Robot Learning

RL

Prof. Songhwai Oh
ECE, SNU

AIMA Ch. 23

Reinforcement Learning

Reinforcement Learning

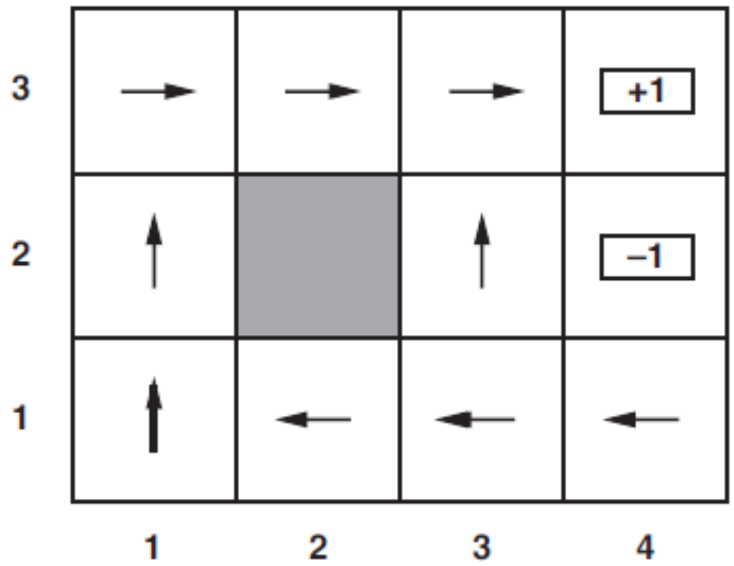
- **Markov decision processes (MDPs)**
  - Complete model is known
- **Reinforcement learning**
  - Use observed rewards to learn an optimal policy for the environment.
  - Complete model is not known.
- Three agent types:
  - **Behavior cloning**: learns a policy that maps directly from states to actions.
  - **Utility-based**: an agent learns a utility function (value function) on states and uses it to select actions that maximize the expected outcome utility.
  - **Q-learning**: an agent learns an action-utility function (**Q-function** or action-value function) giving the expected utility of taking a given action in a given state.

Passive Reinforcement Learning

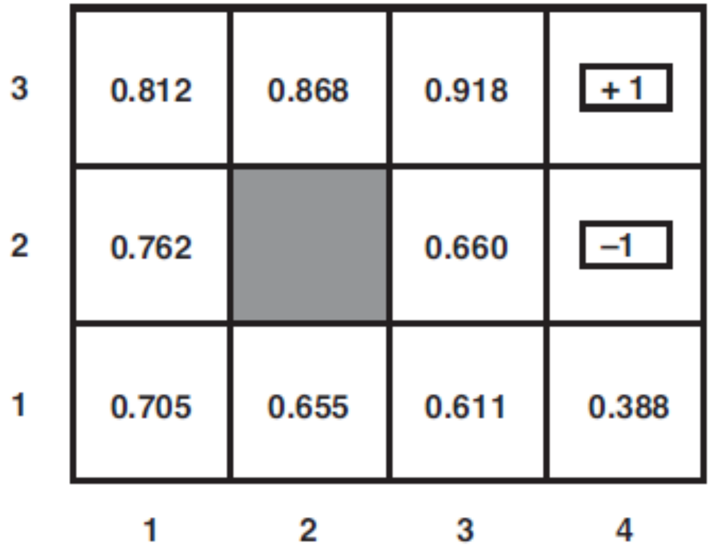
- Fully observable environment
- Agent's policy π is fixed (i.e., the agent always executes π(s) at s).
- Goal: to learn how good the policy is, i.e., to learn the utility function $U^{\pi}(s)$.
- Similar to policy evaluation (a step in policy iteration)
  - Differences: Transition model and reward function are **not** known

- Agent executes a set of trials using π.
- Based on its percepts, collect the current state and the reward at that state.
- Goal: Based on sample trials, learn the expected utility $U^{\pi}(s)$.

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$



Policy π



Utilities (U^{π})

Example of trials

$$(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$$
$$(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (3,2)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$$
$$(1,1)_{-.04} \rightsquigarrow (2,1)_{-.04} \rightsquigarrow (3,1)_{-.04} \rightsquigarrow (3,2)_{-.04} \rightsquigarrow (4,2)_{-1} \; .$$

Direct Utility Estimation

- **Utility of a state** (**value of a state** or **reward-to-go**): the expected total reward from the state onward.

$$(1,1)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (2,3)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (4,3)_{\textbf{+1}}$$

  - (1,1): one sample with a total reward of 0.72
  - (1,2): two samples of total rewards of 0.76 and 0.84
  - (1,3): two samples of total rewards of 0.80 and 0.88
  - ……

- Use a supervised learning algorithm to estimate the utility function.

- But direct utility estimation requires a much larger number of samples than necessary since it ignores the structure of the problem, namely the Bellman equation (utilities of states are not independent).

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

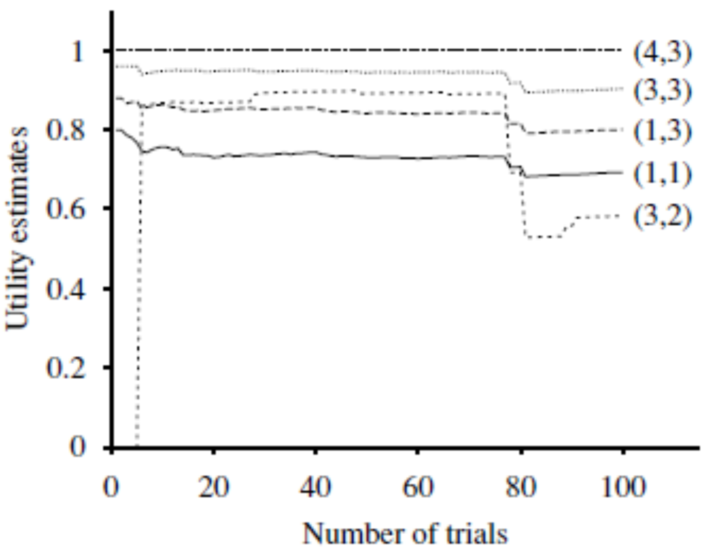
Adaptive Dynamic Programming (ADP)

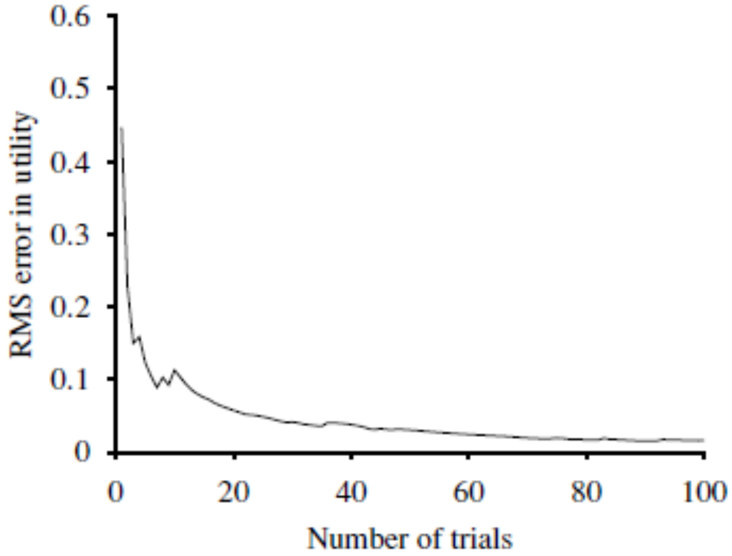
- Learns the transition model from samples (ML estimation)

function PASSIVE-ADP-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
    mdp, an MDP with model P, rewards R, discount γ
    U, a table of utilities, initially empty
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    $N_{s'|sa}$, a table of outcome frequencies given state–action pairs, initially zero
    s, a, the previous state and action, initially null

  if s' is new **then** $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$ and $N_{s'|sa}[s', s, a]$
    for each t such that $N_{s'|sa}[t, s, a]$ is nonzero **do**
      $P(t \mid s, a) \leftarrow N_{s'|sa}[t, s, a] \,/\, N_{sa}[s, a]$
  $U \leftarrow$ POLICY-EVALUATION(π, U, *mdp*)
  if s'.TERMINAL? **then** $s, a \leftarrow$ null **else** $s, a \leftarrow s', \pi[s']$
  return a

Passive ADP Learning





- Issues with ADP
  - Intractable for problems with large state spaces
  - Problems with maximum-likelihood (ML) estimation
    - Methods to avoid this issue:
      - Bayesian reinforcement learning $\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_h P(h \mid \mathbf{e}) u_h^\pi$
      - Robust control theory $\pi^* = \underset{\pi}{\operatorname{argmax}} \min_h u_h^\pi$

Temporal Difference (TD) Learning

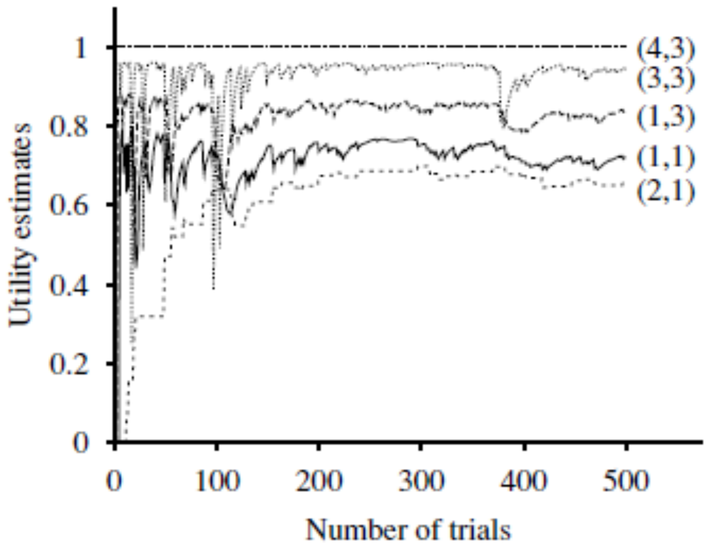
- Temporal difference (TD) equation $U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$
- TD does not need the transition model (cf. ADP)

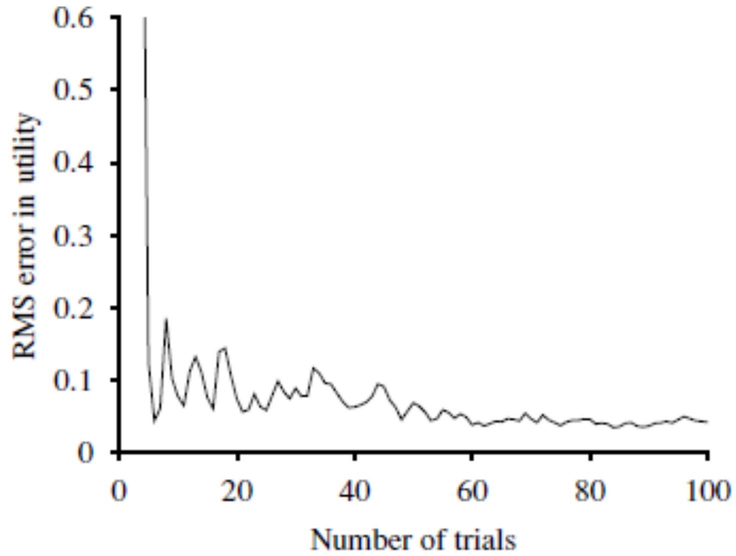
function PASSIVE-TD-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
    U, a table of utilities, initially empty
    N_s, a table of frequencies for states, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if s' is new **then** $U[s'] \leftarrow r'$
  if s is not null **then**
    increment $N_s[s]$
    $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$
  if s'.TERMINAL? **then** $s, a, r \leftarrow$ null **else** $s, a, r \leftarrow s', \pi[s'], r'$
  return a

- The algorithm adjusts the utility estimates towards the ideal equilibrium that holds locally when the utility estimates are correct.
- For a correct choice of α(n), it converges to the correct value.

Passive TD Learning

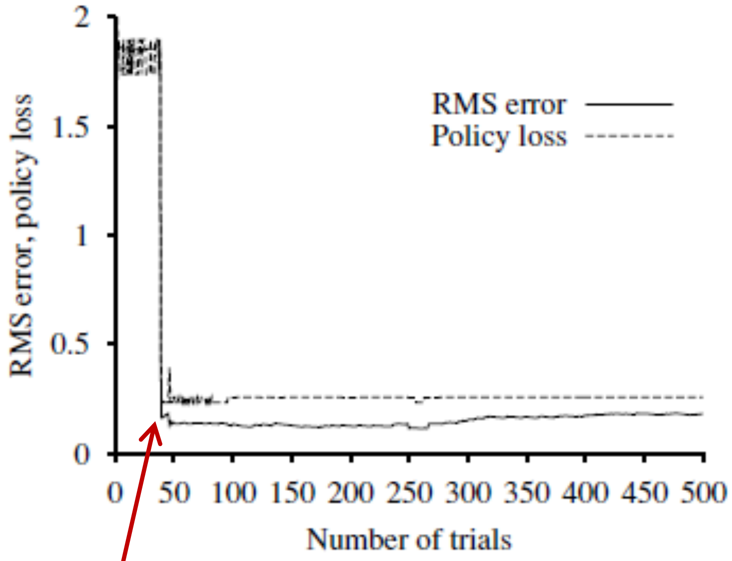




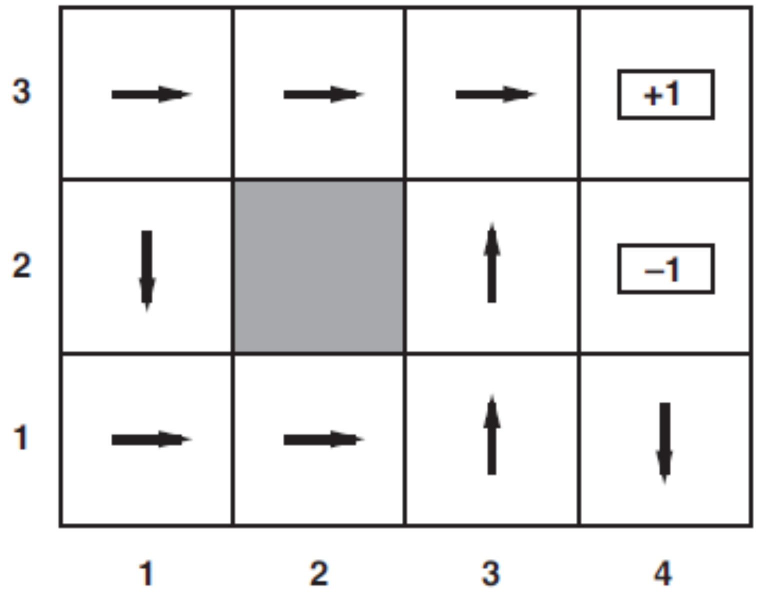
- Compared to ADP
  - Cons: TD learns slowly and shows much variability.
  - Pros: TD is simpler and requires less computation per observation.

Active Reinforcement Learning

Performance of a greedy ADP agent that executes the action recommended by the optimal policy for the learned model (one-step look-ahead).



Finds a policy that reaches (4,3)
via (2,1), (3,1), (3,2), (3,3)



Suboptimal policy

- **Exploration-exploitation tradeoff**: tradeoff between **exploitation** to maximize its reward (as reflected in its current utility estimates) and **exploration** to maximize its long-term well-being

Exploration Function

- $U^{+}(s)$: optimistic estimate of the utility at state *s*
- $N(s,a)$: number of times action *a* has been tried in state *s*
- Update equation (value iteration):

$$U^{+}(s) \leftarrow R(s) + \gamma \max_{a} f\left(\sum_{s'} P(s' \mid s,a) U^{+}(s'),\ N(s,a)\right)$$

- $f(u,n)$: **exploration function**, determining how greed (exploitation) is traded off against curiosity (exploration).
  - This function is increasing in *u* and decreasing in *n*
  - Example:

$$f(u,n) = \begin{cases} R^{+} & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

R^{+}: optimistic estimate of the best possible reward from any state.

Action-Utility Function

- *Q*(*s*,*a*): the value of doing action *a* in state *s*.

$$U(s) = \max_a Q(s,a)$$

- **Q-learning** is a model-free method: TD agent that learns a **Q-function** does not need a model of the form *P*(*s'*|*s*,*a*) for learning or for action selection.
- At equilibrium when Q-values are correct:

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a')$$

cf. Bellman equation: $U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s,a) U(s')$

- ADP Q-learning: (1) estimate the transition model; (2) update Q-values.
- TD Q-learning (does not require the transition model)

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

Q-Learning

- Update rule for Q-learning:

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

function Q-LEARNING-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: Q, a table of action values indexed by state and action, initially zero
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if TERMINAL?(s) **then** $Q[s, None] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s,a]$
    $Q[s,a] \leftarrow Q[s,a] + \alpha(N_{sa}[s,a])(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$
  $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s',a'], N_{sa}[s',a']), r'$
  return a

SARSA

- SARSA (State-Action-Reward-State-Action)
- Update rule for SARSA

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

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

- For a greedy agent, SARSA is the same as Q-learning
- For an explorative agent, they are different
  - Q-learning (off-policy), SARSA (on-policy)

```
Initialize Q(s, a) arbitrarily
Repeat (for each episode):
    Initialize s
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action a, observe r, s'
        Choose a' from s' using policy derived from Q (e.g., ε-greedy)
        Q(s, a) ← Q(s, a) + α[r + γQ(s', a') − Q(s, a)]
        s ← s'; a ← a';
    until s is terminal
```

Function Approximation

- For a large or infinite state space, previous approaches cannot be applied.
- **Function approximation**: approximates the utility or Q-function using a finite number of basis functions

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \cdots + \theta_n f_n(s)$$

- We can reduce the number of values we have to consider (compression).
- The compression achieved by a function approximator, which allows the learning agent to **generalize** from states it has visited to states it has not visited.
- Delta rule: $\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j(s)}{\partial \theta_i} = \theta_i + \alpha\,(u_j(s) - \hat{U}_\theta(s)) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$

$$E_j(s) = (\hat{U}_\theta(s) - u_j(s))^2/2$$

$u_j(s)$ is the observed total reward from state s onward in the jth trial

- Delta rule for a linear function approximator:

$$\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y$$

$$\begin{aligned}
\theta_0 &\leftarrow \theta_0 + \alpha\,(u_j(s) - \hat{U}_\theta(s)) \,, \\
\theta_1 &\leftarrow \theta_1 + \alpha\,(u_j(s) - \hat{U}_\theta(s))x \,, \\
\theta_2 &\leftarrow \theta_2 + \alpha\,(u_j(s) - \hat{U}_\theta(s))y \,.
\end{aligned}$$

- TD-learning with function approximation

$$\theta_i \leftarrow \theta_i + \alpha \left[R(s) + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)\right] \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

- Q-learning with function approximation

$$\theta_i \leftarrow \theta_i + \alpha \left[R(s) + \gamma \max_{a'} \hat{Q}_\theta(s', a') - \hat{Q}_\theta(s, a)\right] \frac{\partial \hat{Q}_\theta(s, a)}{\partial \theta_i}$$

- Changing the parameters θ in response to an observed transition between two states also changes the values of utilities for every other state
  - Reinforcement learner generalizes from its experiences
  - But there is the problem that there could be no function in the hypothesis space that approximates the true utility function sufficiently well.

Policy Search

- Search for a good policy directly in the policy space.
- Parameterize a policy
  - Example: $\pi(s) = \max_a \hat{Q}_\theta(s, a)$
  - Q-learning with function approximation: finds θ such that the approximated Q-value is close to the optimal Q-values
  - Policy search: finds θ that results in good performance
- Stochastic policy representation (to avoid discontinuities)

$$\pi_\theta(s, a) = e^{\hat{Q}_\theta(s,a)} / \sum_{a'} e^{\hat{Q}_\theta(s,a')}$$ (softmax function)

- Policy improvement
  - $\rho(\theta)$, **policy value**: the expected reward-to-go when π_θ is executed.
    - Policy gradient (if $\rho(\theta)$ is differentiable)
    - Empirical gradient (hill climbing)
  - REINFORCE $$\nabla_\theta \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^{N} \frac{(\nabla_\theta \pi_\theta(s, a_j)) R_j(s)}{\pi_\theta(s, a_j)}$$