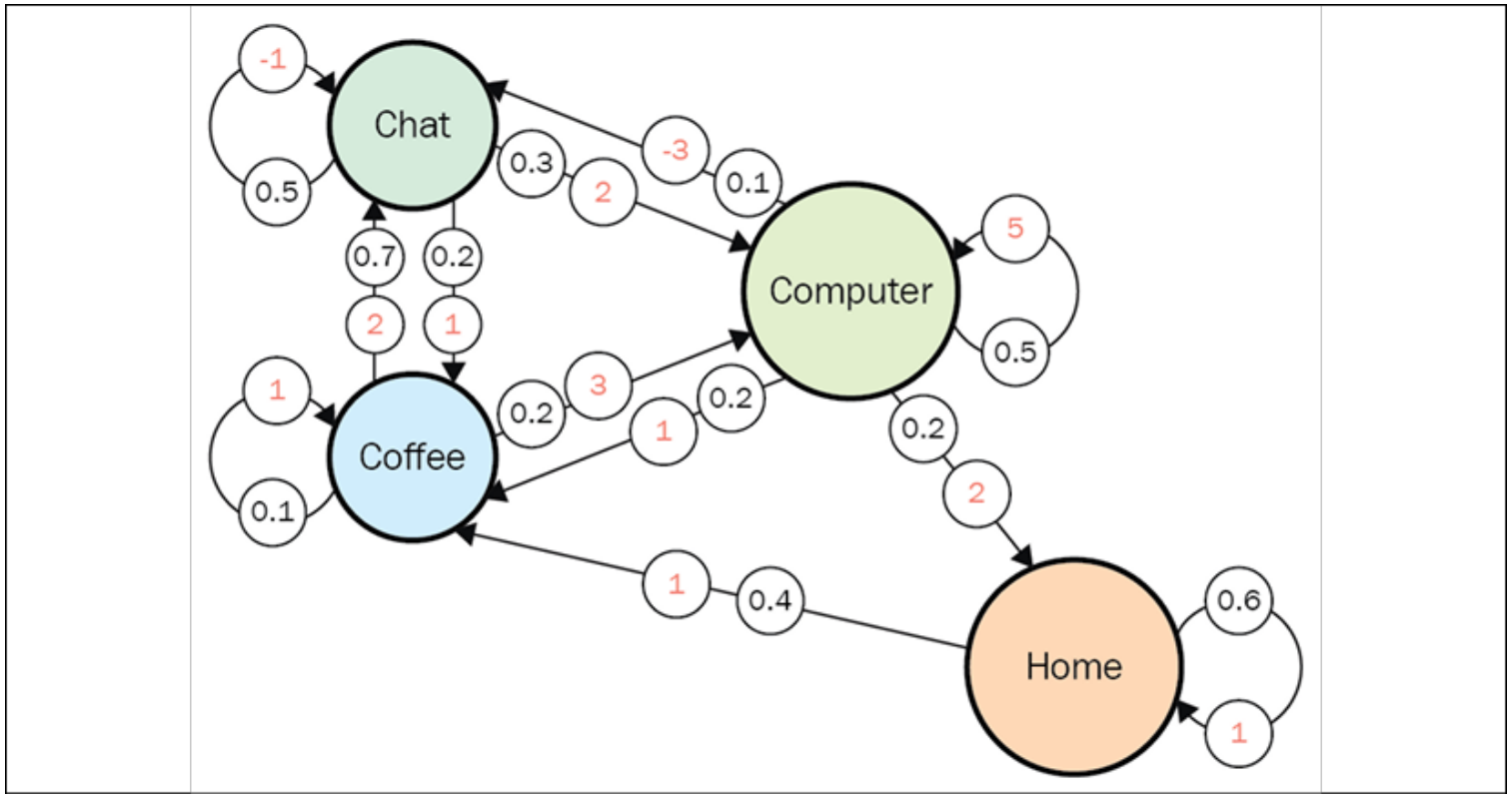
THE UNIVERSITY of TENNESSEE KNOXVILLE

# Lecture 24: Reinforcement Learning

Instructor: Sergei V. Kalinin

# And rewards


Chat
Computer
Coffee
Home
-1
0.5
0.3
2
-3
0.1
5
0.5
0.7
2
0.2
1
1
0.1
0.2
3
1
0.2
0.2
2
1
0.4
0.6
1

# Dynamic Programming

**State Transition Probabilities:** $P(s'|s,a)$— the probability of transitioning to state $s'$ when taking action $a$ from state $s$.

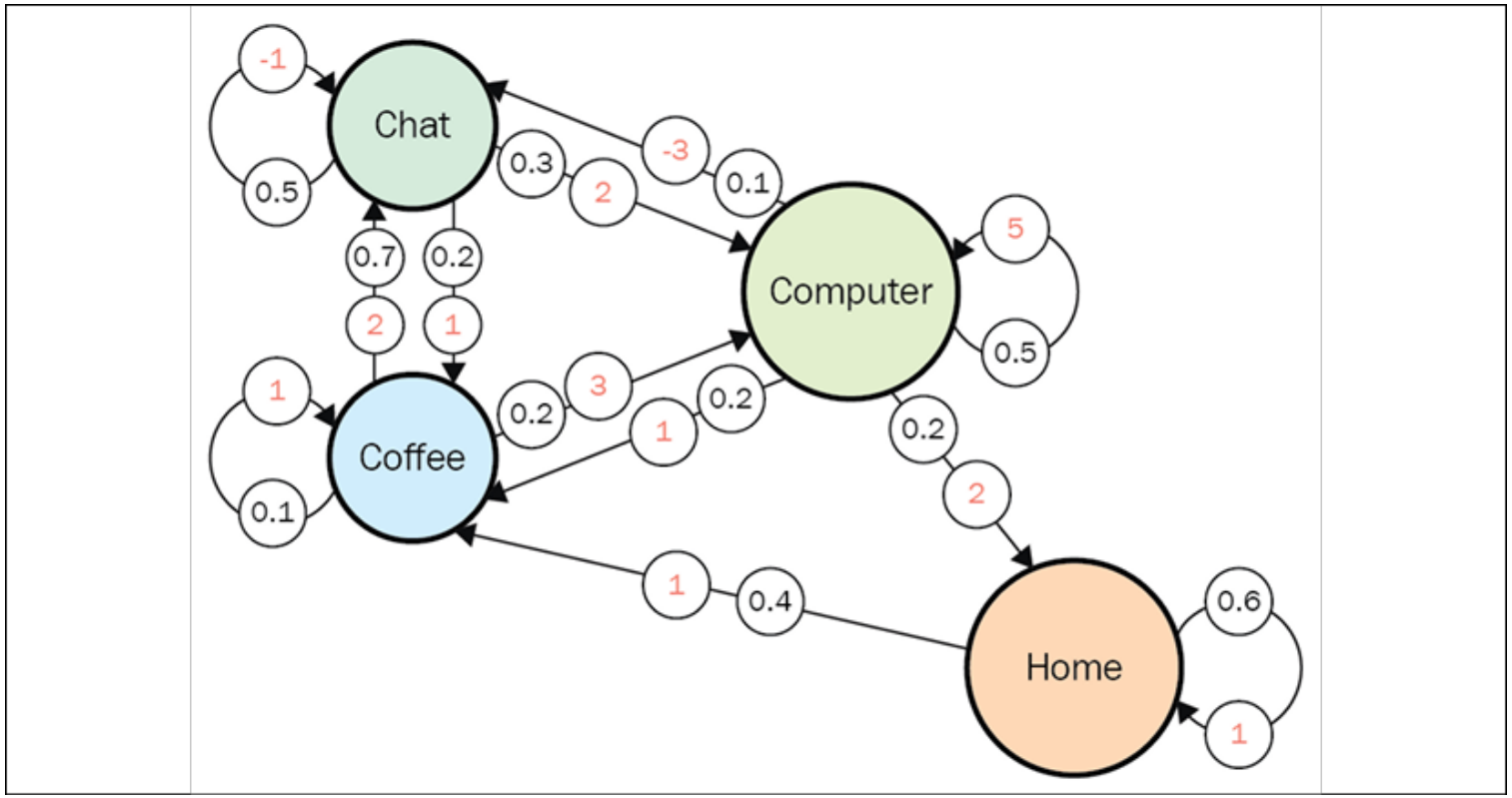
**Rewards:** $R(s,a)$— the expected reward for taking action $a$ in state $s$.

**Policy:** A mapping $\pi(s)$ that specifies the action to take in each state $s$.

**State Value Function:** $V(s)$ — the expected total reward starting from state $s$, following a policy.

**Action Value Function:** $Q(s,a)$ — the expected total reward starting from state $s$, taking action $a$, and then following a policy.

# Now we can get values:




- V(chat) = −1 * 0.5 + 2 * 0.3 + 1 * 0.2 = 0.3
- V(coffee) = 2 * 0.7 + 1 * 0.1 + 3 * 0.2 = 2.1
- V(home) = 1 * 0.6 + 1 * 0.4 = 1.0
- V(computer) = 5 * 0.5 + (−3) * 0.1 + 1 * 0.2 + 2 * 0.2 = 2.8

# Computing Return From Rewards

- Episodic (vs. continuing) tasks
  - "game over" after N steps
  - optimal policy depends on N; harder to analyze

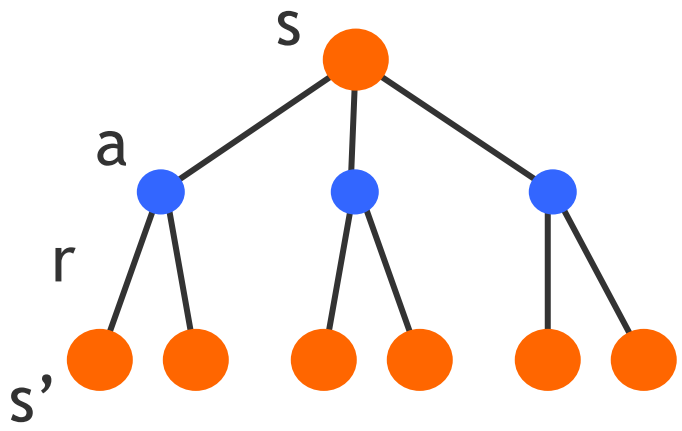
- Additive rewards
  - $V(s_0, s_1, \ldots) = r(s_0) + r(s_1) + r(s_2) + \ldots$
  - infinite value for continuing tasks

- Discounted rewards
  - $V(s_0, s_1, \ldots) = r(s_0) + \gamma * r(s_1) + \gamma^2 * r(s_2) + \ldots$
  - value bounded if rewards bounded

From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

# Value Function

- State value function: $V^{\pi}(s)$
  - expected return when starting in $s$ and following $\pi$
- State-action value function: $Q^{\pi}(s,a)$
  - expected return when starting in $s$, performing $a$, and following $\pi$
- Useful for finding the optimal policy
  - can estimate from experience
  - pick the best action using $Q^{\pi}(s,a)$
- Bellman equation




$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[ r^{a}_{ss'} + \gamma V^{\pi}(s') \right] = \sum_{a} \pi(s,a) Q^{\pi}(s,a)$$

From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

# Optimal Value Function

- There's a set of *optimal* policies
  - $V^{\pi}$ defines partial ordering on policies
  - they share the same optimal value function

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

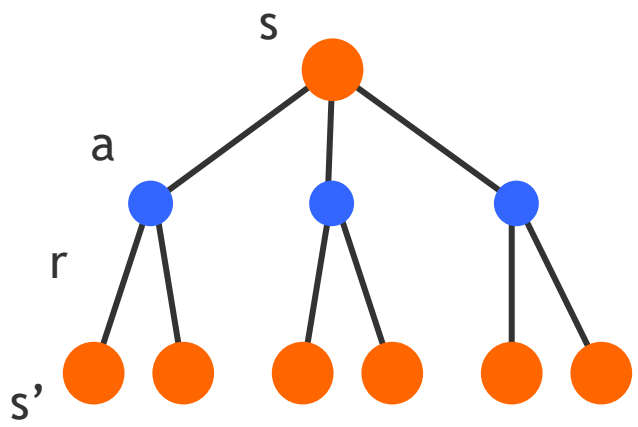
- Bellman optimality equation

$$V^*(s) = \max_{a} \sum_{s'} P^a_{ss'} \left[ r^a_{ss'} + \gamma V^*(s') \right]$$

  - system of *n* non-linear equations
  - solve for V*(s)
  - easy to extract the optimal policy

- Having Q*(*s*,*a*) makes it even simpler

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




From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

# Value Iteration

Value Iteration finds the optimal value function by iteratively improving the estimates of $V(s)$ using the Bellman Optimality Equation. Once the optimal value function $V^*(s)$ is found, the optimal policy $\pi^*(s)$ can be derived by choosing the action that maximizes the expected return at each state.

**Initialize:** Start with an arbitrary value function $V(s)=0$ for all states.

**Update:** Iteratively update the value for each state using the Bellman equation:

$$V_{k+1}(s)=\max_a \sum_{s'} P(s'|s,a)[R(s,a)+\gamma V_k(s')]$$

Where:

- $\gamma$ is the discount factor (between 0 and 1)
- $P(s'|s,a)$ is the transition probability.
- $R(s,a)$ is the immediate reward
- $V_k(s)$ is the value function at iteration $k$

# Value Iteration

**Stop Condition:** Continue until $V(s)$ converges

**Extract Policy:** Once $V^*(s)$ converges, derive the optimal policy $\pi^*(s)$ by selecting the action that maximizes the expected return:

$$\pi^*(s)=\underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s,a)[R(s,a)+\gamma V^*(s')]$$

# Policy Iteration

Policy Iteration alternates between policy evaluation and policy improvement steps to iteratively converge to the optimal policy. Unlike value iteration, where values are updated for all actions, policy iteration evaluates the current policy and then improves it.

**Policy Evaluation:** Given a policy $\pi(s)$, compute the value function $V^{\pi}(s)$ by solving the Bellman equation for the fixed policy:

$$V(s)=\sum_{s'}P(s'|s,\pi(s))[R(s,\pi(s))+\gamma V(s')]$$

This can be solved iteratively or by solving a system of linear equations.

# Policy Iteration

**Policy Improvement:** Update the policy by choosing the action that maximizes the expected return at each state:

$$\pi'(s) = \arg\max_{a} \sum_{s'} P(s'|s,a)[R(s,a) + \gamma V(s')]$$

**Stop Condition:** If the policy doesn't change after an improvement step, stop. Otherwise, repeat the evaluation and improvement steps.

# Value Iteration vs. Policy Iteration

**Value Iteration** iteratively updates the value function using the Bellman Optimality Equation until convergence, then derives the optimal policy.

**Policy Iteration** alternates between evaluating the current policy and improving it until the policy stops changing.

Both methods work when the MDP's transition probabilities and rewards are known, and they converge to the optimal policy $\pi^*(s)$ and optimal value function $V^*(s)$. Value Iteration is more efficient when fewer iterations are needed, while Policy Iteration can be more stable in some environments.

# Using Dynamic Programming

- Need complete model of the environment and rewards
  - robot in a room
    - state space, action space, transition model

- can we use DP to solve
  - robot in a room?
  - back gammon?
  - helicopter?

From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

# Reinforcement Learning

- No knowledge of environment
  - Can only act in the world and observe states and reward
- Many factors make RL difficult:
  - Actions have non-deterministic effects
    - Which are initially unknown
  - Rewards / punishments are infrequent
    - Often at the end of long sequences of actions
    - How do we determine what action(s) were really responsible for reward or punishment? (credit assignment)
  - World is large and complex
- Nevertheless learner must decide what actions to take
  - We will assume the world behaves as an MDP

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

# Monte Carlo Methods

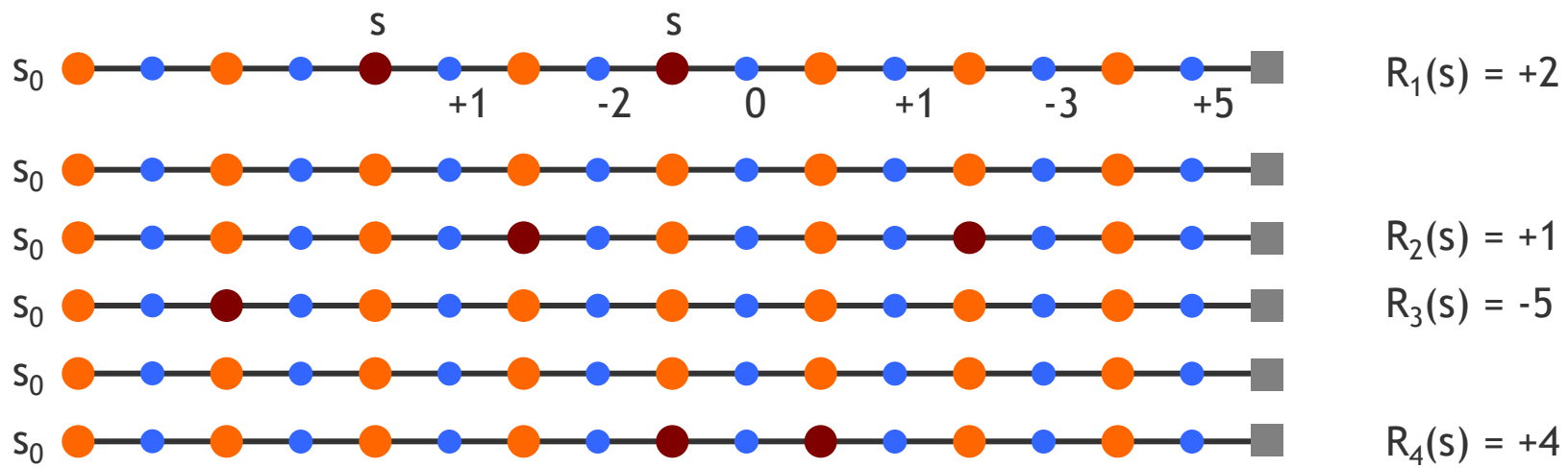
- Don’t need full knowledge of environment
  - just experience, or
  - simulated experience
- But similar to DP
  - policy evaluation, policy improvement
- Averaging sample returns
  - defined only for episodic tasks

From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

# Monte Carlo Policy Evaluation

- Want to estimate $V^{\pi}(s)$
  = expected return starting from s and following $\pi$
  – estimate as average of observed returns in state s

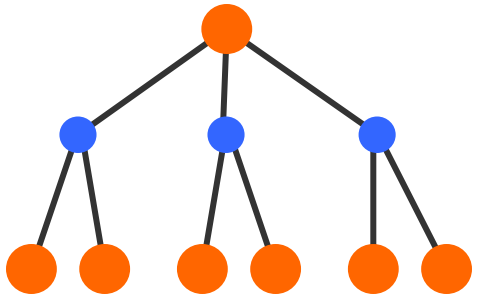
- First-visit MC
  – average returns following the first visit to state s




$V^{\pi}(s) \approx (2 + 1 - 5 + 4)/4 = 0.5$

# Monte Carlo Control

- $V^\pi$ not enough for policy improvement
  - need exact model of environment



- estimate $Q^\pi(s,a)$

$$\pi'(s) = \arg\max_a Q^\pi(s,a)$$

- MC control
  - update after each episode

$$\pi_0 \rightarrow^E Q^{\pi_0} \rightarrow^I \pi_1 \rightarrow^E Q^{\pi_1} \rightarrow^I \ldots \rightarrow^I \pi^* \rightarrow^E Q^*$$

- non-stationary environment

$$V(s) \leftarrow V(s) + \alpha\,[R - V(s)]$$

- a problem
  - greedy policy won't explore all actions

From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

# Maintaining Exploration

- Deterministic/greedy policy won't explore all actions
  - don't know anything about the environment at the beginning
  - need to try all actions to find the optimal one

- Maintain exploration
  - use *soft* policies instead: $\pi(s,a)>0$ (for all s,a)

- ε-greedy policy
  - with probability 1-ε perform the optimal/greedy action
  - with probability ε perform a random action

  - will keep exploring the environment
  - slowly move it towards greedy policy: ε -> 0

From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

# Summary of Monte Carlo

- don't need model of environment
  - averaging of sample returns
  - only for episodic tasks
- learn from sample episodes or simulated experience
- can concentrate on "important" states
  - don't need a full sweep
- need to maintain exploration
  - use soft policies

From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

# Temporal Difference Learning

- Combines ideas from MC and DP
  - like MC: learn directly from experience (don't need a model)
  - like DP: learn from values of successors
  - works for continuous tasks, usually faster than MC

- Constant-alpha MC:
  - have to wait until the end of episode to update

$$V(s_t) \leftarrow V(s_t) + \alpha\,[R_t - V(s_t)]$$

target

- Simplest TD
  - update after every step, based on the successor

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

# MC vs. TD

- observed the following 8 episodes:

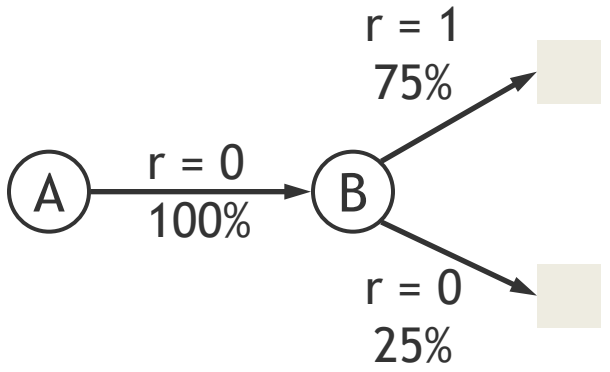
  A – 0, B – 0  B – 1  B – 1  B - 1

  B – 1  B – 1  B – 1  B – 0

- MC and TD agree on V(B) = 3/4

- MC: V(A) = 0
  - converges to values that minimize the error on training data

- TD: V(A) = 3/4
  - converges to ML estimate of the Markov process




From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

# SARSA

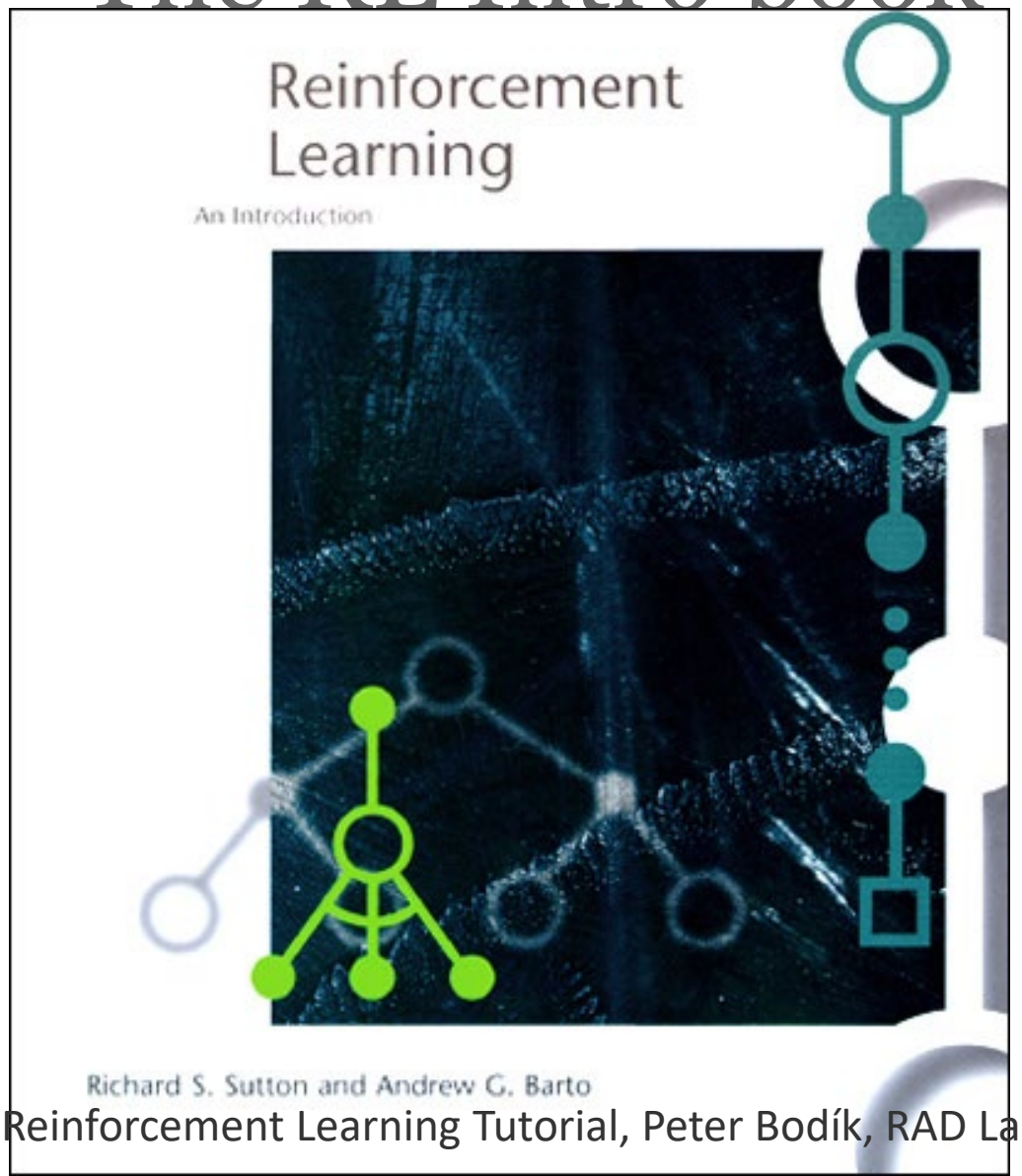
- again, need Q(s,a), not just V(s)


$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

- control
  - start with a random policy
  - update Q and $\pi$ after each step
  - again, need $\varepsilon$-soft policies

From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

# The RL Intro book




Richard Sutton, Andrew Barto
Reinforcement Learning,
An Introduction

http://www.cs.ualberta.ca/~sutton/book/the-book.html

From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley