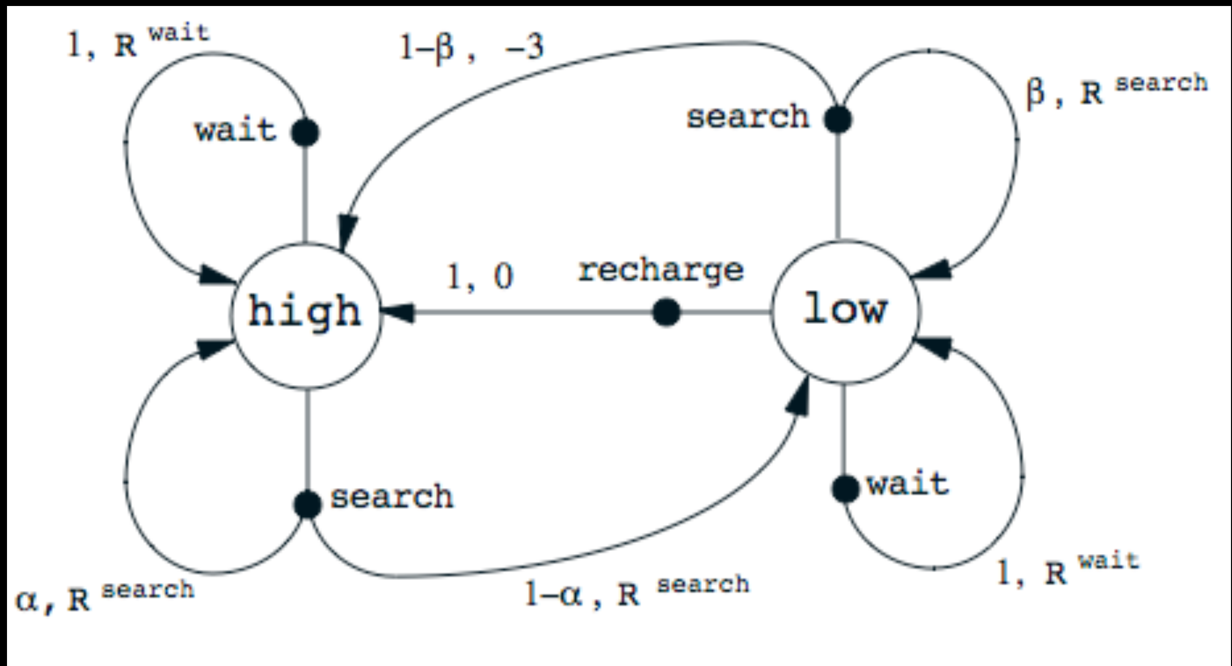
$A(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$A(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

$R^{\texttt{search}}$ = expected no. of cans while searching

$R^{\texttt{wait}}$ = expected no. of cans while waiting

$$R^{\texttt{search}} > R^{\texttt{wait}}$$



Value Functions

- The value of a state = expected return starting from that state; depends on the agent's policy:

State - value function for policy π :

$$V^{\pi}(s) = E_{\pi}\left\{R_t \mid s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

- The value of taking an action in a state under policy π = expected return starting from that state, taking that action, and then following π :

Action - value function for policy π :

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t \mid s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

Bellman Equation for a Policy π

The basic idea:

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

So:

$$\begin{aligned} V^\pi(s) &= E_\pi\left\{ R_t \mid s_t = s \right\} \\ &= E_\pi\left\{ r_{t+1} + \gamma V^\pi\left(s_{t+1}\right) \mid s_t = s \right\} \end{aligned}$$

Or, without the expectation operator:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma V^\pi(s') \right]$$

 [R. S. Sutton and A. G. Barto]

Optimal Value Functions

- For finite MDPs, policies can be partially ordered:

$$\pi \geq \pi' \quad \text{if and only if} \quad V^{\pi}(s) \geq V^{\pi'}(s) \;\; \text{for all } s \in S$$

- Optimal policy = π^*
- Optimal state-value function:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \text{for all } \; s \in S$$

- Optimal action-value function:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \;\; \text{for all } \; s \in S \text{ and } a \in A(s)$$

This is the expected return for taking action *a* in state *s* and thereafter following an optimal policy.

Why Optimal State-Value Functions are Useful

Any policy that is greedy with respect to V^* is an optimal policy.

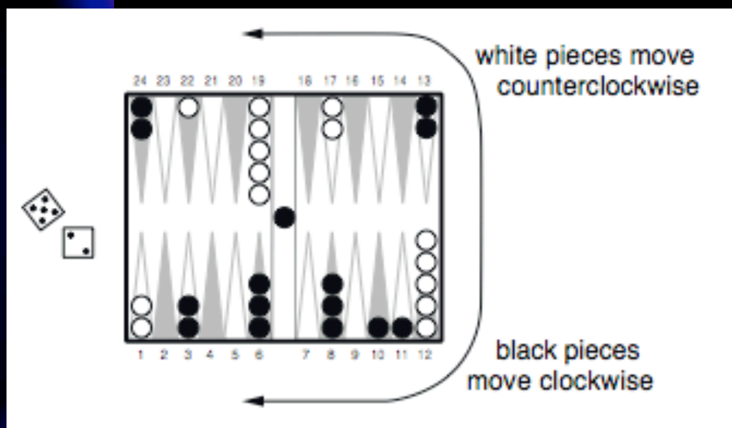
Therefore, given V^*, one-step-ahead search produces the long-term optimal actions.

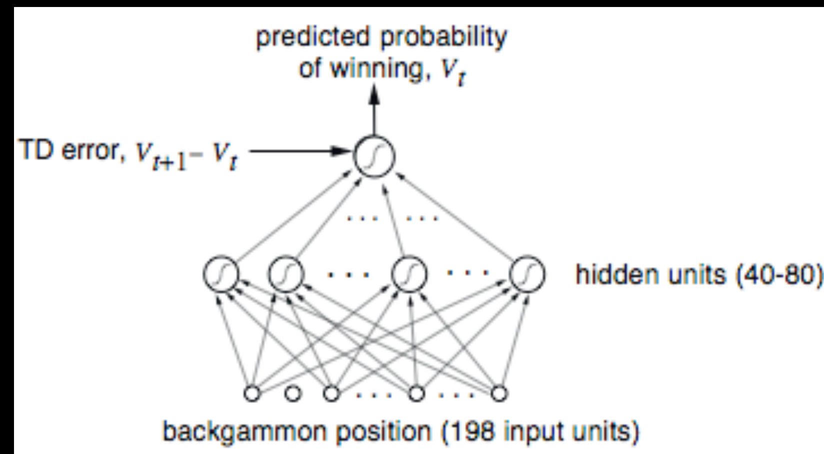
Given Q^*, the agent does not even have to do a one-step-ahead search:

$$\pi^*(s) = \arg\max_{a \in A(s)} Q^*(s,a)$$

TD-Gammon

Tesauro, 1992–1995





Start with a random network

Play very many games against self

Learn a value function from this simulated experience

Action selection by 2–3 ply search

Program	Training games	Opponents	Results
TDG 1.0	300,000	3 experts	-13 pts/51 games
TDG 2.0	800,000	5 experts	-7 pts/38 games
TDG 2.1	1,500,000	1 expert	-1 pt/40 games

Summary: Key Points for Today

- Reinforcement Learning
  - How different from supervised, unsupervised?
- Key components
  - Actions, states, transition probs, rewards
  - Markov Decision Process
  - Episodic vs. continuing tasks
  - Value functions, optimal value functions

Next Time

- Reading
  - Reinforcement Learning
    (read Ch. 16.1-16.5)
  - Reading question volunteers: Lewis, Jimmy, Kevin
- New topic: Ensemble Learning
  - Machine learning algorithms unite!