CS 461: Machine Learning
Lecture 8

Dr. Kiri Wagstaff
kiri.wagstaff@calstatela.edu

Plan for Today

- Review Clustering
- Reinforcement Learning
  - How different from supervised, unsupervised?
- Key components
- How to learn
  - Deterministic
  - Nondeterministic
- Homework 4 Solution

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

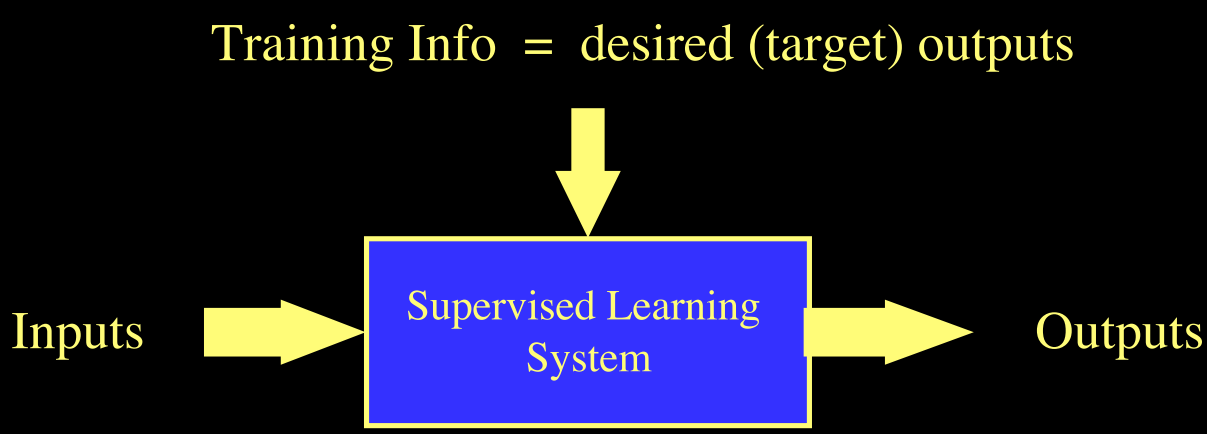
Reinforcement Learning

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



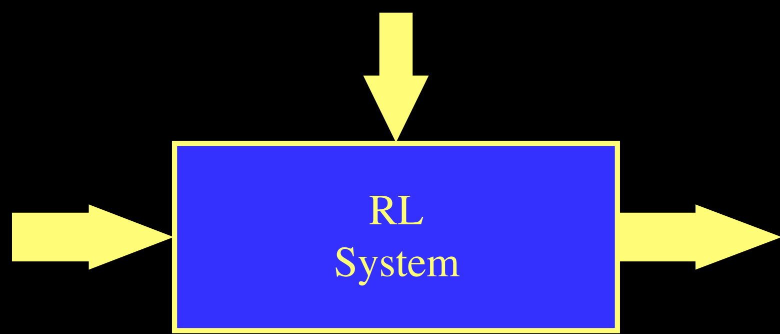
Error = (target output – actual output)

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

Reinforcement Learning

Training Info = evaluations ("rewards" / "penalties")

Inputs



Outputs ("actions")

Objective: get as much reward as possible

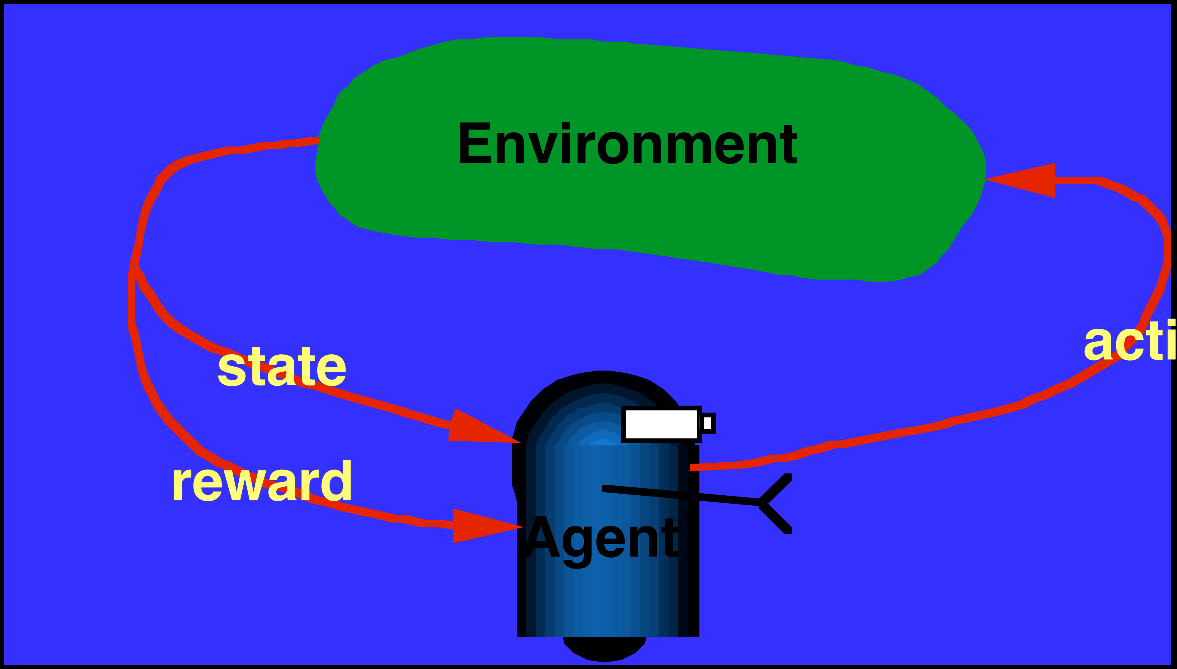
 [R. S. Sutton and A. G. Barto]

Key Features of RL

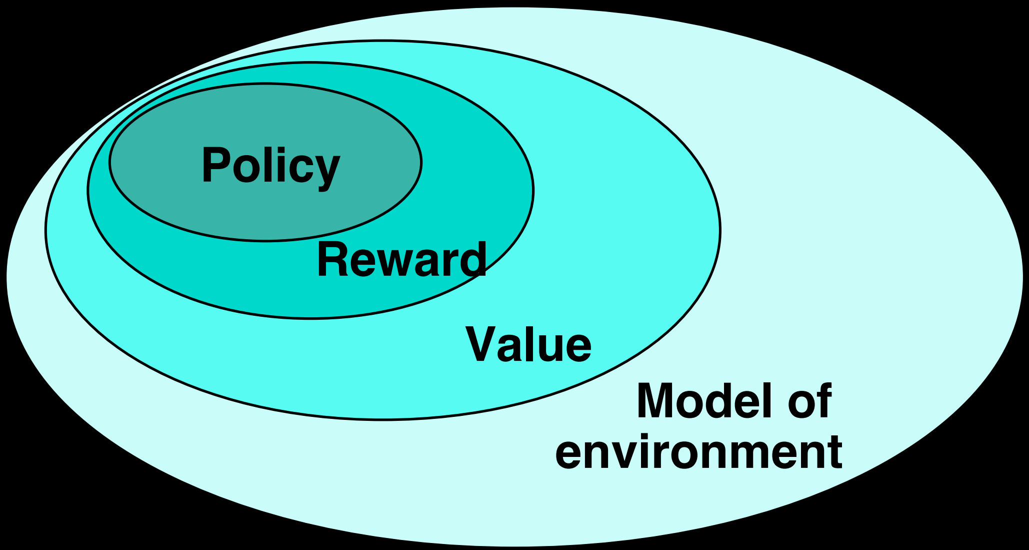
- Learner is not told which actions to take
- Trial-and-Error search
- Possibility of delayed reward
  - Sacrifice short-term gains for greater long-term gains
- The need to *explore* and *exploit*
- Considers the whole problem of a goal-directed agent interacting with an uncertain environment

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



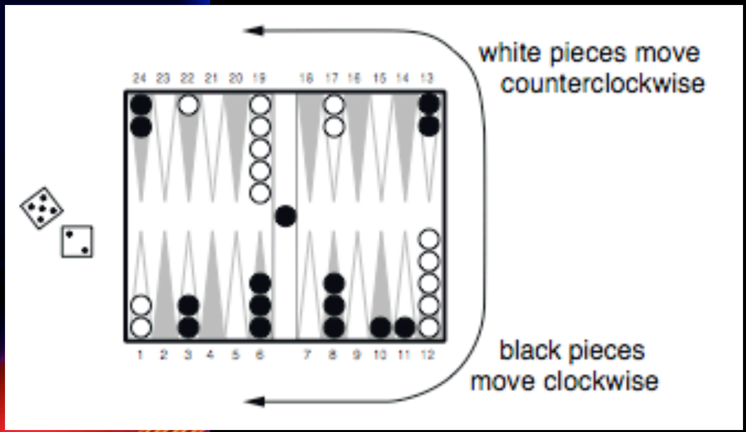
- Policy: what to do
- Reward: what is good
- Value: what is good because it *predicts* reward
- Model: what follows what

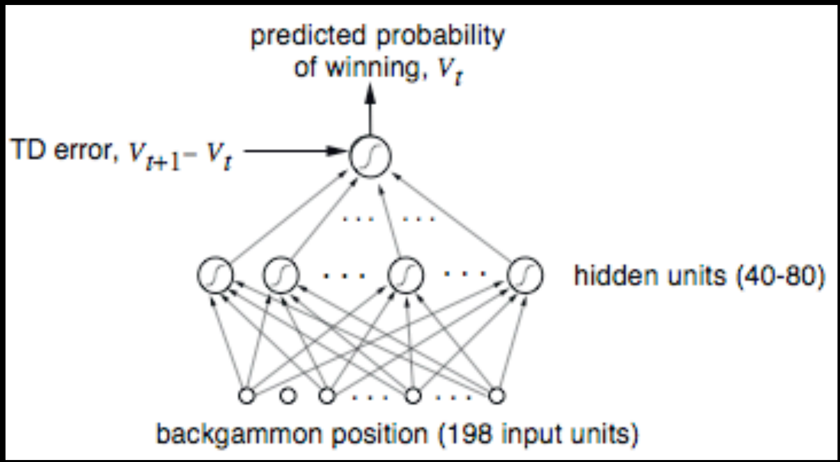
Some Notable RL Applications

- TD-Gammon: Tesauro
  - world's best backgammon program
- Elevator Control: Crites & Barto
  - high performance down-peak elevator controller
- Inventory Management: Van Roy, Bertsekas, Lee, & Tsitsiklis
  - 10–15% improvement over industry standard methods
- Dynamic Channel Assignment: Singh & Bertsekas, Nie & Haykin
  - high performance assignment of radio channels to mobile telephone calls

TD-Gammon

Tesauro, 1992–1995





Action selection by 2–3 ply search

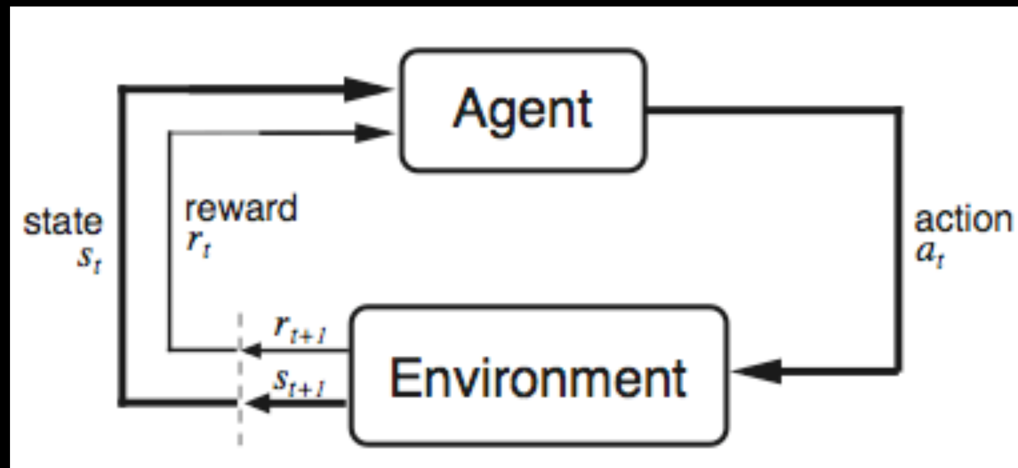
Start with a random network

Play very many games against self

Learn a value function from this simulated experience

This produces arguably the best player in the world

The Agent-Environment Interface



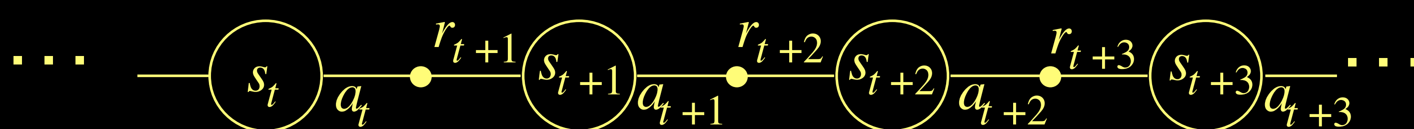
Agent and environment interact at discrete time steps: $t = 0, 1, 2, \ldots$

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 goal is to get as much reward as it can over the long run.

Getting the Degree of Abstraction Right

- Time: steps need not refer to fixed intervals of real time.
- Actions:
  - Low level (e.g., voltages to motors)
  - High level (e.g., accept a job offer)
  - "Mental" (e.g., shift in focus of attention), etc.
- States:
  - Low-level "sensations"
  - Abstract, symbolic, based on memory, or subjective
    - e.g., the state of being "surprised" or "lost"
- The environment is not necessarily unknown to the agent, only incompletely controllable
- Reward computation is in the agent's environment because the agent cannot change it arbitrarily

Goals and Rewards

- Goal 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.

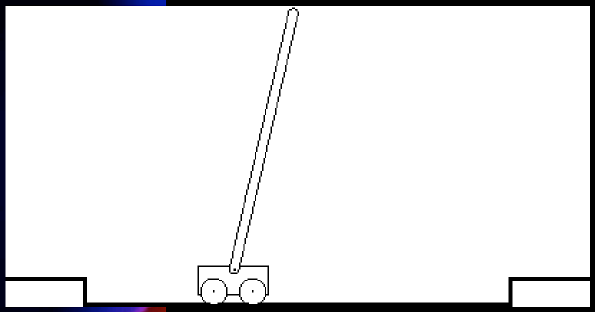
Discounted return:

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

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:

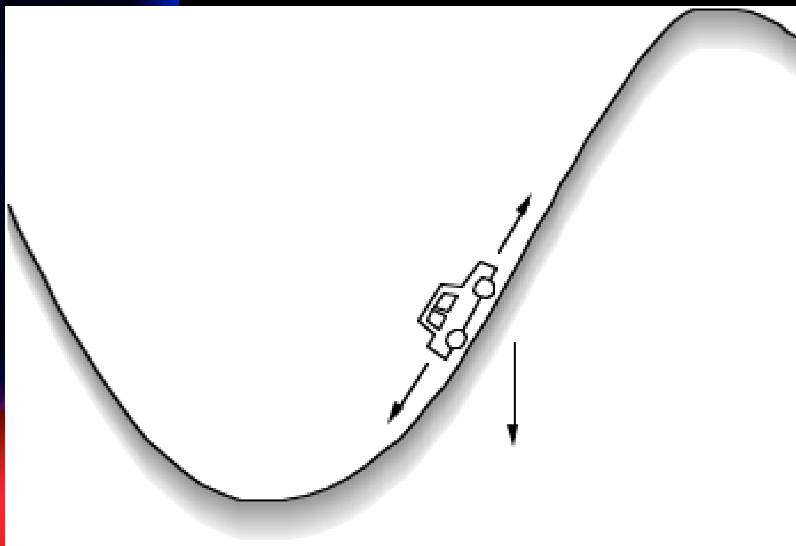
$$\text{reward} = +1 \text{ for each step before failure}$$
$$\Rightarrow \text{return} = \text{number of steps before failure}$$

As a continuing task with discounted return:

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

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.

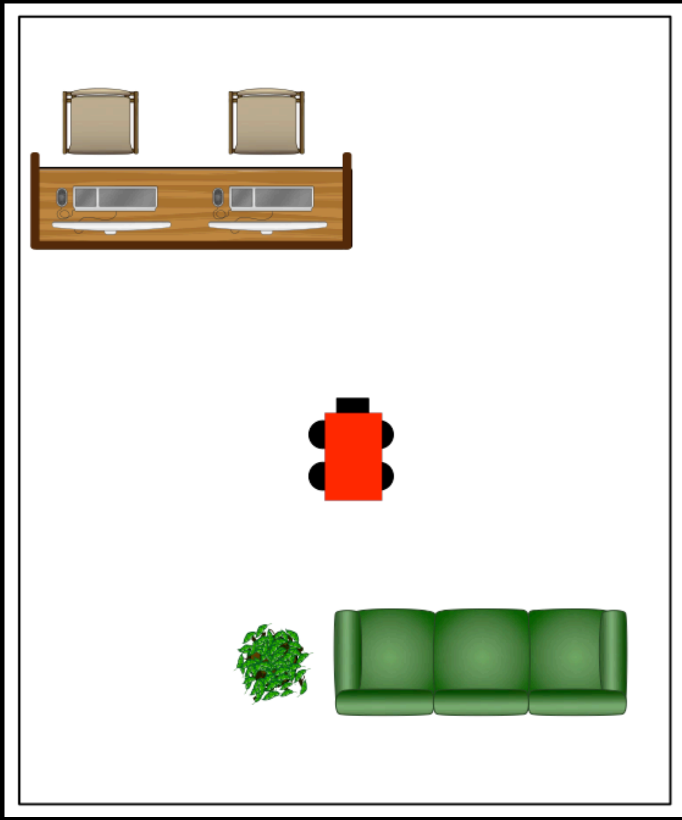
reward = –1 for each step where **not** at top of hill

⇒ return = – number of steps before reaching top of hill

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

Markovian Examples

Robot navigation



Settlers of Catan

- State does contain
  - board layout
  - location of all settlements and cities
  - your resource cards
  - your development cards
  - Memory of past resources acquired by opponents
- State does *not* contain:
  - Knowledge of opponents' development cards
  - Opponent's internal development plans

Markov Decision Processes

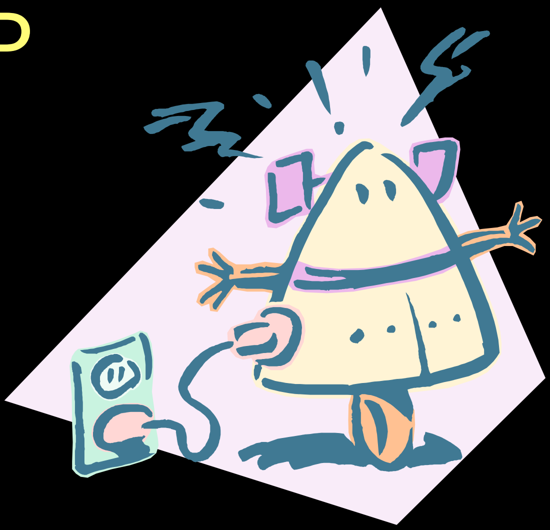
- If an RL task has the Markov Property, it is a Markov Decision Process (MDP)
- If state, action sets are finite, it is a finite MDP
- To define a finite MDP, you need:
  - state and action sets
  - one-step "dynamics" defined by transition probabilities:

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

  - 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

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

Recycling Robot MDP

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

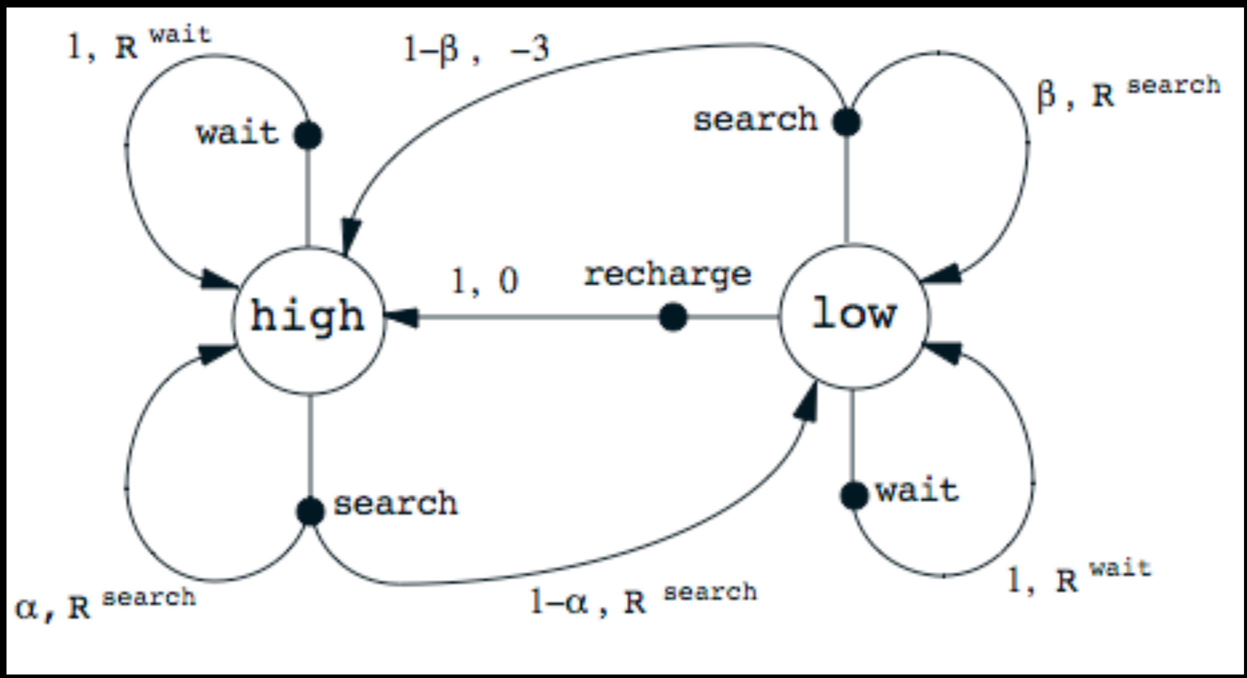
$A(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

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

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

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

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



Example: Drive a car

- States?
- Actions?
- Goal?
- Next-state probs?
- Reward probs?

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:

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

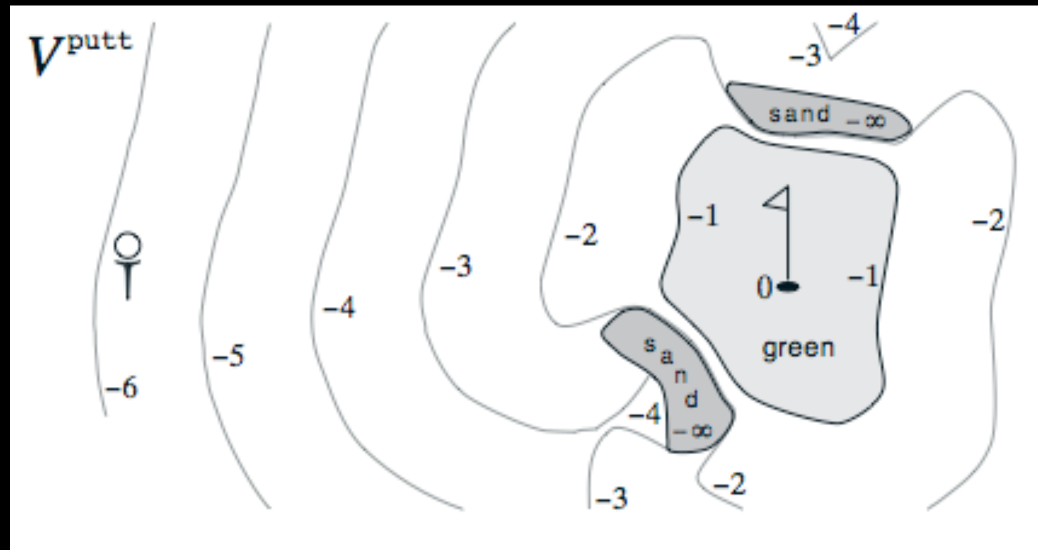
So:

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

Or, without the expectation operator:

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

Golf



- State is ball location
- Reward of –1 for each stroke until the ball is in the hole
- Value of a state?
- Actions:
  - `putt` (use putter)
  - `driver` (use driver)
- `putt` succeeds anywhere on the green

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:

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

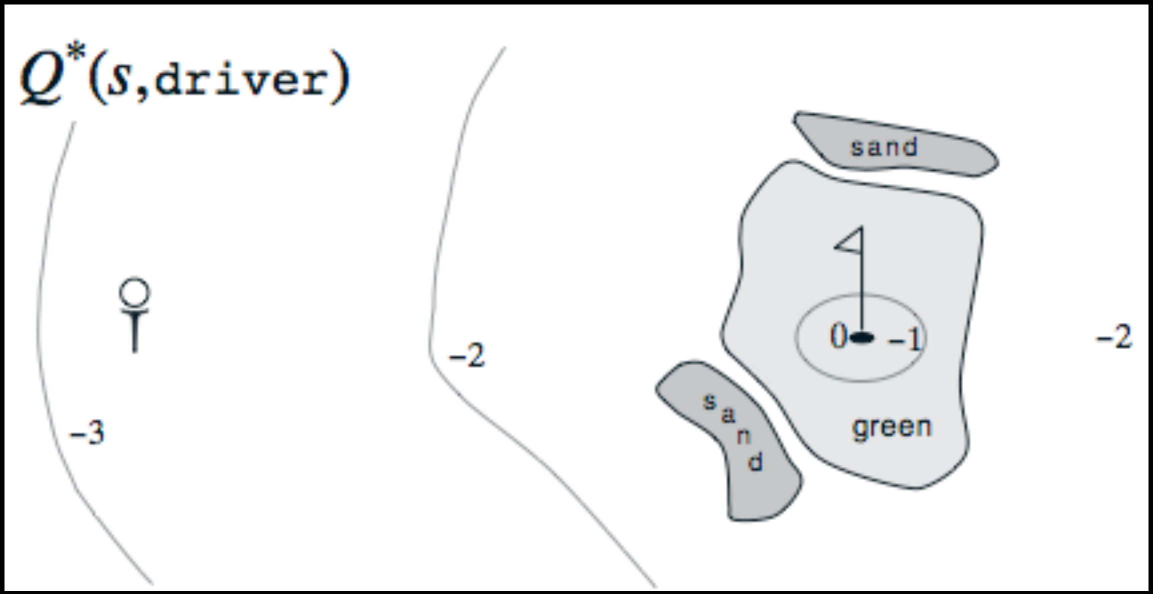
- Optimal action-value function:

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

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

Optimal Value Function for Golf

- We can hit the ball farther with `driver` than with `putter`, but with less accuracy
- $Q^*(s,\texttt{driver})$ gives the value of using `driver` first, then using whichever actions are best



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

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:

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

Summary so far...

- Agent-environment interaction
  - States
  - Actions
  - Rewards
- Policy: stochastic rule for selecting actions
- Return: the function of future rewards agent tries to maximize
- Episodic and continuing tasks
- Markov Decision Process
  - Transition probabilities
  - Expected rewards
- Value functions
  - State-value fn for a policy
  - Action-value fn for a policy
  - Optimal state-value fn
  - Optimal action-value fn
- Optimal value functions
- Optimal policies
- Bellman Equation

Model-Based Learning

- Environment, $P(s_{t+1} \mid s_t, a_t)$, $p(r_{t+1} \mid s_t, a_t)$, is known
- There is no need for exploration
- Can be solved using dynamic programming
- Solve for

$$V^*(s_t) = \max_{a_t}\left(E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V^*(s_{t+1}) \right)$$

- Optimal policy

$$\pi^*(s_t) = \arg\max_{a_t}\left(E[r_{t+1} \mid s_t, a_t] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V^*(s_{t+1}) \right)$$

Value Iteration

Initialize $V(s)$ to arbitrary values
Repeat
    For all $s \in \mathcal{S}$
        For all $a \in \mathcal{A}$
            $Q(s,a) \leftarrow E[r|s,a] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s')$
        $V(s) \leftarrow \max_a Q(s,a)$
Until $V(s)$ converge

Policy Iteration

Initialize a policy π arbitrarily

Repeat

$\pi \leftarrow \pi'$

Compute the values using π by

solving the linear equations

$$V^{\pi}(s) = E[r|s, \pi(s)] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^{\pi}(s')$$

Improve the policy at each state

$$\pi'(s) \leftarrow \arg\max_a (E[r|s, a] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi}(s'))$$

Until $\pi = \pi'$

Temporal Difference Learning

- Environment, $P(s_{t+1} \mid s_t, a_t)$, $p(r_{t+1} \mid s_t, a_t)$, is not known; model-free learning
- There is need for exploration to sample from $P(s_{t+1} \mid s_t, a_t)$ and $p(r_{t+1} \mid s_t, a_t)$
- Use the reward received in the next time step to update the value of current state (action)
- The temporal difference between the value of the current action and the value discounted from the next state

Exploration Strategies

- ϵ-greedy:
  - With prob ϵ,choose one action at random uniformly
  - Choose the best action with pr 1-ϵ
- Probabilistic (softmax: all p > 0):

$$P(a \mid s) = \frac{\exp Q(s,a)}{\sum_{b=1}^{\mathcal{A}} \exp Q(s,b)}$$

- Move smoothly from exploration/exploitation
- Annealing: gradually reduce T

$$P(a \mid s) = \frac{\exp\left[Q(s,a)/T\right]}{\sum_{b=1}^{\mathcal{A}} \exp\left[Q(s,b)/T\right]}$$

Deterministic Rewards and Actions

- Deterministic: single possible reward and next state

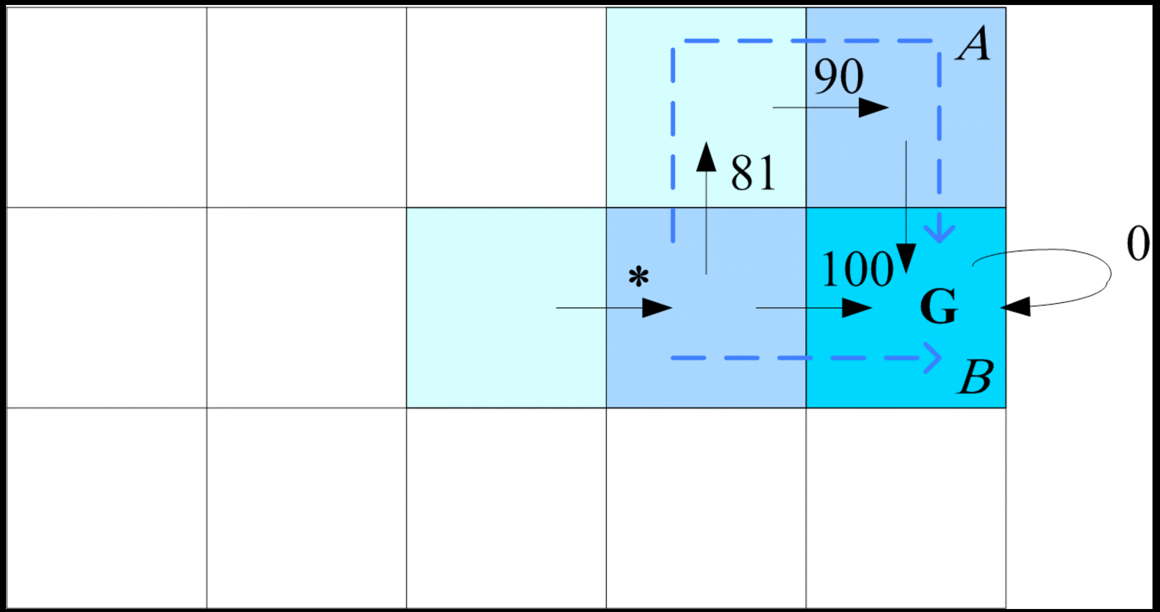
$$Q(s_t, a_t) = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

- Used as an update rule (backup)

$$\hat{Q}(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$$

- Updates happen only after reaching the reward (then are "backed up")

Starting at zero, *Q* values increase, never decrease



γ=0.9

Consider the value of action marked by '*':

If path A is seen first, Q(*)=0.9*max(0,81)=73

Then B is seen, Q(*)=0.9*max(100,81)=90

Or,

If path B is seen first, Q(*)=0.9*max(100,0)=90

Then A is seen, Q(*)=0.9*max(100,81)=90

Q values increase but never decrease

Nondeterministic Rewards and Actions

- When next states and rewards are nondeterministic (there is an opponent or randomness in the environment), we keep averages (expected values) instead as assignments
- Q-learning (Watkins and Dayan, 1992):

$$\hat{Q}(s_t,a_t) \leftarrow \hat{Q}(s_t,a_t) + \eta\left(\boxed{r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1},a_{t+1})} - \hat{Q}(s_t,a_t)\right)$$

backup

- Learning *V* (TD-learning: Sutton, 1988)

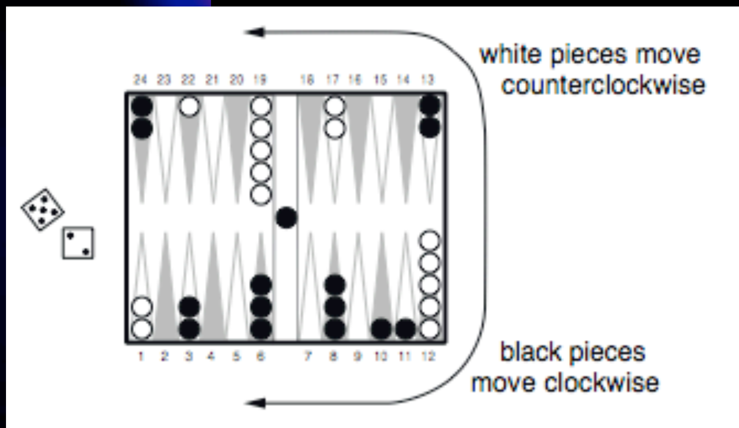
$$V(s_t) \leftarrow V(s_t) + \eta\left(\boxed{r_{t+1} + \gamma V(s_{t+1})} - V(s_t)\right)$$

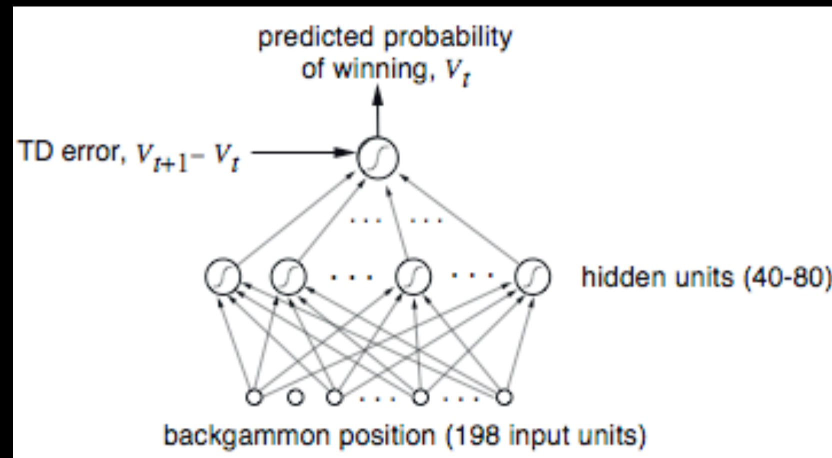
Q-learning

Initialize all $Q(s,a)$ arbitrarily
For all episodes
  Initalize s
  Repeat
    Choose a using policy derived from Q, e.g., ϵ-greedy
    Take action a, observe r and s'
    Update $Q(s,a)$:
      $$Q(s,a) \leftarrow Q(s,a) + \eta(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$
    $s \leftarrow s'$
  Until s is terminal state

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
- Learn: policy (based on V, Q)
  - Model-based: value iteration, policy iteration
  - TD learning
    - Deterministic: backup rules (max)
    - Nondeterministic: TD learning, Q-learning (running avg)

Homework 4 Solution

Next Time

- Ensemble Learning
  (read Ch. 15.1-15.5)

- Reading questions are posted on website