# CSE 574 Lecture 13: Reinforcement Learning I

**Professor:** Stephanie Gil
**Email:** sgil@asu.edu (Office hours M 12-1pm BYENG 386)
**TAs:** Sushmita Bhattacharya sbhatt55@asu.edu (Office hours M 5-6 BYENG 392)
Weiying Wang wwang239@asu.edu (Office hours Th 2:30-3:30 BYENG 392)

# Intro to Reinforcement Learning

- Why reinforcement learning?
  - Complex domains – hard to model the value of a large number of inputs
  - Much easier to provide feedback of "you did well" or "you did poorly" and *learn* an evaluation function

- Basic framework
  - Markov Decision Procress
  - Fully observable environment
  - Probabilistic action outcomes
  - We do *not* know:
    - How the environment works (probability distributions)
    - What actions do (state transition functions)

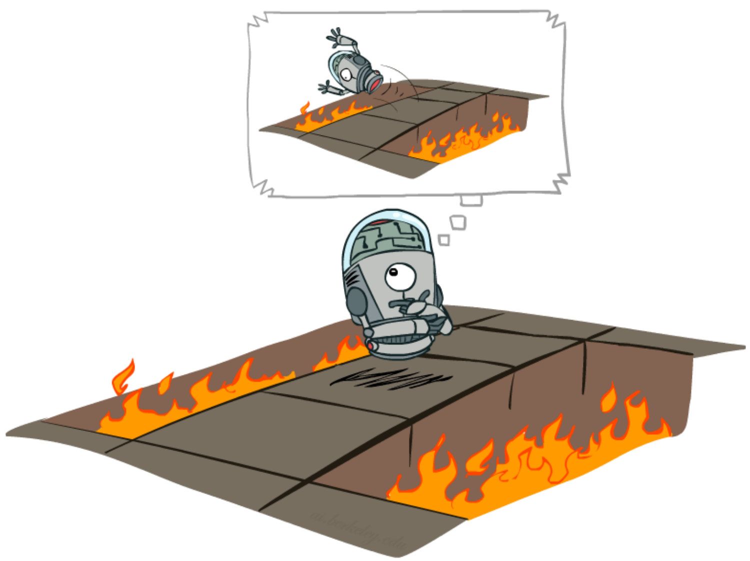
# Different Agent Designs

- **Utility-based agent:** learns a utility function on states and uses this to select actions that maximize the expected outcome utility
  - *Does* require a model of the environment (legal moves and transition model to other states)

- **Q-learning agent:** learns an action-utility function or *Q-function* giving the expected utility of taking a given action in a given state
  - *Does not* require a model of the environment as it does not know where their actions lead.
  - No ability to look ahead

- **Reflex agent:** learns a policy that maps directly from states to actions

# Offline (MDPs) vs. Online (RL)



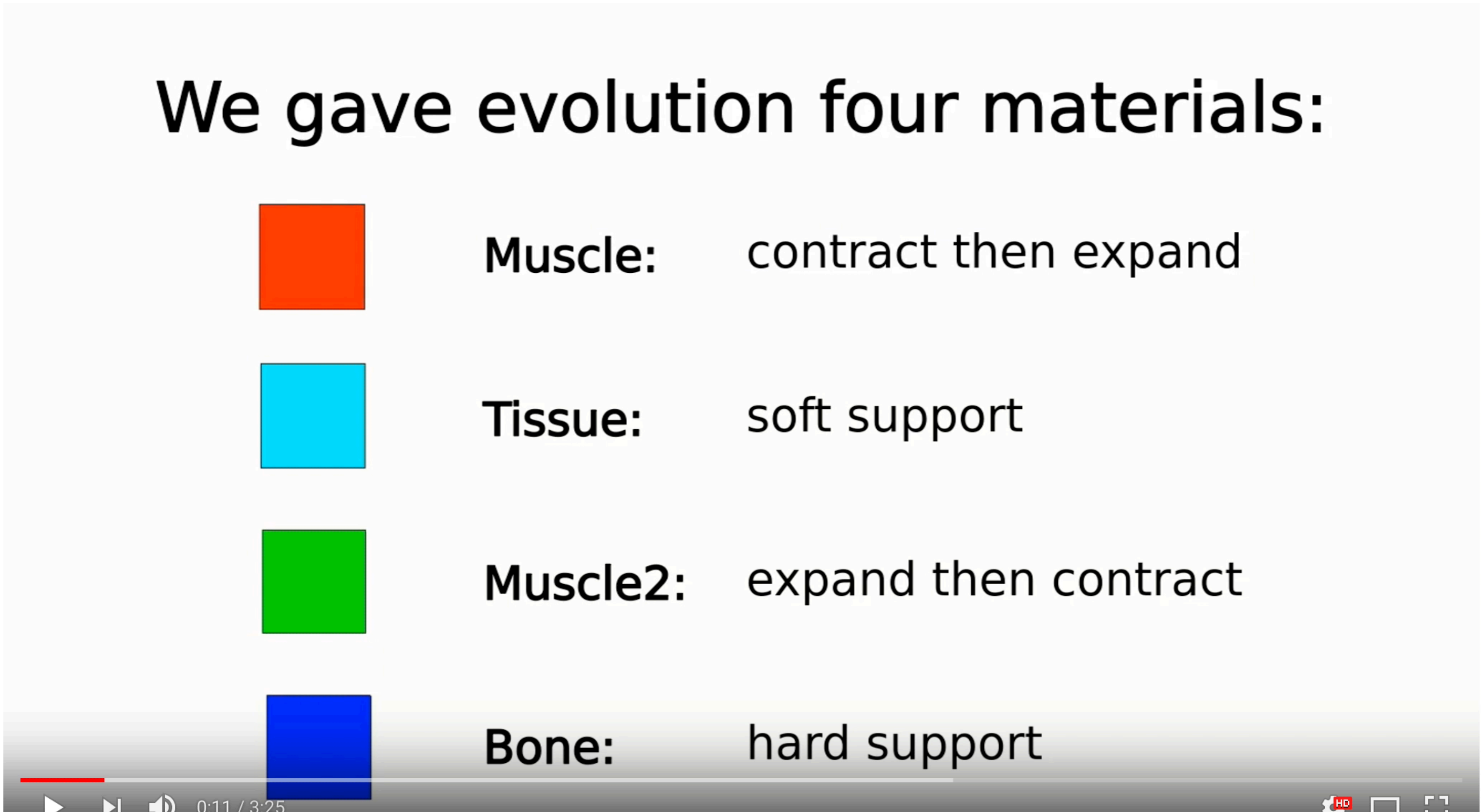
Offline Solution

Online Learning

Slide source: Berkeley CS 188 Lecture 10

# Recall the Utility Function

- Hod Lipson 2013 – genetic algorithms and



# Utility Estimation

- Agent's policy is fixed
  - Rule: in state s, always execute the action
  - Goal: learn the utility function

- You've seen this before! → Policy evaluation

# Example: Gridworld

- Execute trials in the environment using policy

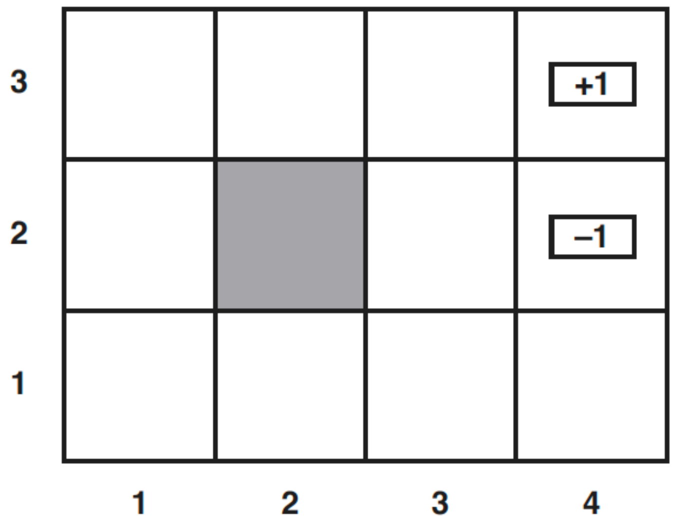



Figure 21.1 from Rusell and Norvig text

$(1,1)_{-0.04}$, $(1,2)_{-0.04}$, $(1,3)_{-0.04}$, $(1,2)_{-0.04}$, $(1,3)_{-0.04}$, $(2,3)_{-0.04}$, $(3,3)_{-0.04}$, $(4,3)_{+1}$

$(1,1)_{-0.04}$, $(1,2)_{-0.04}$, $(1,3)_{-0.04}$, $(2,3)_{-0.04}$, $(3,3)_{-0.04}$, $(3,2)_{-0.04}$, $(3,3)_{-0.04}$, $(4,3)_{+1}$

$(1,1)_{-0.04}$, $(2,1)_{-0.04}$, $(3,1)_{-0.04}$, $(3,2)_{-0.04}$, $(4,2)_{-1}$

# Direct Utility Estimation

- Executing trials gives us several observations from which we can learn the expected reward of executing a given policy for each state

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \overset{\text{Discount}}{\gamma^t} \underset{\text{Reward}}{R}(\overset{\text{State}}{S_t})\right]$$

$\overset{0.72}{\boxed{(1,1)_{-0.04}}}, \overset{0.76}{\boxed{(1,2)_{-0.04}}}, \overset{0.80}{\boxed{(1,3)_{-0.04}}}, \overset{0.84}{\boxed{(1,2)_{-0.04}}}, \overset{0.88}{\boxed{(1,3)_{-0.04}}}, (2,3)_{-0.04}, (3,3)_{-0.04}, (4,3)_{+1}$

$\overset{0.72}{\boxed{(1,1)_{-0.04}}}, (1,2)_{-0.04}, (1,3)_{-0.04}, (2,3)_{-0.04}, (3,3)_{-0.04}, (3,2)_{-0.04}, (3,3)_{-0.04}, (4,3)_{+1}$

$\overset{-1.16}{\boxed{(1,1)_{-0.04}}}, (2,1)_{-0.04}, \overset{-1.08}{\boxed{(3,1)_{-0.04}}}, (3,2)_{-0.04}, (4,2)_{-1}$

Note that for this example

- Value of each state can only be computed after reaching terminal state

# Difference to the Bellman Equation

- This no longer uses the Bellman recursion – consequence is that we do not learn the value of a particular state given the utility of its neighbors

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} \underbrace{P\left(s' \middle| s, \pi(s)\right)}_{\text{We no longer have this}} U^{\pi}(s')$$

$(1,1)_{-0.04}$, $(1,2)_{-0.04}$, $(1,3)_{-0.04}$, $(1,2)_{-0.04}$, $(1,3)_{-0.04}$, $\overset{0.96}{\boxed{(2,3)_{-0.04}}}$, $(3,3)_{-0.04}$, $(4,3)_{+1}$

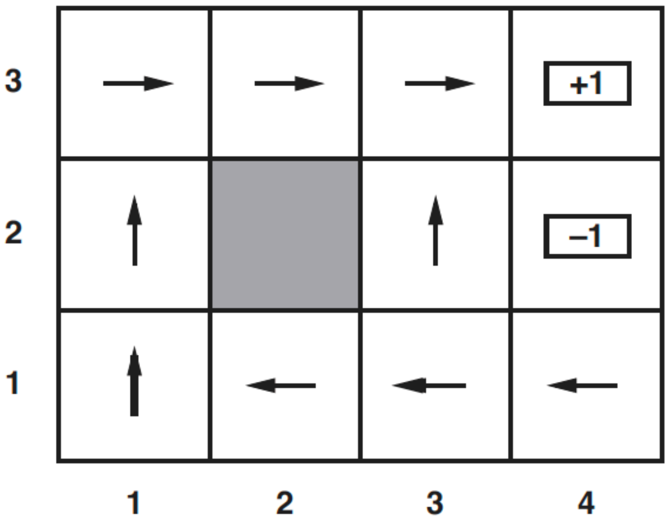
$(1,1)_{-0.04}$, $(1,2)_{-0.04}$, $(1,3)_{-0.04}$, $(2,3)_{-0.04}$, $\boxed{(3,3)_{-0.04}}$, $\overset{0.96}{\boxed{(3,2)_{-0.04}}}$, $(3,3)_{-0.04}$, $(4,3)_{+1}$

➢Consequence: Direct utility estimation has slow convergence

# Adaptive Dynamic Programming

- Idea: what if we learn the transition model? Then perhaps we can take advantage of a Bellman-like calculation

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




$(1,1)_{-0.04}$, $(1,2)_{-0.04}$, $(1,3)_{-0.04}$, $(1,2)_{-0.04}$, $(1,3)_{-0.04}$, $(2,3)_{-0.04}$, $(3,3)_{-0.}$ $(4,3)_{+1}$

$(1,1)_{-0.04}$, $(1,2)_{-0.04}$, $(1,3)_{-0.04}$, $(2,3)_{-0.04}$, $(3,3)_{-0.04}$, $(3,2)_{-0.04}$, $(3,3)_{-0}$ $(4,3)_{+1}$

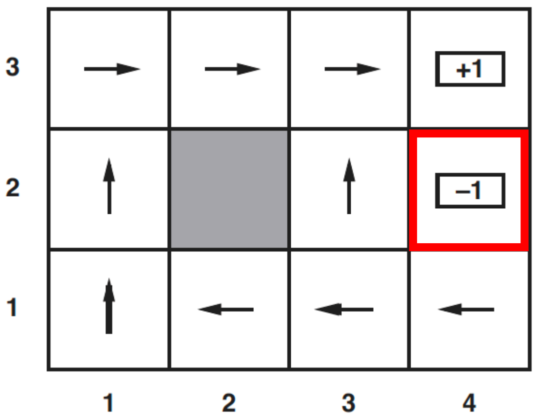
$(1,1)_{-0.04}$, $(2,1)_{-0.04}$, $(3,1)_{-0.04}$, $(3,2)_{-0.04}$, $(4,2)_{-1}$

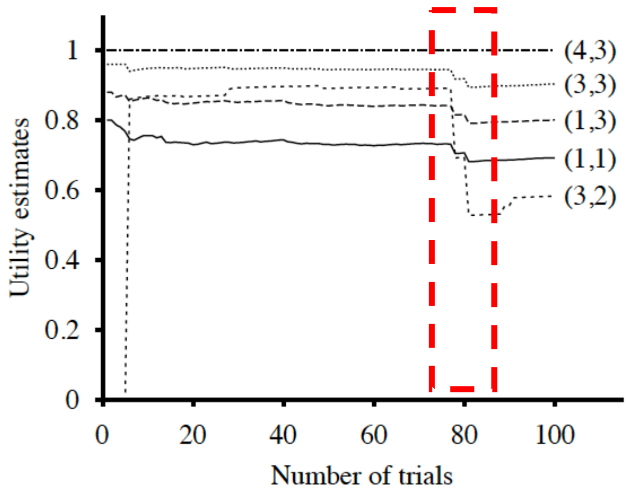
(1,3) → (1,2)

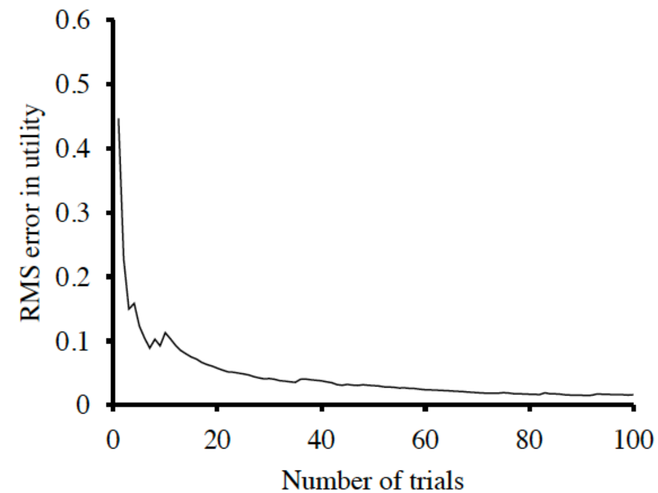
(1,3) → (2,3) ⇨ P((2,3)|(1,3),Right)=2/3

(1,3) → (2,3)

# Example Performance ADP




Russell and Norvig text ch 21

# Temporal Difference Learning

- Again, the goal here is to predict the utility function of each state

Actual return following time t

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha[\underbrace{G_t - U^\pi(s)}_{\text{Update}}]$$

Note: Sutton and Barto text uses slightly different notation for

- How to find ?

# Temporal-Difference Learning

- Update the utility function on a per-observation basis

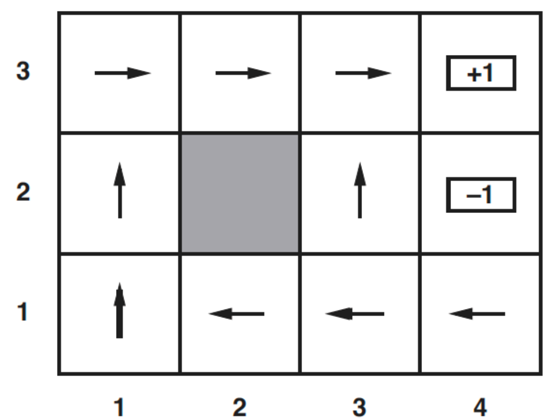
$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \underset{\substack{\uparrow \\ \text{Learning rate parameter}}}{\alpha}(R(s) + \gamma U^{\pi}(\underset{\substack{\uparrow \\ \text{Observed successor of s}}}{s'}) - U^{\pi}(s))$$

- If we change from a fixed parameter to a function that decreases as the number of times a state has been visited increases, then will converge to the correct value (conditions given on p725 of Rusell and Norvig)

# Example: TD for Gridworld

**Tabular TD(0) for estimating $v_\pi$**

Input: the policy $\pi$ to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
  Initialize $S$
  Repeat (for each step of episode):
    $A \leftarrow$ action given by $\pi$ for $S$
    Take action $A$, observe $R$, $S'$
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until $S$ is terminal



# How Does this Compare to What We've Seen Before?

- Recall policy iteration:

**Step 1:** Evaluation

**Step 2:** Improvement using one-step look-ahead

We are not doing control right now (remember the policy is *fixed*

# How Does this Compare to What We've Seen Before?

- Recall policy iteration:

Policy Iteration Evaluation

What we are doing now: TD(0)

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

What is missing?

Is this still sound? Will TD(0) converge?

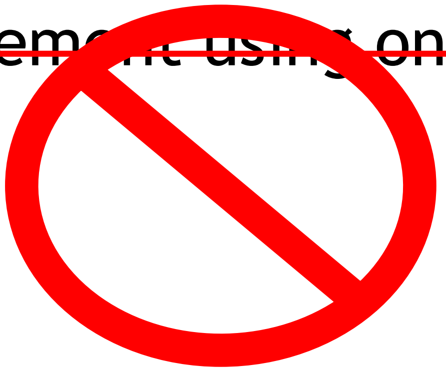
Yes for the right stepsize conditions on

# What About Learning a Policy?

- How to learn about best action to take, not just the value of a particular state
- Recall policy iteration:

**Step 1:** Evaluation

**Step 2:** Improvement using one-step look-ahead



# Active Reinforcement Learning

- Remember how we used to choose the best action:

- Now we don't have a model… Can we still learn the best action?

- Idea: estimate the Q-value

- This is called *State-Action-Reward-State-Action* (SARSA)

# Q-Learning

- Off-policy learning (tries to estimate the optimal policy, independent of current policy)

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a} Q(s',a') - Q(s,a))$$

**Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$**

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize $S$
  Repeat (for each step of episode):
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Take action $A$, observe $R$, $S'$
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until $S$ is terminal

Sutton and Barto text p107

# Summary

- Difference between MDP and now…?
- Learning state utilities vs policies (passive versus active)
- Still converge to true state values and optimal policies so long as you have enough trials (samples) and learning parameters like satisfy conditions for convergence