CMSC 671
Fall 2010

Tue 11/30/10

Reinforcement Learning
Chapter 21

Based on the lecture slides for textbook *Machine Learning* by Tom M. Mitchell

Prof. Laura Zavala, laura.zavala@umbc.edu, ITE 373, 410-455-8775

The Reinforcement Learning Problem

- How an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals?

Crawl before you can walk

- Perhaps our programming isn’t for crawling at all, but for the *desire for movement*!

Reinforcement Learning Hypothesis

- *Intelligent behavior arises from the actions of an individual seeking to maximize its received reward signals in a complex and changing world.*
- Research program:
  - identify where reward signals come from,
  - develop algorithms that search the space of behaviors to maximize reward signals.

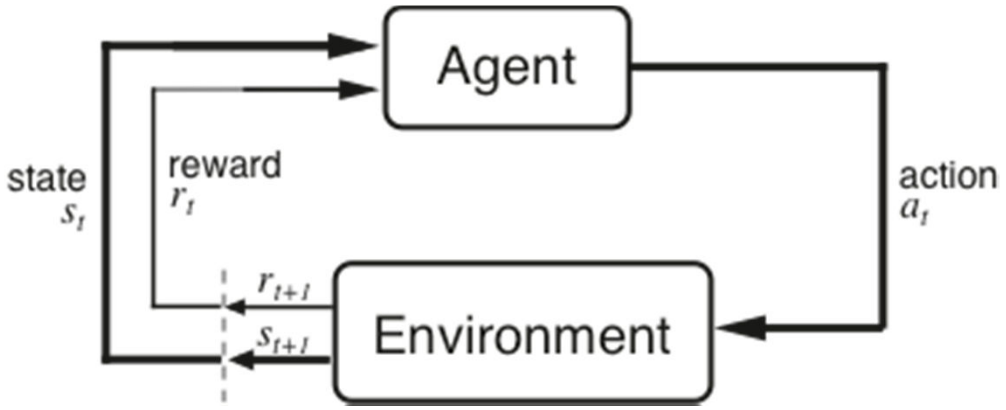
The Agent Learns a Policy

- What is learnt?
  - It learns a control strategy, or *policy,* for choosing actions that achieve its goals

$$\pi : S \rightarrow A$$

- What's the goal?
  - Roughly, the agent's goal is to get as much reward as it can over the long run.
- Reinforcement learning methods specify how the agent changes its policy as a result of experience

The Agent-Environment Interface



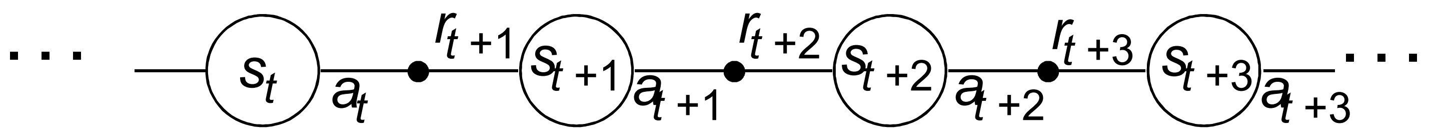
Agent and environment interact at discrete time steps : $t = 0,1,2,\ldots$

Agent observes state at step t : $s_t \in S$

chooses action at step t : $a_t \in A(s_t)$

gets resulting reward : $r_{t+1} = r(s_t, a_t)$

and resulting next state : $s_{t+1} = \delta(s_t, a_t)$



The Learning Task

Agent and environment interact at discrete time steps : $t = 0,1,2,\ldots$

Agent observes state at step t : $s_t \in S$

chooses action at step t : $a_t \in A(s_t)$

gets resulting reward : $r_{t+1} = r(s_t, a_t)$

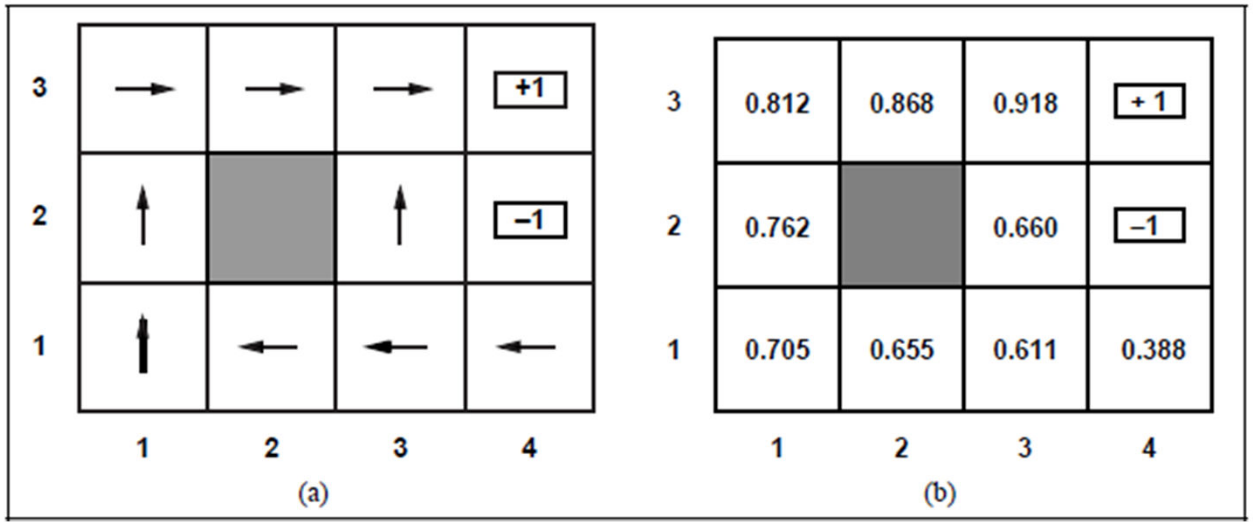
and resulting next state : $s_{t+1} = \delta(s_t, a_t)$

- The functions r (reward function) and δ (transition function) are part of the environment and are not necessarily known to the agent
- The task of the agent is to learn a policy:

$$\pi : S \rightarrow A$$

for selecting its next action a_t based on the current observed state s_t

Example



reward +1 at [4,3], -1 at [4,2] (final states:)
reward -0.04 for each step leading to a nonterminal

a) specific policy π for the 4x3 world
b) utilities of each state

Trials are sequences of state transitions until it reaches one of the end states.
For example:
$(1,1)_{-.04}\rightarrow(1,2)_{-.04}\rightarrow(1,3)_{-.04}\rightarrow(1,2)_{-.04}\rightarrow(1,3)_{-.04}\rightarrow(2,3)_{-.04}\rightarrow(3,3)_{-.04}\rightarrow(4,3)_{+1}$
sub-indices show the reward given to the agent by performing the action

The Learning Task (2)

- Use information from about rewards to learn
- The task of the agent is to learn a policy:

$$\pi : S \rightarrow A$$

for selecting its next action a_t based on the current observed state s_t

- Which policy π do we want the agent to learn?
- The policy that produces the greatest possible cumulative reward for the agent over time.

Value Function

- For each possible policy π the agent might adopt, we can define an evaluation function over states

$$V^{\pi}(s) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \ldots$$

$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where $r_t, r_{t+1}, \ldots$ are generated by following policy π starting at state s

$\gamma, 0 \le \gamma \le 1$, is the **discount rate**

shortsighted $0 \leftarrow \gamma \rightarrow 1$ farsighted

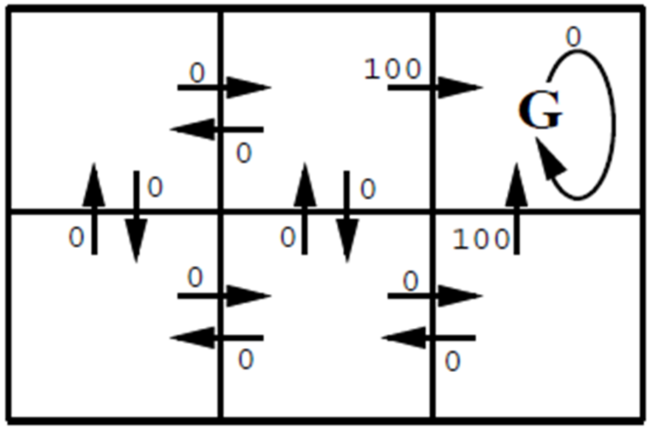
The Learning Task (3)

- Formally stated, the agent's learning task is to learn a policy π that maximizes $V^{\pi}(s)$ for all states *s*.
- Such a policy is an optimal policy and we denote it by π*:

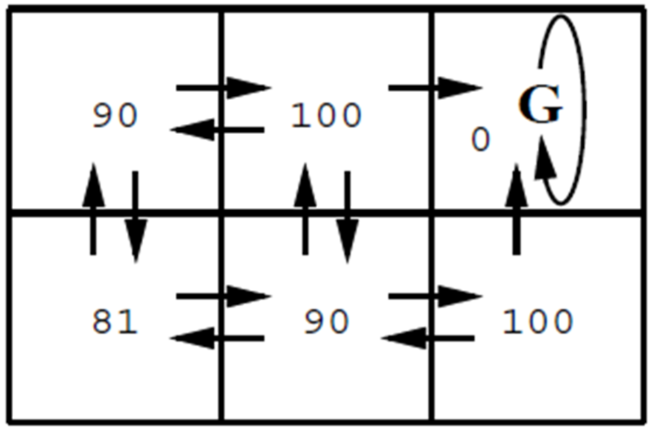
$$\pi^* = \arg\max_{\pi} V^{\pi}(s), (\forall s)$$

We will denote *V*(s)* the value function of the optimal policy (value function obtained by executing the optimal policy)

Exercise



r(s,a) (immediate reward) values



$V^*(s)$ values

- How do we get the V* value of the bottom left corner state?
  - actions: (up, right, right) or (right, right, up)
  - $0 + 0 + \gamma^2\, 100 = 81$

What to Learn

- Passive Reinforcement Learning
  - We might try to have the agent learn the evaluation function V*
  - It could then do a lookahead search to choose best action from any state s because

$$\pi^*(s) = \arg\max_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

- A problem:
  - This works well if agent knows the transition and reward functions, $r(s_t, a_t)$ and $\delta(s_t, a_t)$
  - But when it doesn’t, it can’t choose actions this way

Q Function

- We define an evaluation function *Q(s,a)* so that its value is the maximum discounted cumulative reward that can be achieved starting from state *s* and applying action a :

$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))$$

- Optimal policy: $\pi^*(s) = \arg\max_a [r(s,a) + \gamma V^*(\delta(s,a))]$
- We can rewrite using Q function:

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

- Q is the evaluation function the agent will learn
  - If agent learns Q, it can choose optimal action even without knowing δ!

Q-Learning

Note Q and V^* closely related:

$$V^*(s) = \max_{a'} Q(s, a')$$

Which allows us to write Q recursively as

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t))) \\ &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

Nice! Let $\hat{Q}$ denote learner's current approximation to Q. Consider training rule

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

where s' is the state resulting from applying action a in state s

Q-Learning (2)

- The agent uses the training rule (aka updating rule) to learn the Q function as it explores the world
- The agent maintains a table to store the estimated Q values (for each state-action pair)
  - The table can be initially filled with random values or zeros
- The agent repeatedly observes its current state, chooses an action and executes it, and observes the resulting reward (*r function*) and the new state (*δ function*). It then updates the entry for the estimated Q value, $\hat{Q}(s, a)$

Q-Learning Algorithm

For each s, a initialize table entry $\hat{Q}(s,a) \leftarrow 0$

Observe current state s

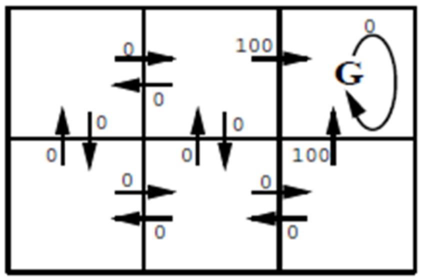
Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s,a)$ as follows:

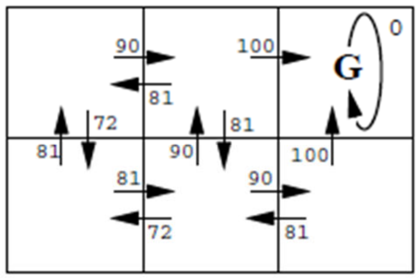
$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

- $s \leftarrow s'$

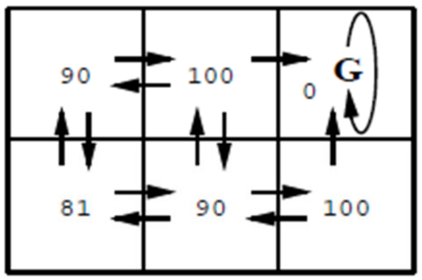
Q-Learning Example



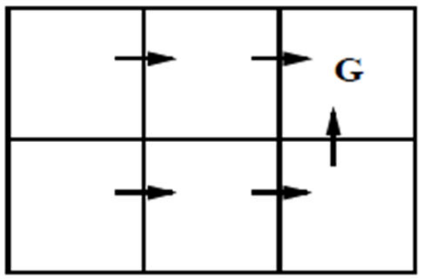
$r(s,a)$ (immediate reward) values



$Q(s,a)$ values

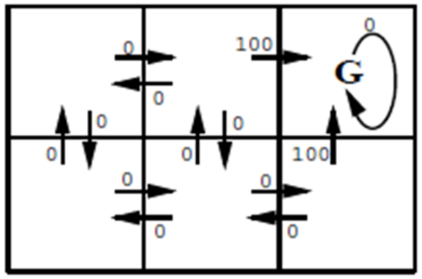


$V^*(s)$ values

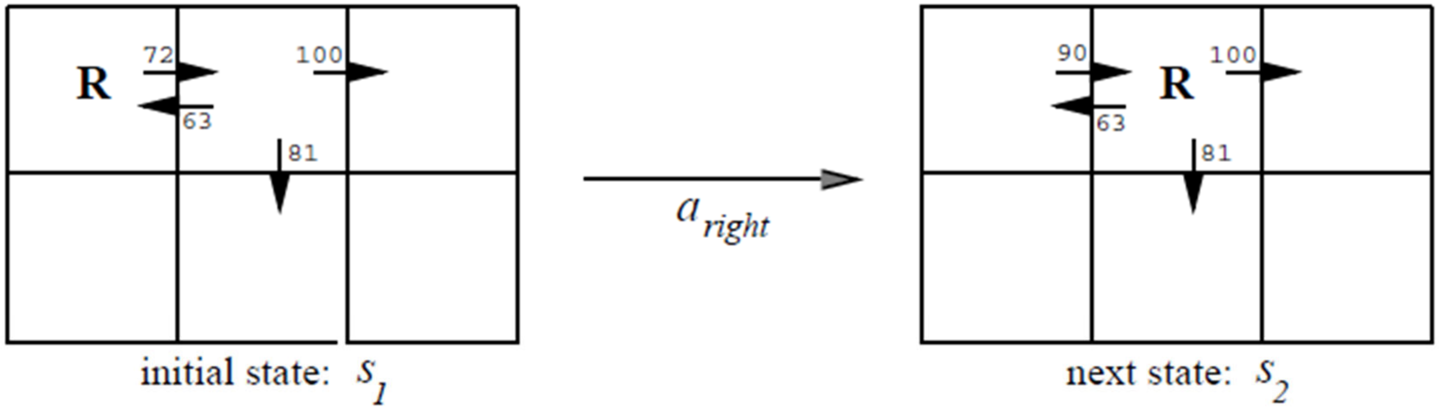


One optimal policy

Exercise



$r(s, a)$ (immediate reward) values



$$\begin{aligned}\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow 0 + 0.9 \ \max\{63, 81, 100\} \\ &\leftarrow 90\end{aligned}$$

Strategies

- Explotation
  - Choose action that maximizes the current estimated Q value
- Exploration
  - Probabilistic approach (probabilities assigned according to current estimated Q value)
    - Actions with higher estimated Q value are assigned higher probabilities, but every action is assigned a nonzero probability (of being chosen)
- How do we learn the Q values?
  - Training episodes (a sequence of actions from some start state to the goal)

Evolving estimated Q values

- How will the estimated Q values (table)?
  - With all values in the table initialized to 0, the agent will mae no changes until it reaches the goal and receives a nonzero reward.
  - This will change the value only for the single transition leading to the goal.
  - On the next episode, if the agent passes through this state adjacent to the goal, its nonzero value will cause an update in some transition two steps from the goal and so on.
- Given a sufficient number of training episodes the information will propagate back through the entire state-action space.

A Real Application: TD-Gammon

- Learn to play Backgammon
- Immediate rewards
  - +100 if win
  - -100 if loose
  - 0 all other states
- Trained by playing 1.5 million games against itself
- Now approximately equal to best human player

Subtleties and Ongoing Research

- Replace $\hat{Q}$ table with neural net or other generalizer
- Handle case where state only partially observable
- Design optimal exploration strategies
- Extend to continuous action, state
- Learn and use $\hat{\delta} : S \times A \rightarrow S$
- Relationship to dynamic programming