Fall 2010 Graduate Course on
Dynamic Learning

Chapter 6: MDP and Reinforcement Learning

October 12, 2010

Byoung-Tak Zhang

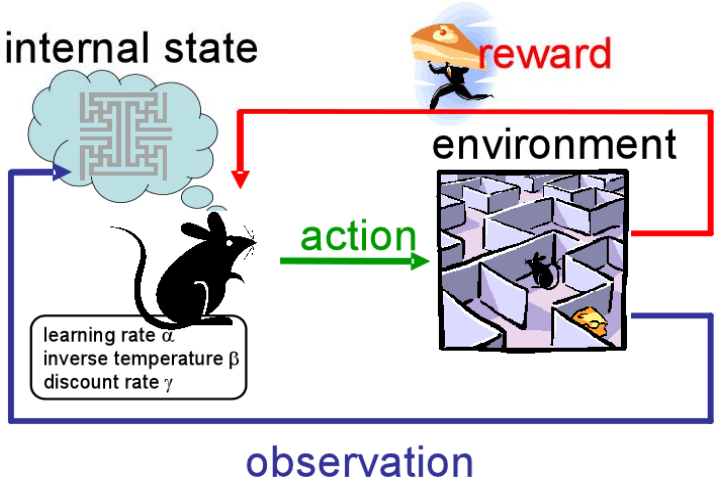
School of Computer Science and Engineering &
Cognitive Science and Brain Science Programs
Seoul National University

http://bi.snu.ac.kr/~btzhang/

Overview

- Motivating Applications
  - Learning robots
  - Web agents
- Markov Decision Processes
  - MDP
  - POMDP
- Reinforcement Learning
  - Adaptive Dynamic Programming (ADP)
  - Temporal Difference (TD) Learning
  - TD-Q Learning

Autonomous Helicopter Control

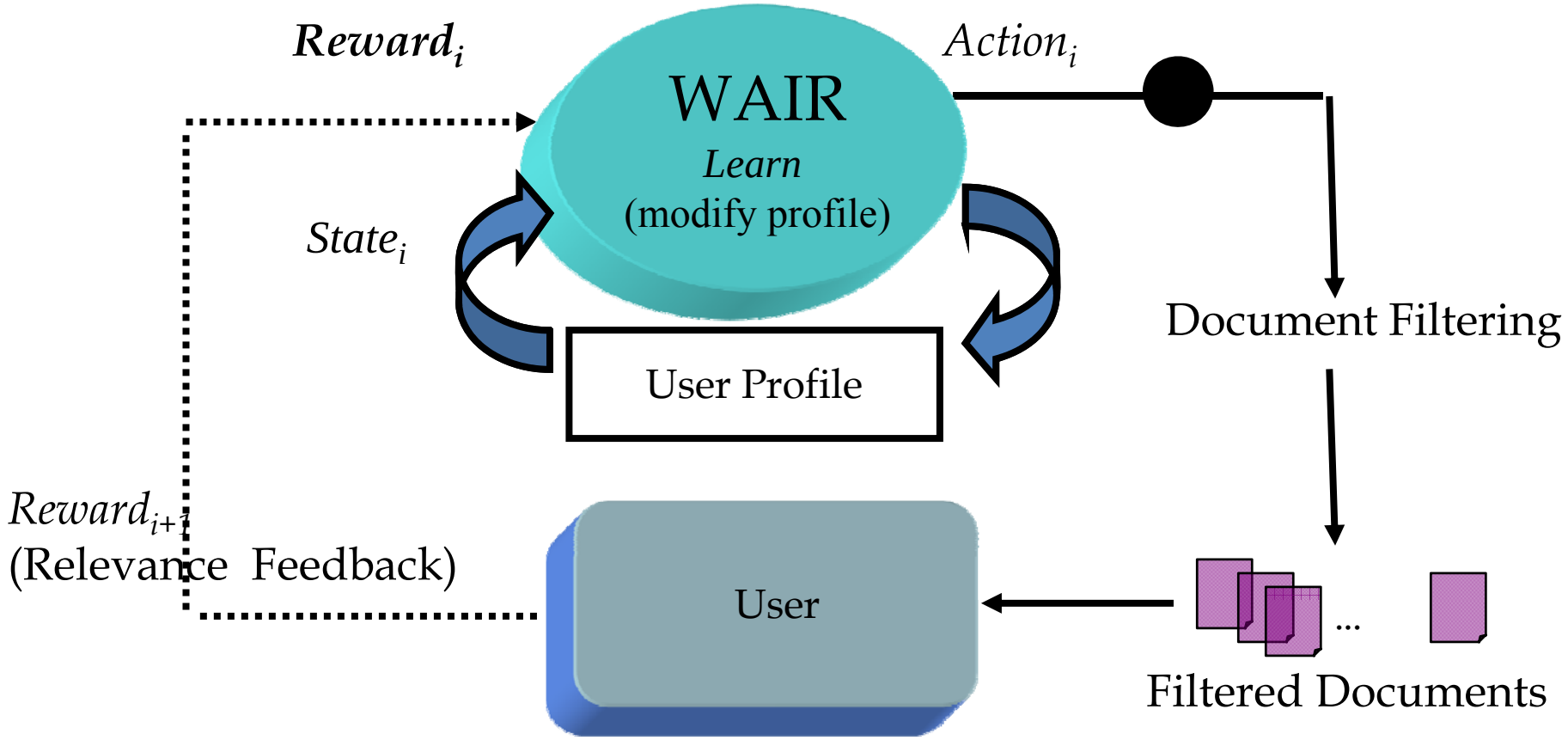


Stanford Autonomous Helicopter - Airshow #2:
http://www.youtube.com/watch?v=VCdxqn0fcnE

Motivating Applications

- Generalized model learning for reinforcement learning on a humanoid robot: http://www.youtube.com/watch?v=mRpX9DFCdwI
- Autonomous spider learns to walk forward by reinforcement learning: http://www.youtube.com/watch?v=RZf8fR1SmNY&feature=related
- Reinforcement learning for a robotic soccer goalkeeper: http://www.youtube.com/watch?v=CIF2SBVY-J0&feature=related

WAIR: Reinforcement Learning Web Agent
for Personalized Information Filtering



Zhang, B.-T. and Seo, Y.-W., *Applied Artificial Intelligence*, 15(7):665-685, 2001

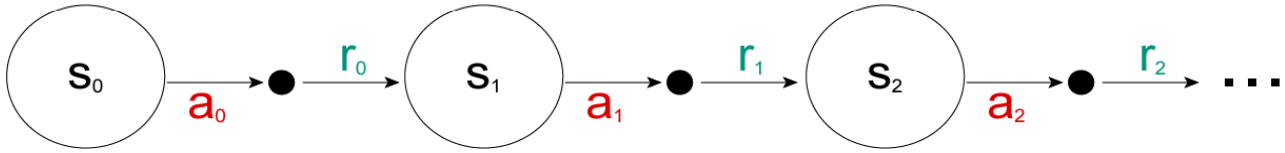
Reinforcement Learning is…

… learning from **trial-and-error** and **reward** by **interaction** with an environment.

Markov Decision Processes (MDPs)

- A simple, general framework for modeling sequential decision making in a stochastic environment
- Early work by Bellman and Howard in the 1950s
- Currently a popular model in OR and AI
- Used to model problems in robotics, finance, agriculture, etc.

Markov Decision Processes (MDPs)

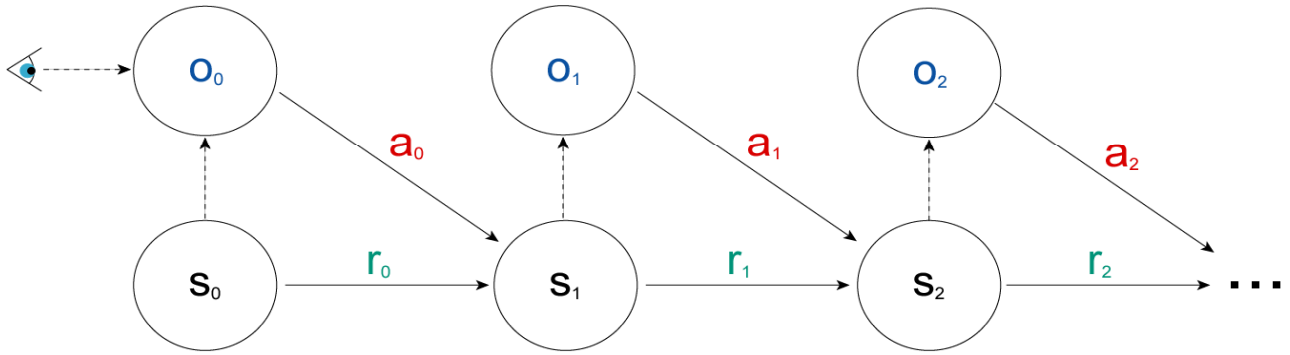


- The decision-making agent starts in some state and chooses an action according to its policy
- This determines an immediate reward and causes a stochastic transition to the next state
- Aim is to find the policy that leads to the highest total reward over T time steps (finite horizon)
- Because of the Markov property, the policy does not have to remember previous states

The Formal Decision Problem - **MDP**

- Given <S, A, P, R, T, K>
  - S is a finite state set (with start state s_0)
  - A is a finite action set
  - P(s' | s, a) is a table of transition probabilities
  - R(s, a, s') is a reward function
- Policy π(s, t) = a
- Is there a policy that yields total reward at least K over finite horizon T
- We will require that T < |S|

POMDP: Partially Observable MDP

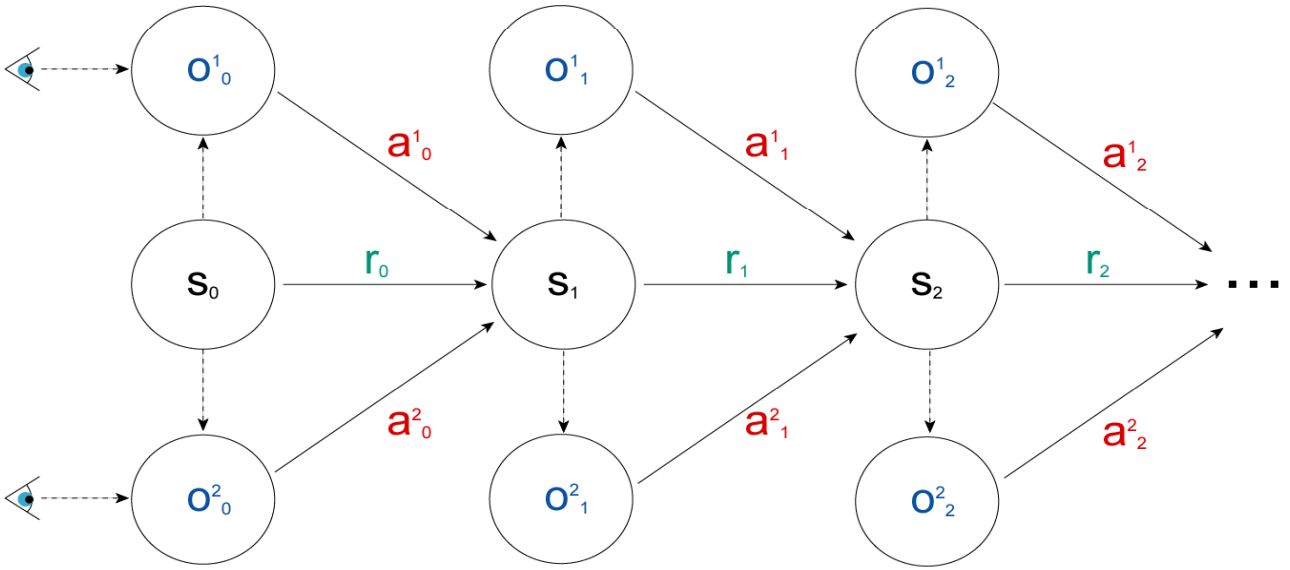


- The agent's observation is a function of the current state
- Now you may need to remember previous observations in order to act optimally

POMDP

- Given <S, A, P, R, Ω, O, T, K>
  - Ω is a finite observation set
  - O(o | s, a, s') is a table of observation probabilities
- Policy $\pi(o_1, \ldots, o_t) = a$
- Does there exist a policy that yields total reward at least K over horizon T?

DEC-POMDP$_2$



- Now two cooperating (**dec**entralized) agents with different observation functions

DEC-POMDP$_2$

- Given <S, A, P, R, Ω_1, Ω_2, O, T, K>
  - Ω_1 and Ω_2 are finite observation sets
  - O(o^1, o^2 | s, a^1, a^2, s') is a table of observation probabilities
- Local policies $\pi^1(o^1_1, \ldots, o^1_t) = a^1$, $\pi^2(o^2_1, \ldots, o^2_t) = a^2$
- Joint policy <π^1, π^2>
- Does there exist a joint policy that yields total reward at least K over horizon T?

Note: **DEC-POMDP$_1$ = POMDP**

DEC-MDP$_2$

- **DEC-POMDP$_2$** with requirement that the state is *uniquely* determined by the tuple of observations:

$$J : \Omega_1 \times \Omega_2 \rightarrow S$$

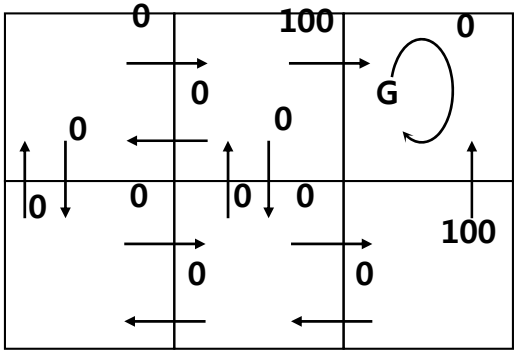
Note: **DEC-MDP$_1$ = MDP**

MDP

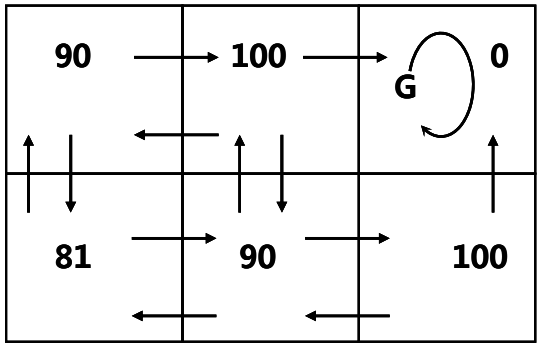
- Can use dynamic programming to “back up” values from the end

$$V(s) = \max_{a \in A} \left(R(s,a) + \sum_{s' \in S} P(s' \mid s,a) V(s') \right)$$

Elements of Reinforcement Learning



r(*state*, *action*)
immediate reward values



V^*(*state*) values

- Value function: maps states to state values

$$V(s) \equiv r(t) + \gamma r(t+1) + \gamma^2 r(t+2) + \ldots$$

Discount factor $\gamma \in [0, 1)$ (here 0.9)

Elements of MDP

- A set S of **states** $\{s_1, s_2, s_3, \ldots\}$
- A set A of **actions** $\{a_1, a_2, a_3, \ldots\}$
- A **transition** function T: S x A → S (deterministic)
  or T: S x A x S → [0, 1] (stochastic)
- A **reward** function R: S x A → Real
  or R: S x A x S → Real
- A **policy** π: S → A (deterministic)
  or π: S x A → [0, 1] (stochastic)

Optimality Criteria

Suppose an agent receives a reward r_t at time t. Then optimal behaviour might:

- Maximise the **sum of expected future rewards**: $\sum_t r_t$
- Maximise over a **finite horizon**: $\sum_{t=0}^{T} r_t$
- Maximise over an **infinite horizon**: $\sum_{t=0}^{\infty} r_t$
- Maximise over a **discounted infinite horizon**: $\sum_{t=0}^{\infty} \gamma^t r_t$
- Maximise **average reward**: $\lim_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} r_t$

Examples of MDPs

- Goal-directed, Indefinite Horizon, Cost Minimization MDP
  - <**S**, **A**, **P**r, **C**, **G**, s_0>
  - Most often studied in planning community
- Infinite Horizon, Discounted Reward Maximization MDP
  - <**S**, **A**, **P**r, **R**, γ>
  - Most often studied in reinforcement learning
- Goal-directed, Finite Horizon, Prob. Maximization MDP
  - <**S**, **A**, **P**r, **G**, s_0, T>
  - Also studied in planning community
- Oversubscription Planning: Non absorbing goals, Reward Max. MDP
  - <**S**, **A**, **P**r, **G**, **R**, s_0>
  - Relatively recent model

Assumptions

- **First-Order Markovian dynamics** (history independence)
  - $\Pr(S^{t+1}|A^t,S^t,A^{t-1},S^{t-1},..., S^0) = \Pr(S^{t+1}|A^t,S^t)$
  - Next state only depends on current state and current action
- **First-Order Markovian reward process**
  - $\Pr(R^t|A^t,S^t,A^{t-1},S^{t-1},..., S^0) = \Pr(R^t|A^t,S^t)$
  - Reward only depends on current state and action
  - As described earlier we will assume reward is specified by a deterministic function R(s)
    - i.e. $\Pr(R^t=R(S^t) \mid A^t,S^t) = 1$
- **Stationary dynamics and reward**
  - $\Pr(S^{t+1}|A^t,S^t) = \Pr(S^{k+1}|A^k,S^k)$ for all t, k
  - The world dynamics do not depend on the absolute time
- **Full observability**
  - Though we can't predict exactly which state we will reach when we execute an action, once it is realized, we know what it is

Policies ("Plans" for MDPs)

- Nonstationary policy
  - π:S x T $\rightarrow$ A, where T is the non-negative integers
  - π(s,t) is action to do at state s with t stages-to-go
  - What if we want to keep acting indefinitely?
- Stationary policy
  - π:S $\rightarrow$ A
  - π(s) is action to do at state s (regardless of time)
  - specifies a continuously reactive controller
- These assume or have these properties:
  - full observability
  - history-independence
  - deterministic action choice

Value of a Policy

- How good is a policy π?
- How do we measure "accumulated" reward?
- **Value function** V: S $\rightarrow \mathbb{R}$ associates value with each state (or each state and time for non-stationary π)
- $V_\pi(s)$ denotes value of policy at state *s*
  - Depends on immediate reward, but also what you achieve subsequently by following π
  - An optimal policy is one that is no worse than any other policy at any state
- The goal of MDP planning is to compute an optimal policy (method depends on how we define value)

Finite-Horizon Value Functions

- We first consider maximizing total reward over a finite horizon
- Assumes the agent has *n* time steps to live
- To act optimally, should the agent use a stationary or non-stationary policy?
- Put another way:
  - If you had only one week to live would you act the same way as if you had fifty years to live?

Finite Horizon Problems

- Value (utility) depends on stage-to-go
  - hence so should policy: nonstationary $\pi(s,k)$
- $V_\pi^k(s)$ is k-stage-to-go value function for π
  - expected total reward after executing π for k time steps

$$V_\pi^k(s) = E\left[\sum_{t=0}^{k} R^t \mid \pi, s\right]$$

$$= E\left[\sum_{t=0}^{k} R(s^t) \mid a^t = \pi(s^t, k-t), s = s^0\right]$$

- Here R^t and s^t are random variables denoting the reward received and state at stage t respectively

Computing Finite-Horizon Value

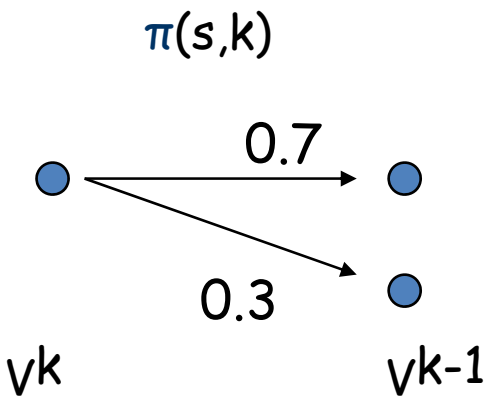
- Can use dynamic programming to compute $V_\pi^k(s)$
  - Markov property is critical for this

(a) $V_\pi^0(s) = R(s), \quad \forall s$

(b) $V_\pi^k(s) = R(s) + \sum_{s'} T(s, \pi(s,k), s') \cdot V_\pi^{k-1}(s')$

immediate reward

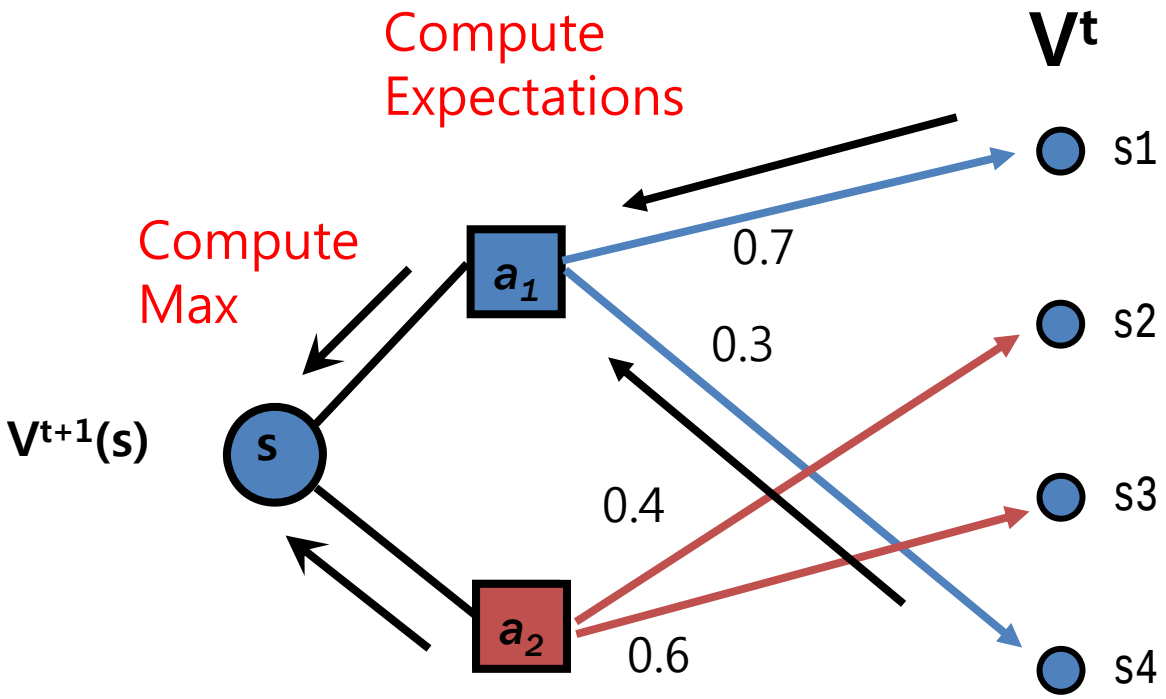
expected future payoff with *k*-1 stages to go



What is time complexity?

Bellman Backup

How can we compute optimal $V^{t+1}(s)$ given optimal V^t ?



$$V^{t+1}(s) = R(s) + \max \{ 0.7\, V^t(s1) + 0.3\, V^t(s4),\ 0.4\, V^t(s2) + 0.6\, V^t(s3) \}$$

Value Iteration: Finite Horizon Case

- Markov property allows exploitation of DP principle for optimal policy construction
  - no need to enumerate $|A|^{Tn}$ possible policies
- Value Iteration

Bellman backup

$$V^0(s) = R(s), \quad \forall s$$

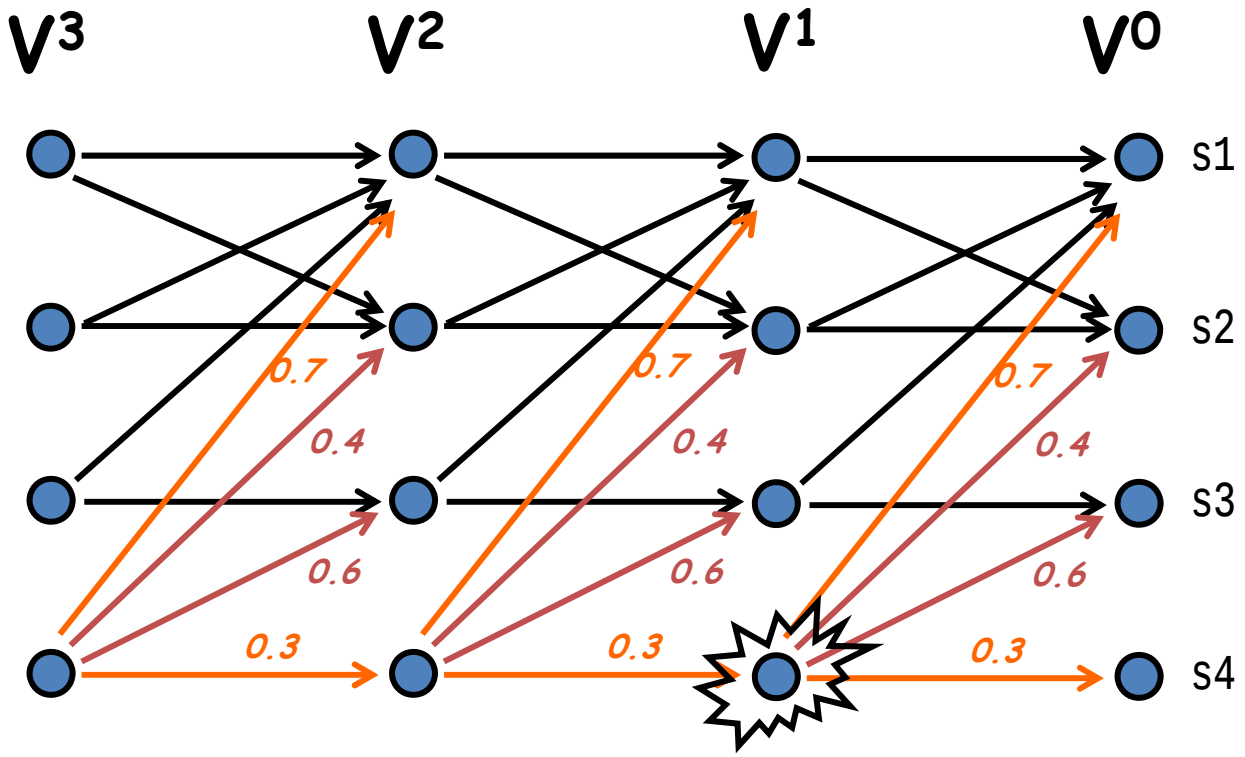
$$V^k(s) = R(s) + \max_a \sum_{s'} T(s,a,s') \cdot V^{k-1}(s')$$

$$\pi^*(s,k) = \arg\max_a \sum_{s'} T(s,a,s') \cdot V^{k-1}(s')$$

V^k is optimal k-stage-to-go value function

$\Pi^*(s,k)$ is optimal k-stage-to-go policy

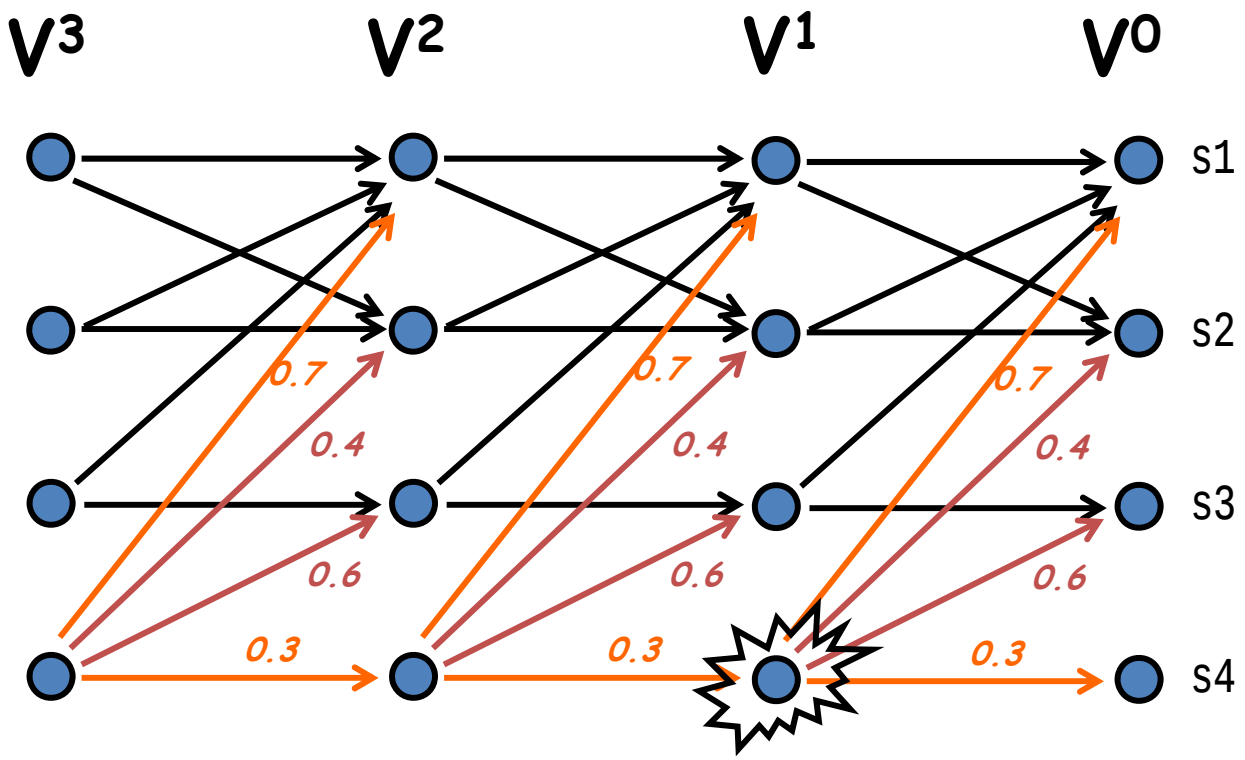
Value Iteration



$V^1(s4) = R(s4) + \max \{ 0.7\ V^0(s1) + 0.3\ V^0(s4)$

$0.4\ V^0(s2) + 0.6\ V^0(s3) \}$

Value Iteration



$\Pi^*(s4,t) = \max\{$ ■ ■ $\}$

Discounted Infinite Horizon MDPs

- Defining value as total reward is problematic with infinite horizons
  - many or all policies have infinite expected reward
  - some MDPs are ok (e.g., zero-cost absorbing states)
- "Trick": introduce discount factor $0 \leq \beta < 1$
  - future rewards discounted by β per time step

$$V_\pi^k(s) = E\left[\sum_{t=0}^{\infty} \beta^t R^t \mid \pi, s\right]$$

- Note: $$V_\pi(s) \leq E\left[\sum_{t=0}^{\infty} \beta^t R^{\max}\right] = \frac{1}{1-\beta} R^{\max}$$
- Motivation: economic? failure prob? convenience?

Computing an Optimal Value Function

- **Bellman equation** for optimal value function

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

  - Bellman proved this is always true
- How can we compute the optimal value function?
  - The MAX operator makes the system non-linear, so the problem is more difficult than policy evaluation
- Notice that the optimal value function is a fixed-point of the Bellman Backup operator B
  - B takes a value function as input and returns a new value function

$$B[V](s) = R(s) + \beta \max_a \sum_{s'} T(s,a,s')V(s')$$

Value Iteration

- Can compute optimal policy using value iteration, just like finite-horizon problems (just include discount term)

$$V^0(s) = 0$$

$$V^k(s) = R(s) + \beta \max_a \sum_{s'} T(s,a,s')V^{k-1}(s')$$

- Will converge to the optimal value function as k gets large. Why?

Policy Evaluation

- Value equation for fixed policy

$$V_\pi(s) = R(s) + \beta \sum_{s'} T(s, \pi(s), s') V_\pi(s')$$

- How can we compute the value function for a policy?
  - we are given *R* and *Pr*
  - simple linear system with *n* variables (each variables is value of a state) and *n* constraints (one value equation for each state)
  - Use linear algebra (e.g. matrix inverse)

Policy Iteration

- Given fixed policy, can compute its value exactly:

$$V_\pi(s) = R(s) + \beta \sum_{s'} T(s, \pi(s), s') V_\pi(s')$$

- Policy iteration exploits this: iterates steps of policy evaluation and policy improvement

1. Choose a random policy π
2. Loop:
   (a) Evaluate V_π
   (b) For each s in S, set
   (c) Replace π with π'

Until no improving action possible at any state

Policy improvement

$$\pi'(s) = \arg\max_a \sum_{s'} T(s, a, s') V_\pi(s')$$

Policy Iteration Notes

- Each step of policy iteration is guaranteed to strictly improve the policy at some state when improvement is possible
- Convergence assured (Howard)
  - intuitively: no local maxima in value space, and each policy must improve value; since finite number of policies, will converge to optimal policy
- Gives exact value of optimal policy

Value Iteration vs. Policy Iteration

- Which is faster? VI or PI
  - It depends on the problem
- VI takes more iterations than PI, but PI requires more time on each iteration
  - PI must perform policy evaluation on each step which involves solving a linear system
- Complexity:
  - There are at most exp(n) policies, so PI is no worse than exponential time in number of states
  - Empirically $O(n)$ iterations are required
  - Still no polynomial bound on the number of PI iterations (open problem)!

Adaptive Dynamic Programming (ADP)

For each s, initialize $V(s)$, $P(s'|s,a)$ and $R(s,a)$

Initialize s to current state that is perceived

Loop forever

{

Select an action a and execute it (using current model R and P) using

$$a = \arg\max_{a}(R(s,a) + \gamma \sum_{s'} P(s'|,s,a)V(s'))$$

Receive immediate reward r and observe the new state s'

Using the transition tuple <s,a,s',r> to update model R and P

For all the sate s, update $V(s)$ using the updating rule

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

$s = s'$

}

How to Learn Model?

- Use the transition tuple <s, a, s', r> to learn $P(s'|s,a)$ and $R(s,a)$. That's supervised learning!
  - Since the agent can get every transition (s, a, s',r) directly, so take (s,a) and s' as an input-output example of the transition probability function $P(s'|s,a)$.
  - Different techniques in the supervised learning
    - Neural networks
    - Decision trees
  - Use r and $P(s'|s,a)$ to learn $R(s,a)$

$$R(s,a) = \sum_{s'} P(s' \mid s,a) r$$

From ADP to TD Learning

- In each step, ADP updates V for all states s'

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

  - This is intractable for large state space!
  - Improve this by *prioritized-sweeping.*
- Temporal difference (TD) learning:
  - Adjust the estimated utility value V of the current state based on its immediate reward $R(s)$ and the estimated value of the next state s'.

$$V(s) = V(s) + \alpha(R(s) + V(s') - V(s))$$

TD-Q Learning

- Define Q-value function

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

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

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

- Recall: TD learning

$$V(s) = V(s) + \alpha(R(s) + V(s') - V(s))$$

- Key idea of TD-Q learning

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

- Note: Delta rule (neural network learning)

$$w(t) = w(t) + \alpha(d - o)x$$

TD-Q Learning algorithm

For each pair (s,a), initialize $Q(s,a)$
Observe the current state s
Loop forever
{
Select an action a and execute it

$$a = \arg\max_{a} Q(s,a)$$

Receive immediate reward r and observe the new state s'
Update $Q(s,a)$

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

$s=s'$
}

Generalization in RL

- So far we assumed that all the functions learned by the agent are (V, P, R, Q) are tabular forms, i.e. it is possible to enumerate state and action spaces.
- Use generalization techniques to deal with large state or action space.
  - Function approximation techniques
  - Neural networks
  - Other supervised/unsupervised learning methods

Alternatives

- Function approximation of the Q-table:
  - Neural networks
  - Decision trees
  - Gradient descent methods
- Reinforcement learning variants:
  - Relational reinforcement learning
  - Hierarchical reinforcement learning
  - Intrinsically motivated reinforcement learning

References and Further Reading

- Sutton, R., Barto, A., (2000) *Reinforcement Learning: an Introduction*, The MIT Press

  http://www.cs.ualberta.ca/~sutton/book/the-book.html
- Kaelbling, L., Littman, M., Moore, A., (1996) Reinforcement Learning: a Survey, *Journal of Artificial Intelligence Research*, **4**:237-285
- Barto, A., Mahadevan, S., (2003) Recent Advances in Hierarchical Reinforcement Learning, *Discrete Event Dynamic Systems: Theory and Applications*, **13**(4):41-77