CSC520 - Artificial Intelligence

Lecture 19

Dr. Scott N. Gerard

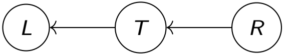
North Carolina State University

Mar 25, 2025

Class Exercise

Use Variable Elimination

- Random variables
  - R: Raining
  - T: Traffic
  - L: Late for class



Compute $P(L) = \alpha \sum_l \sum_r P(L, T, R) = \alpha \sum_l P(L|T) \sum_r P(T|R)P(R)$.

T	$P(T)$
$+t$	0.17
$-t$	0.83

$\sum_r$

T	R	$P(T, R)$
$+t$	$+r$	0.08
$-t$	$+r$	0.02
$+t$	$-r$	0.09
$-t$	$-r$	0.81

T	R	$P(T\|R)$
$+t$	$+r$	0.8
$-t$	$+r$	0.2
$+t$	$-r$	0.1
$-t$	$-r$	0.9

R	$P(R)$
$+r$	0.1
$-r$	0.9

L	$P(L)$
$+l$	0.202
$-l$	0.798

$\sum_t$

L	T	$P(L, T)$
$+l$	$+t$	0.119
$-l$	$+t$	0.051
$+l$	$-t$	0.083
$-l$	$-t$	0.747

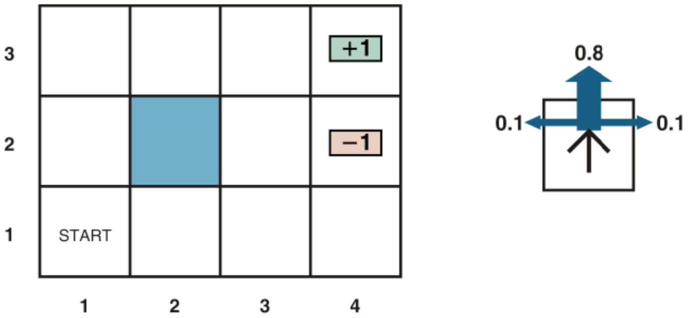
L	T	$P(L\|T)$
$+l$	$+t$	0.7
$-l$	$+t$	0.3
$+l$	$-t$	0.1
$-l$	$-t$	0.9

T	$P(T)$
$+t$	0.17
$-t$	0.83

Agenda

- Sequential Decision Problems
- Markov Decision Processes
- Policies and discounting
- Value iteration
- Reinforcement learning
- Model-based, Temporal difference and Q-learning

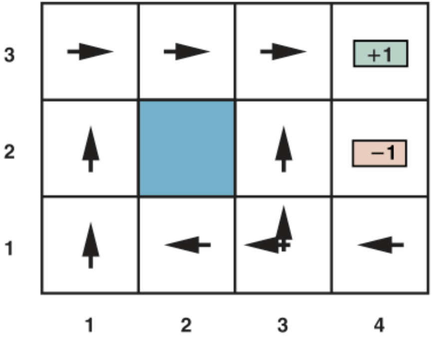
Sequential Decision Problems
Grid World



(a) (b)

- Big rewards (+1 or -1) at terminal states
- Small living reward for each step (e.g., -0.04)
- Objective: Maximize cumulative rewards

Sequential Decision Problems



- Optimal policy with r = -0.04

Markov Decision Process

- MDP models a fully observable and stochastic sequential decision problem
- MDP is defined by:
  - Set of states $s \in S$
  - Set of actions $a \in A$
  - Transition function $P(s'|s, a) = T(s, a, s')$
  - Reward function $R(s, a, s')$
- Transitions follow Markov property
  - $P(s_{t+1}|s_1, a_1, s_2, a_2, \ldots s_t, a_t) = P(s_{t+1}|s_t, a_t)$

MDP Policies

- A policy maps a state to an action: $\pi : S \rightarrow A$
- An optimal policy maximizes the expected utility: $\pi^*(s)$
  - For each state, an optimal policy gives the action that maximizes the expected utility
- Goal is to compute an optimal policy

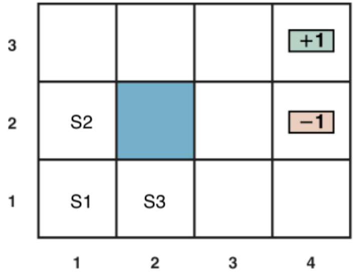
Discounting

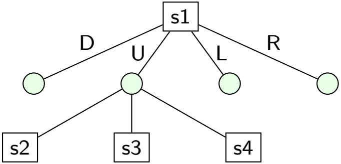
- Generally, earlier rewards are preferred than later rewards
  - For e.g., would you prefer $1M today or after 5 years?
- Future rewards are discounted using a discounting factor γ
  - $0 \leq \gamma \leq 1$
  - $U([s_0, a_0, s_1, a_1, \ldots]) = R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3) + \ldots$
- Prevents the issue of infinite rewards for infinite sequence of actions

Solving MDPs

- Solving MDP means finding an optimal policy: $\pi^*(s)$
- Value of a state: $V^*(s) =$ expected utility starting in state s and acting optimally
- Q-value of a state: $Q^*(s, a) =$ expected utility starting in state s and taking action a, and then acting optimally

Solving MDPs





$$
\begin{aligned}
Q(S1, U) = {} & T(s1, U, s1)[R(s1, U, s1) + \gamma V^*(s1)] \\
& + T(s1, U, s2)[R(s1, U, s2) + \gamma V^*(s2)] \\
& + T(s1, U, s3)[R(s1, U, s3) + \gamma V^*(s3)]
\end{aligned}
$$

$$Q^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

$$V^*(s) = \max_a Q^*(s, a)$$

- $V^*(s1)$ is the utility of the optimal path to the end.
- $Q^*(s1, a1)$ is the utility of taking action $a1$ and then acting optimally.

Solving MDPs

Bellman equation

$$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

$$V^*(s) = \max_a Q^*(s,a)$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a)$$

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

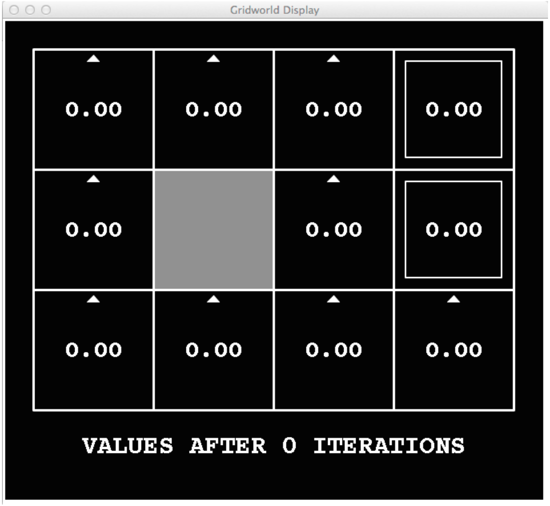
Value Iteration

- Iterative method for solving Bellman equations
- Initialize $V_0^*(s)$ with some initial values
  - Can be set given some prior knowledge
  - Otherwise, can be set to 0
- Update $V^*(s)$ using Bellman equation

$$V_{i+1}^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_i^*(s')]$$

- Repeat until convergence

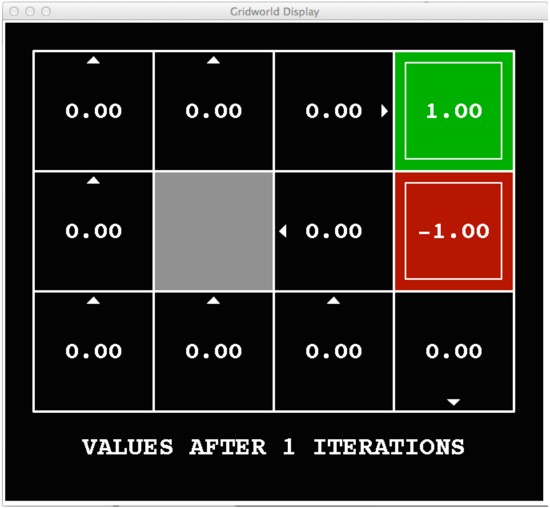
Value Iteration



Noise $= 0.2$, Discount $= 0.9$, Living reward $= 0$

Image credit: Dan Klein and Pieter Abbeel

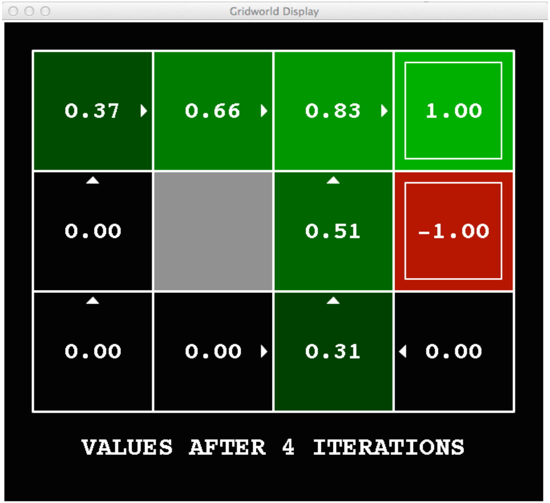
Value Iteration



Noise $= 0.2$, Discount $= 0.9$, Living reward $= 0$

Image credit: Dan Klein and Pieter Abbeel

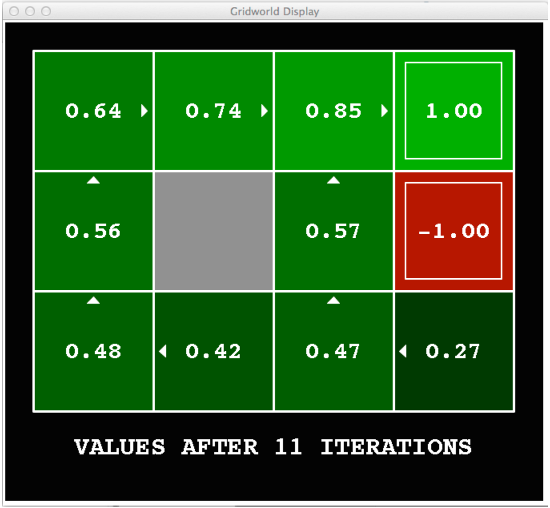
Value Iteration



Noise $= 0.2$, Discount $= 0.9$, Living reward $= 0$

Image credit: Dan Klein and Pieter Abbeel

Value Iteration



Noise = 0.2, Discount = 0.9, Living reward = 0

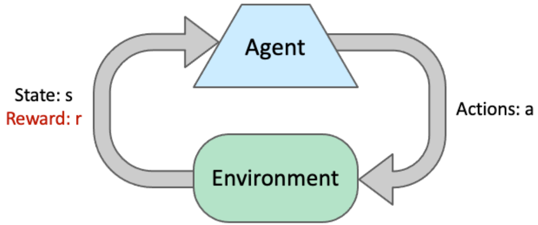
Image credit: Dan Klein and Pieter Abbeel

Value Iteration

Noise $= 0.2$, Discount $= 0.9$, Living reward $= 0$

Image credit: Dan Klein and Pieter Abbeel

Reinforcement Learning



- Agent performs an action on the environment
- As a result, agent receives a reward r from the environment and the state transitions to the next state
- Agent must learn to act to maximize expected rewards

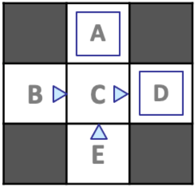
Reinforcement Learning

- Similar to solving MDPs
  - MDP involves offline solution; transition probabilities T and the reward function R are given
  - RL involves online learning; T and R must be learnt
- Model-based vs model-free RL
  - Model-based learning involves estimating T and R and then solving the MDP to obtain the policy
  - In model-free learning, agent does not estimate T or R
- Passive vs active RL
  - In passive RL, policy is fixed and agent learns values of states
  - In active RL, policy is not fixed and agent selects the actions to execute; goal is to learn optimal policy

Model-based Learning

- Learn approximate T and R from given experience
- Solve the MDP to obtain $\pi^*(s)$

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$\hat{T}(s, a, s')$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
...

$\hat{R}(s, a, s')$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
...

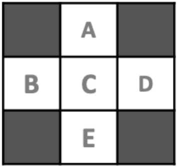
From: Dan Klein and Pieter Abbeel

Model-free Learning: Temporal Difference Learning

- Passive learning: policy π is fixed and objective is to learn $V^\pi(s)$
- Initialize $V^\pi(s)$ using prior knowledge or to 0
- Bellman: $V^\pi(s) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^\pi(s')]$
- Update $V^\pi(s)$ using each experience sample: (s, a, s', r)
- From a sample, calculate:
  $current = R(s, a, s') + \gamma V^\pi(s')$
- Then update $V^\pi(s)$ using:
  $V^\pi(s) \leftarrow V^\pi(s) + \alpha(current - V^\pi(s))$
- Decrease α over time for convergence

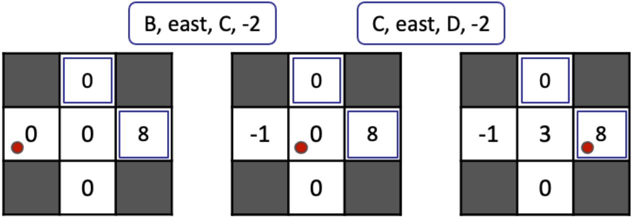
Temporal Difference Learning - Example

States



Assume: $\gamma = 1$, $\alpha = 1/2$

Observed Transitions



$$V^\pi(s) = V^\pi(s) + \alpha(R(s,a,s') + \gamma V^\pi(s') - V^\pi(s))$$

$$V^\pi(B) = 0 + \frac{1}{2}(-2 + 1 * 0 - 0)$$

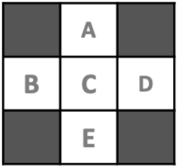
$$V^\pi(C) = 0 + \frac{1}{2}(-2 + 1 * 8 - 0)$$

Q-Learning

- Active learning: agent selects actions to execute
- Initialize $Q(s, a)$ using prior knowledge or to 0
- Agent selects an action to perform in state s
  - Exploitation: Select the optimal action $\underset{a}{\operatorname{argmax}}\, Q(s, a)$ with some probability $1 - \epsilon$
  - Exploration: Select a random action with probability ϵ
- Agent performs selected action and gets experience: (s, a, s', r)
- From a sample, calculate:
  $current = R(s, a, s') + \gamma \max_{a'} Q(s', a')$
- Then update $Q(s, a)$ using:
  $Q(s, a) \leftarrow Q(s, a) + \alpha(current - Q(s, a))$
- Decrease α over time for convergence
- It can be shown that, Q-learning converges to optimal policy

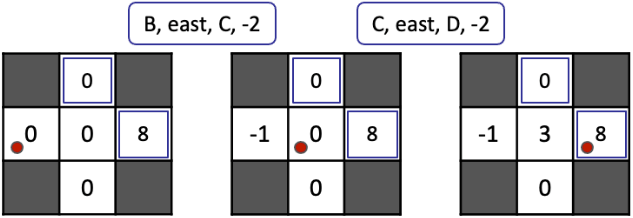
Class Exercise

States



Assume: $\gamma = 1$, $\alpha = 1/2$

Observed Transitions



$$V^{\pi}(s) = V^{\pi}(s) + \alpha(R(s,a,s') + \gamma V^{\pi}(s') - V^{\pi}(s))$$

- Calculate $V^{\pi}(E)$ if next observed transition is: (E, north, C, -2)