Reinforcement Learning

CS 486/686: Introduction to Artificial Intelligence

Outline

- What is reinforcement learning
- Quick MDP review
- Passive learning
  - Temporal Difference Learning
- Active learning
  - Q-Learning

What is RL?

- Reinforcement learning is learning what to do so as to maximize a numerical reward signal
- Learner is not told what actions to take
- Learner discovers value of actions by
  - Trying actions out
  - Seeing what the reward is

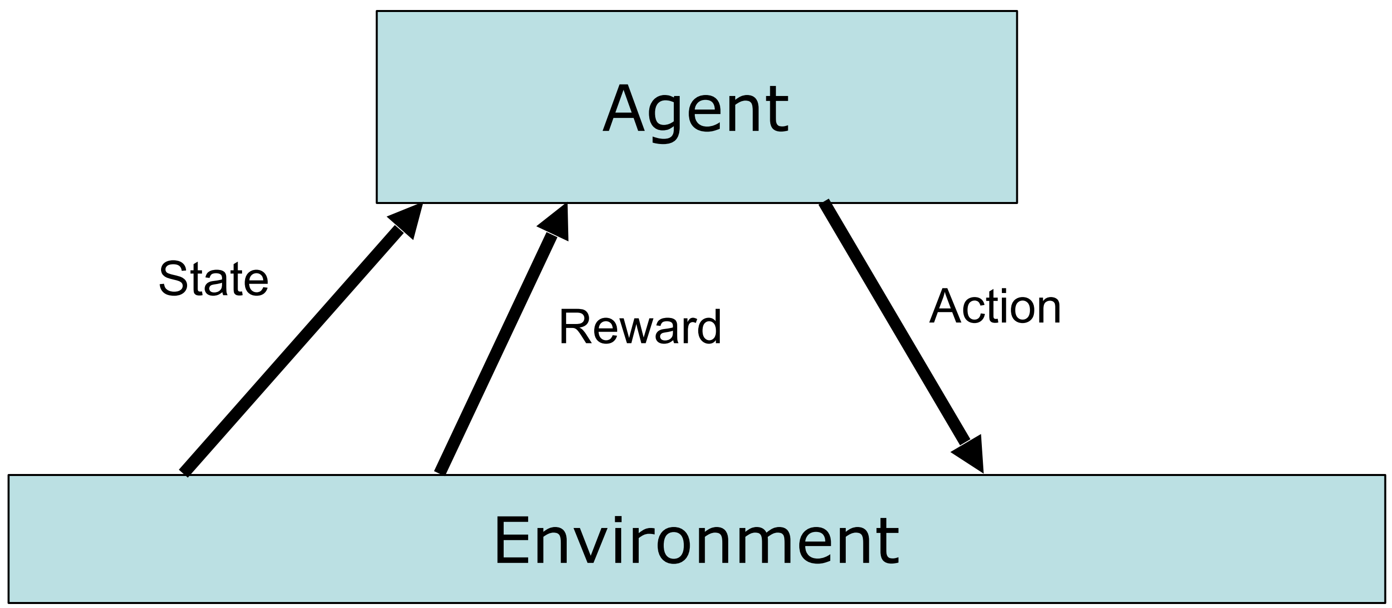
What is RL?

- Another common learning framework is supervised learning (we will see this later in the semester)

Supervised learning

Reinforcement learning

Reinforcement Learning Problem



$$s0 \xrightarrow[r0]{a0} s1 \xrightarrow[r1]{a1} s2 \xrightarrow[r2]{a2}$$

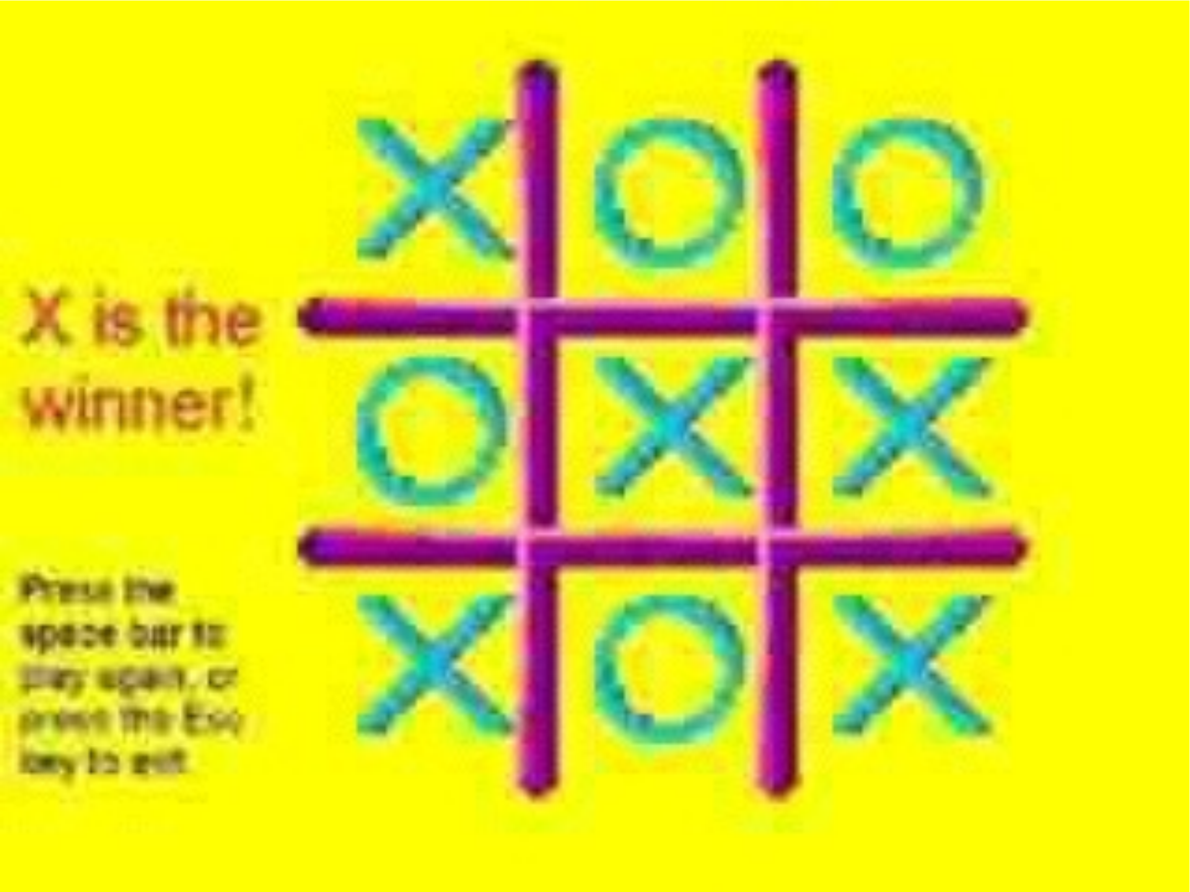
Goal: Learn to choose actions that maximize $r_0+\gamma\ r_1+\gamma^2 r_2+\ldots$, where $0\cdot<\gamma<1$

Example: Slot Machine

- **State**: Configuration of slots
- **Actions**: Stopping time
- **Reward**: $$$
- **Problem**: Find π: S →A that maximizes the reward

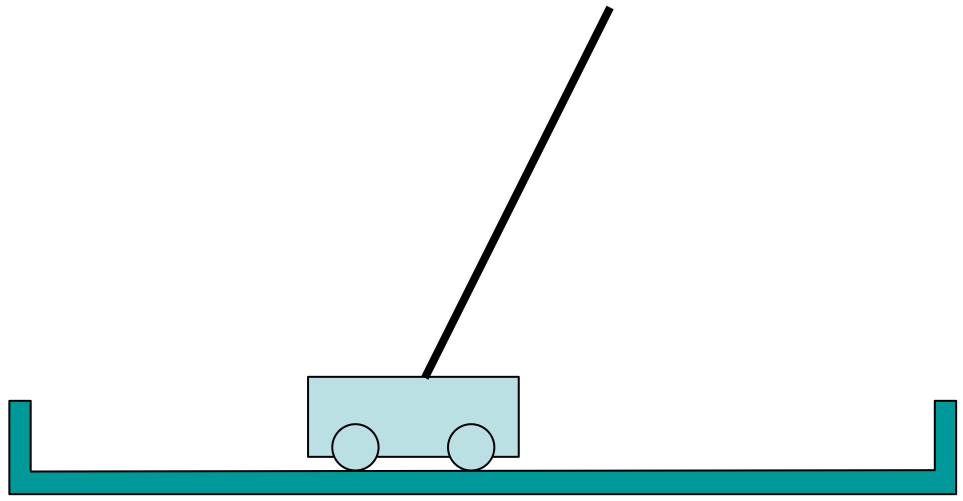
Example: Tic Tac Toe

- **State**: Board configuration
- **Actions**: Next move
- **Reward**: 1 for a win, -1 for a loss, 0 for a draw
- **Problem**: Find π: S →A that maximizes the reward



Example: Inverted Pendulem

- **State**: x(t), x'(t), θ(t), θ'(t)
- **Actions**: Force F
- **Reward**: 1 for any step where the pole is balanced
- **Problem**: Find π: S → A that maximizes the reward



Example: Mobile Robot

- **State**: Location of robot, people
- **Actions**: Motion
- **Reward**: Number of happy faces
- **Problem**: Find π: S → A that maximizes the reward

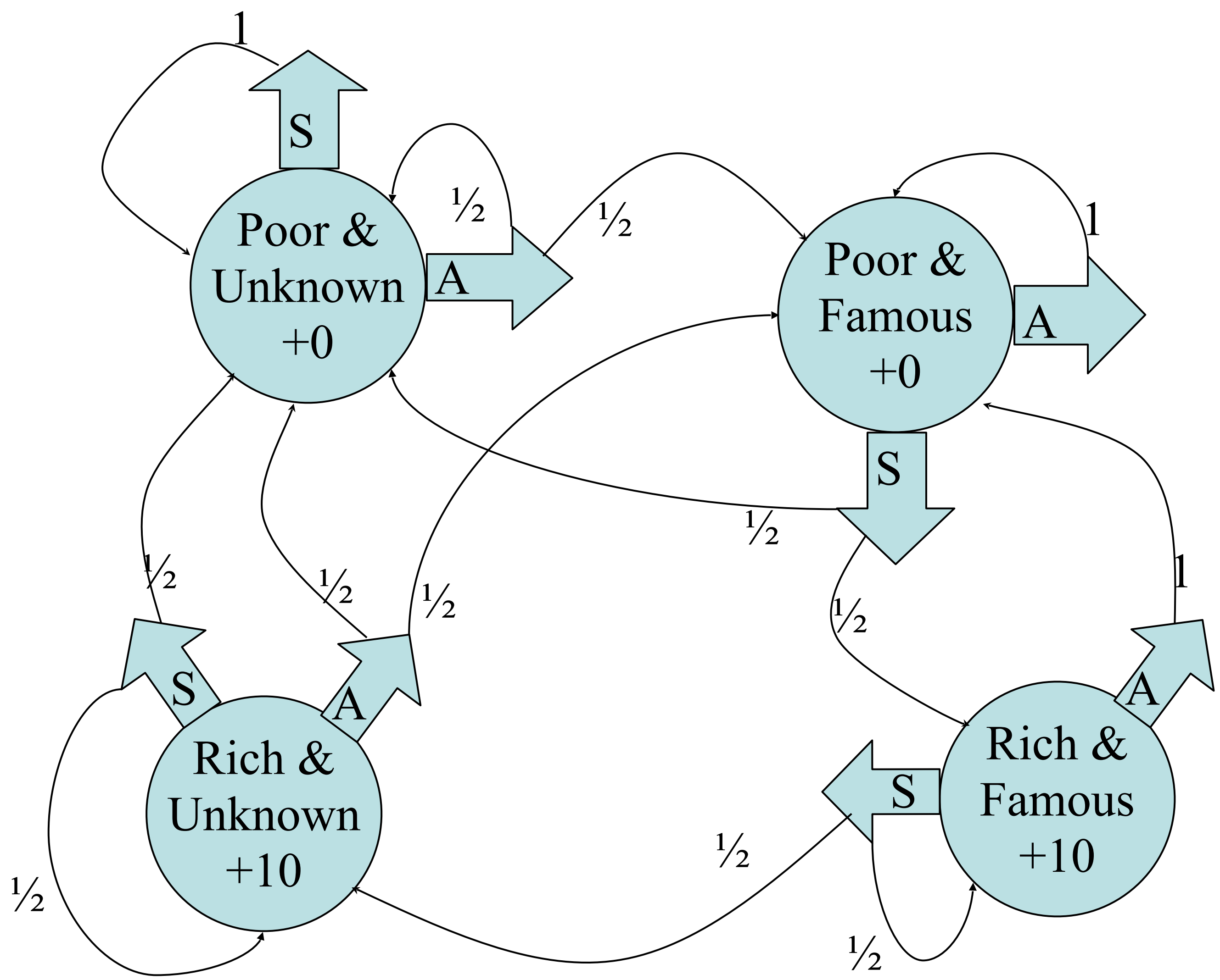
Reinforcement Learning Characteristics

- Delayed reward
  - Credit assignment problem
- Exploration and exploitation
- Possibility that a state is only partially observable
- Life-long learning

Reinforcement Learning Model

- Set of states S
- Set of actions A
- Set of reinforcement signals (rewards)
  - Rewards may be delayed

Markov Decision Process



$\gamma = 0.9$

You own a company

In every state you must choose between **S**aving money or **A**dvertising

Markov Decision Process

- Set of states $\{s_1, s_2, \ldots s_n\}$
- Set of actions $\{a_1, \ldots, a_m\}$
- Each state has a reward $\{r_1, r_2, \ldots r_n\}$
- Transition probability function

$$P_{ij}^k = (\text{Next}=s_j | \text{ This}=s_i \text{ and I take action } a_k)$$

- ON EACH STEP…
  0. Assume your state is s_i
  1. You get given reward r_i
  2. Choose action a_k
  3. You will move to state s_j with probability P_{ij}^k
  4. All future rewards are discounted by γ

MDPs and RL

- With an MDP our goal was to **find the optimal policy given the model**
  - Given rewards and transition probabilities
- In RL our goal is to **find the optimal policy but we start without knowing the model**
  - Not given rewards and transition probabilities

Agent's Learning Task

- Execute actions in the world
- Observe the results
- Learn policy $\pi: S \rightarrow A$ that maximizes $E[r_t + \Upsilon r_{t+1} + \Upsilon^2 r_{t+2} + \ldots]$ from any starting state in S

Types of RL

- **Model-based vs Model-free**
  - **Model-based**: Learn the model and use it to determine the optimal policy
  - **Model-free**: Derive optimal optimal policy without learning the model
- **Passive vs Active**
  - **Passive**: Agent observes the world and tries to determine the value of being in different states
  - **Active**: Agent watches and takes actions

Passive Learning

- An agent has a policy π
- Executes a set of trials using π
  - Starts in s_0, has a series of state transitions until it reaches the terminal state
- Tries to determine the expected utility of being in each state

Passive Learning

3	r	r	r	+1
2	u		u	-1
1	u	l	l	l
	1	2	3	4

$\gamma = 1$

r_i = -0.04 for non-terminal states

We do not know the transition probabilities

$(1,1)\rightarrow(1,2)\rightarrow(1,3)\rightarrow(1,2)\rightarrow(1,3)!\rightarrow(2,3)\rightarrow(3,3)\rightarrow(4,3)_{+1}$

$(1,1)\rightarrow(1,2)\rightarrow(1,3)\rightarrow(2,3)\rightarrow(3,3)\rightarrow(3,2)\rightarrow(3,3)\rightarrow(4,3)_{+1}$

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

What is the value, $V^*(s)$ of being in state s?

Direct Utility Estimation

- Direct utility estimation is a form of supervised learning
  - Input: State
  - Output: Reward
- Ignore an important piece of information
  - Utility values obey Bellman equation
- Misses opportunities for learning

Adaptive Dynamic Programming (ADP)

- Recall Bellman equations:
  - $V^{\pi}(s_i)=r_i+\gamma\Sigma_j P_{ij}^{\pi(si)} V^{\pi}(s_j)$
    - Connection between states can speed up learning
    - Do not need to consider any situation where the above constraint is violated
- Adaptive dynamic programming (ADP)
  - Learns transition probabilities, rewards from observations
  - Updates values of states

Example: ADP

$\gamma = 1$

3	**r**	r	r	+1
2	u		u	-1
1	u	l	l	l
	1	2	3	4

r_i = -0.04 for non-terminal states

$$V^{\pi}(s_i) = r(s_i) + \gamma \sum_j P^{\pi}_{ij} V^{\pi}(s_j)$$

(1,1)→(1,2)→ **(1,3)** → (1,2)→**(1,3)**→ (2,3)→(3,3)→$(4,3)_{+1}$

(1,1)→(1,2)→**(1,3)**→(2,3)→(3,3)→(3,2)→(3,3)→$(4,3)_{+1}$

(1,1)→(2,1)→(3,1)→(3,2)→$(4,2)_{-1}$

$$\left.\begin{array}{l} P_{(1,3)(2,3)}^{r}=2/3 \\ P_{(1,3)(1,2)}^{r}=1/3 \end{array}\right\}$$ Use this information in the Bellman equation

Temporal Difference

- Model free
- Key Idea:
  - Use observed transitions to adjust values of observed states so that they satisfy Bellman equations
  - At each time step
    - Observe s, a, s', r
    - Update V^{π} after each move
    - $V^{\pi}(s)=V^{\pi}(s)+\alpha(r(s)+\gamma V^{\pi}(s')-V^{\pi}(s))$

TD(0)

$$\underbrace{V^\pi(s) = V^\pi(s) + \alpha}_{}(\underbrace{r(s) + \gamma V^\pi(s') - V^\pi(s)}_{\text{Temporal difference}})$$

Learning rate

- **Theorem**: If α is appropriately decreased with the number of times a state is visited, then Vπ(s) converges to the correct value.
  - α must satisfy
    - $\Sigma_n \alpha(n) \rightarrow 1$
    - $\Sigma_n \alpha^2(n) < 1$

TD-Lambda

- Idea: Update from the whole training sequence, not just a single state transition

$$V^\pi(s_i) \rightarrow V^\pi(s_i) + \alpha \sum_{m=i}^{\infty} \lambda^{m-i}[r(s_m) + \gamma V^\pi(s_{m+1}) - V^\pi(s_m)]$$

- Special cases:
  - Lambda = 1 (basically ADP)
  - Lambda=0 (TD)
- Intermediate choice of lambda is best (empirically lambda=0.7 works well)

Active Learning

- Recall, that real goal is to find a good policy
  - If the transition and reward model is known then
    - $V^*(s)=\max_a[r(s)+\gamma\Sigma_{s'}P(s'|s,a)V^*(s')]$
  - If the transition and reward model is unknown
    - Improve policy as agent executes it

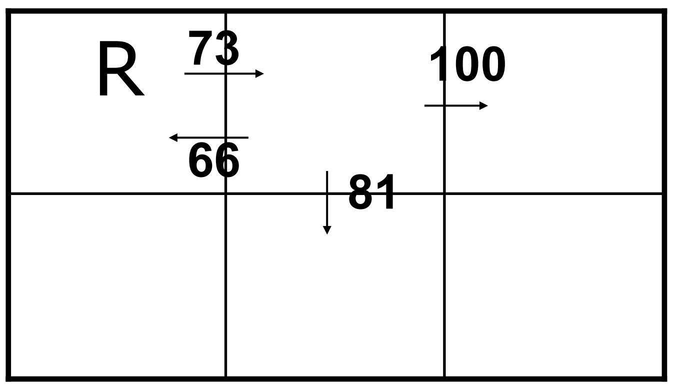
Q-Learning

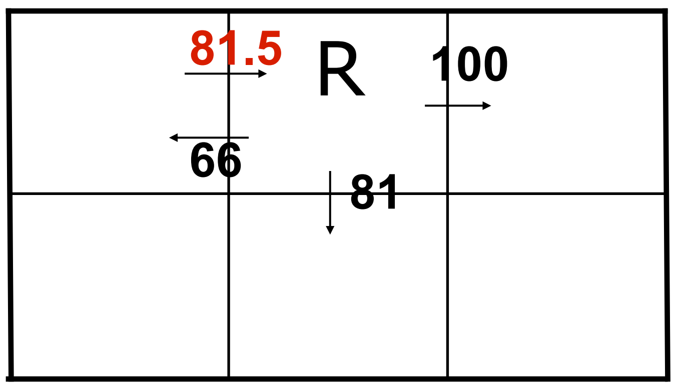
- Key idea: Learn a function Q:SxA->**R**
  - Value of a state-action pair
  - Policy $\pi(s)=\operatorname{argmax}_a Q(s,a)$ is the optimal policy
  - $V^*(s)=\max_a Q(s,a)$
- Bellman's equation
  - $Q(s,a)=r(s)+\gamma\Sigma_{s'}P(s'|s,a)\max_{a'}Q(s',a')$

Q-Learning

- For each state s and action a, initialize Q(s,a)
  - Q(s,a)=0 or some random value
- Observe current state
- **Loop**
  - Select action a and execute it
  - Receive immediate reward r
  - Observe new state s'
  - Update Q(s,a)
    - $Q(s,a)=Q(s,a)+\alpha(r+\gamma \max_{a'} Q(s',a')-Q(s,a))$
  - s=s'

Example: Q-Learning





r=0 for non-terminal states
γ=0.9
α = 0.5

$$\begin{aligned} Q(s_1, a_{\text{right}}) &= Q(s_1, a_{\text{right}}) + \alpha(r + \gamma \max_{a'} Q(s_2, a') - Q(s_1, a_{\text{right}})) \\ &= 73 + 0.5(0 + 0.9\max[66, 81, 100] - 73) \\ &= 73 + 0.5(17) \\ &= 81.5 \end{aligned}$$

Q-Learning

- For each state s and action a, initialize Q(s,a)
  - Q(s,a)=0 or some random value
- Observe current state
- **Loop**
  - **Select action a and execute it**
  - Receive immediate reward r
  - Observe new state s'
  - Update Q(s,a)
    - $Q(s,a)=Q(s,a)+\alpha(r+\gamma \max_{a'} Q(s',a')-Q(s,a))$
  - s=s'

Exploration vs Exploitation

- If an agent always choses the action with highest value then it is **exploiting**
- If an agent always choses an action at random then it may learn the model (**exploring**)
- Need to balance the two

Common Exploration Methods

- Use an optimistic estimate of utility
- Chose best action with probability p and a random action otherwise
- Boltzmann exploration

$$P(a) = \frac{e^{Q(s,a)/T}}{\sum_a e^{Q(s,a)/T}}$$

Exploration and Q-Learning

- Q-Learning converges to the optimal Q-values if
  - Every state is visited infinitely often (due to exploration)
  - The action selection becomes greedy as time approaches infinity
  - The learning rate is decreased appropriately

Summary

- Active vs Passive Learning
- Model-Based vs Model-Free
- ADP
- TD
- Q-learning
  - Exploration-Exploitation tradeoff