Prediction and Search in Probabilistic Worlds

# Markov Systems, Markov Decision Processes, and Dynamic Programming

Note to other teachers and users of these slides. Andrew would be delighted if you found this source material useful in giving your own lectures. Feel free to use these slides verbatim, or to modify them to fit your own needs. PowerPoint originals are available. If you make use of a significant portion of these slides in your own lecture, please include this message, or the following link to the source repository of Andrew's tutorials: http://www.cs.cmu.edu/~awm/tutorials . Comments and corrections gratefully received.

**Andrew W. Moore**

**Professor**

**School of Computer Science**

**Carnegie Mellon University**

www.cs.cmu.edu/~awm

awm@cs.cmu.edu

412-268-7599



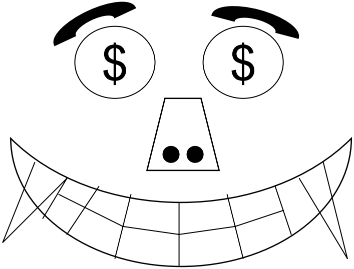
April 21st, 2002

# Discounted Rewards

An assistant professor gets paid, say, 20K per year.

How much, in total, will the A.P. earn in their life?

20 + 20 + 20 + 20 + 20 + … = Infinity




What's wrong with this argument?

# Discounted Rewards

"A reward (payment) in the future is not worth quite as much as a reward now."

- Because of chance of obliteration
- Because of inflation

Example:

Being promised $10,000 next year is worth only 90% as much as receiving $10,000 right now.

Assuming payment *n* years in future is worth only $(0.9)^n$ of payment now, what is the AP's Future Discounted Sum of Rewards ?

# Discount Factors

People in economics and probabilistic decision-making do this all the time.

The "Discounted sum of future rewards" using discount factor $\gamma$" is

(reward now) +

$\gamma$ (reward in 1 time step) +

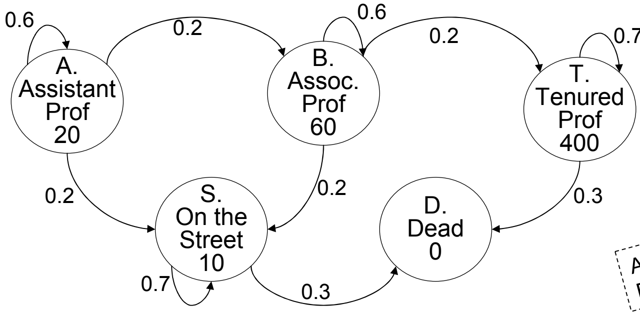
$\gamma^2$ (reward in 2 time steps) +

$\gamma^3$ (reward in 3 time steps) +

:

: (infinite sum)

# The Academic Life




Assume Discount Factor $\gamma = 0.9$

Define:

$J_A$ = Expected discounted future rewards starting in state A

$J_B$ = Expected discounted future rewards starting in state B

$J_T$ = " " " " " " " T

$J_S$ = " " " " " " " S

$J_D$ = " " " " " " " D

How do we compute $J_A$, $J_B$, $J_T$, $J_S$, $J_D$ ?

# Computing the Future Rewards of an Academic

# A Markov System with Rewards…

- Has a set of states $\{S_1\ S_2 \cdots S_N\}$
- Has a transition probability matrix

$$P = \begin{pmatrix} P_{11}\ P_{12} \cdots P_{1N} \\ P_{21} \\ \vdots \\ P_{N1} \quad \cdots \quad P_{NN} \end{pmatrix} \qquad P_{ij} = \text{Prob}(\text{Next} = S_j \mid \text{This} = S_i)$$

- Each state has a reward. $\{r_1\ r_2 \cdots r_N\}$
- There's a discount factor $\gamma$ . $0 < \gamma < 1$

On Each Time Step …

0. Assume your state is $S_i$
1. You get given reward $r_i$
2. You randomly move to another state
   $$P(\text{NextState} = S_j \mid \text{This} = S_i) = P_{ij}$$
3. All future rewards are discounted by $\gamma$

# Solving a Markov System

Write $J^*(S_i)$ = expected discounted sum of future rewards starting in state $S_i$

$J^*(S_i) = r_i + \gamma$ x (Expected future rewards starting from your next state)

$$= r_i + \gamma(P_{i1}J^*(S_1)+P_{i2}J^*(S_2)+ \cdots P_{iN}J^*(S_N))$$

Using vector notation write

$$\underline{J}=\begin{pmatrix} J^*(S_1) \\ J^*(S_2) \\ : \\ J^*(S_N) \end{pmatrix} \qquad \underline{R}=\begin{pmatrix} r_1 \\ r_2 \\ : \\ r_N \end{pmatrix} \qquad \underline{P}=\begin{pmatrix} P_{11} & P_{12} & \cdot\cdot & P_{1N} \\ P_{21} & \ddots & & \\ : & & & \\ P_{N1} & P_{N2} & \cdot\cdot & P_{NN} \end{pmatrix}$$

Question: can you invent a closed form expression for $\underline{J}$ in terms of $\underline{R}$ $\underline{P}$ and $\gamma$ ?

# Solving a Markov System with Matrix Inversion

- Upside: You get an exact answer
- Downside:

# Solving a Markov System with Matrix Inversion

- Upside: You get an exact answer

- Downside: If you have 100,000 states you're solving a 100,000 by 100,000 system of equations.

# Value Iteration: another way to solve a Markov System

Define

$J^1(S_i)$ = Expected discounted sum of rewards over the next 1 time step.

$J^2(S_i)$ = Expected discounted sum rewards during next 2 steps

$J^3(S_i)$ = Expected discounted sum rewards during next 3 steps

:

$J^k(S_i)$ = Expected discounted sum rewards during next *k* steps

$J^1(S_i) =$ (what?)

$J^2(S_i) =$ (what?)

:

$J^{k+1}(S_i) =$ (what?)

# Value Iteration: another way to solve a Markov System

Define

$J^1(S_i)$ = Expected discounted sum of rewards over the next 1 time step.

$J^2(S_i)$ = Expected discounted sum rewards during next 2 steps

$J^3(S_i)$ = Expected discounted sum rewards during next 3 steps

:

$J^k(S_i)$ = Expected discounted sum rewards during next *k* steps

N = Number of states

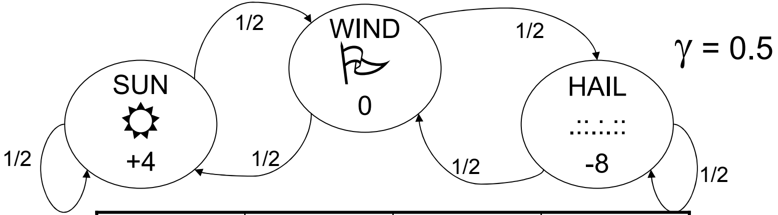
$J^1(S_i) = r_i$ (what?)

$J^2(S_i) = r_i + \gamma \sum_{j=1}^{N} p_{ij} J^1(s_j)$ (what?)

:

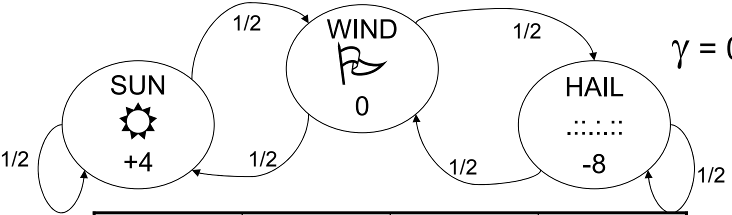
$J^{k+1}(S_i) = r_i + \gamma \sum_{j=1}^{N} p_{ij} J^k(s_j)$ (what?)

# Let's do Value Iteration




| k | $J^k$(SUN) | $J^k$(WIND) | $J^k$(HAIL) |
|---|---|---|---|
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |

# Let's do Value Iteration




$\gamma = 0.5$

| k | $J^k$(SUN) | $J^k$(WIND) | $J^k$(HAIL) |
|---|---|---|---|
| 1 | 4 | 0 | -8 |
| 2 | 5 | -1 | -10 |
| 3 | 5 | -1.25 | -10.75 |
| 4 | 4.94 | -1.44 | -11 |
| 5 | 4.88 | -1.52 | -11.11 |

# Value Iteration for solving Markov Systems

- Compute $J^1(S_i)$ for each $j$
- Compute $J^2(S_i)$ for each $j$

:

- Compute $J^k(S_i)$ for each $j$

As $k \to \infty$ $J^k(S_i) \to J^*(S_i)$ . Why?

When to stop? When

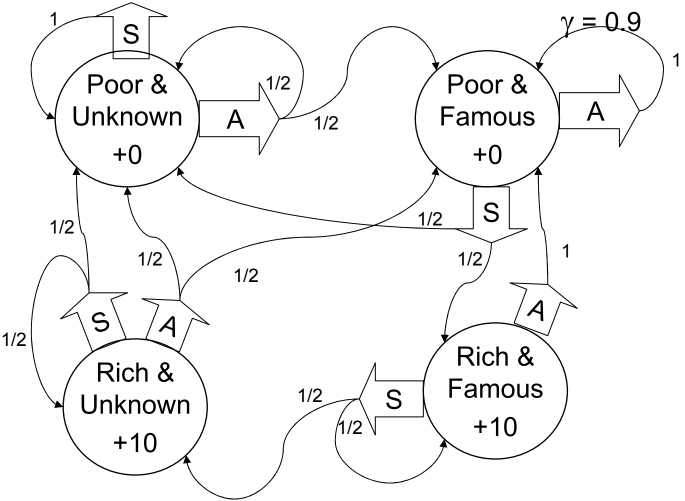
$$\max_i \left| J^{k+1}(S_i) - J^k(S_i) \right| < \xi$$

This is faster than matrix inversion ($N^3$ style)
if the transition matrix is sparse

# A Markov Decision Process

You run a startup company.

In every state you must choose between Saving money or Advertising.



# Markov Decision Processes

An MDP has…

- A set of states $\{s_1 \cdots S_N\}$
- A set of actions $\{a_1 \cdots a_M\}$
- A set of rewards $\{r_1 \cdots r_N\}$ (one for each state)
- A transition probability function

$$P_{ij}^{k} = \text{Prob}\left(\text{Next} = j \middle| \text{This} = i \text{ and I use action } k\right)$$

On each step:

0. Call current state $S_i$
1. Receive reward $r_i$
2. Choose action $\in \{a_1 \cdots a_M\}$
3. If you choose action $a_k$ you'll move to state $S_j$ with probability $P_{ij}^{k}$
4. All future rewards are discounted by $\gamma$

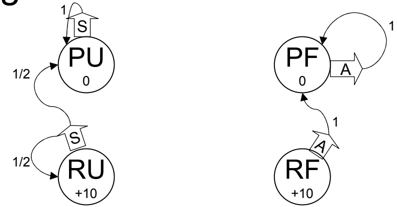
# A Policy

A policy is a mapping from states to actions.

Examples

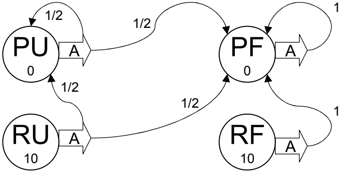
Policy Number 1:

| STATE → | ACTION |
|---|---|
| PU | S |
| PF | A |
| RU | S |
| RF | A |




Policy Number 2:

| STATE → | ACTION |
|---|---|
| PU | A |
| PF | A |
| RU | A |
| RF | A |




- How many possible policies in our example?
- Which of the above two policies is best?
- How do you compute the optimal policy?

# Interesting Fact

For every M.D.P. there exists an optimal policy.

It's a policy such that for every possible start state there is no better option than to follow the policy.

(Not proved in this lecture)

# Computing the Optimal Policy

Idea One:

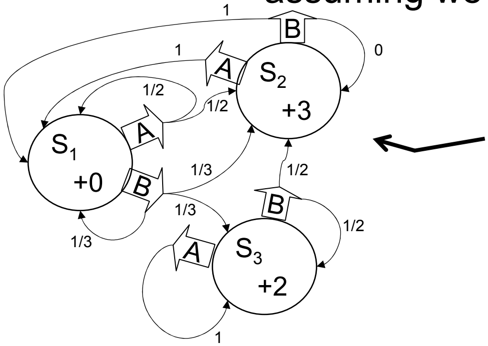
Run through all possible policies.

Select the best.

What's the problem ??

# Optimal Value Function

Define $J^*(S_i)$ = Expected Discounted Future Rewards, starting from state $S_i$, assuming we use the optimal policy




Question

What (by inspection) is an optimal policy for that MDP?

(assume $\gamma = 0.9$)

What is $J^*(S_1)$ ?
What is $J^*(S_2)$ ?
What is $J^*(S_3)$ ?

# Computing the Optimal Value Function with Value Iteration

Define

$J^k(S_i)$ = Maximum possible expected sum of discounted rewards I can get if I start at state $S_i$ and I live for *k* time steps.

Note that $J^1(S_i) = r_i$

# Let's compute $J^k(S_i)$ for our example

| k | $J^k(PU)$ | $J^k(PF)$ | $J^k(RU)$ | $J^k(RF)$ |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |

# Let's compute $J^k(S_i)$ for our example

| k | $J^k(PU)$ | $J^k(PF)$ | $J^k(RU)$ | $J^k(RF)$ |
|---|---|---|---|---|
| 1 | 0 | 0 | 10 | 10 |
| 2 | 0 | 4.5 | 14.5 | 19 |
| 3 | 2.03 | 6.53 | 25.08 | 18.55 |
| 4 | 3.852 | 12.20 | 29.63 | 19.26 |
| 5 | 7.22 | 15.07 | 32.00 | 20.40 |
| 6 | 10.03 | 17.65 | 33.58 | 22.43 |

# Bellman's Equation

$$J^{n+1}(S_i) = \max_k \left[ r_i + \gamma \sum_{j=1}^{N} P_{ij}^k J^n(S_j) \right]$$

## Value Iteration for solving MDPs

- Compute $J^1(S_i)$ for all *i*
- Compute $J^2(S_i)$ for all *i*
- :
- Compute $J^n(S_i)$ for all *i*

…..until converged

$$\left[ \text{converged when } \max_i \left| J^{n+1}(S_i) - J^n(S_i) \right| \langle \xi \right]$$

…Also known as

Dynamic Programming

# Finding the Optimal Policy

1. Compute $J^*(S_i)$ for all i using Value Iteration (a.k.a. Dynamic Programming)

2. Define the best action in state $S_i$ as

$$\arg\max_k \left[ r_i + \gamma \sum_j P_{ij}^k J^*\left(S_j\right) \right]$$

(Why?)

# Applications of MDPs

This extends the search algorithms of your first lectures to the case of probabilistic next states.

Many important problems are MDPs….

… Robot path planning

… Travel route planning

… Elevator scheduling

… Bank customer retention

… Autonomous aircraft navigation

… Manufacturing processes

… Network switching & routing

# Asynchronous D.P.

Value Iteration:

“Backup $S_1$”, “Backup $S_2$”, ···· “Backup $S_N$”,
then “Backup $S_1$”, “Backup $S_2$”, ····
repeat :
: There's no reason that you need to do the backups in order!

Random Order …still works. Easy to parallelize (Dyna, Sutton 91)

On-Policy Order

Simulate the states that the system actually visits.

Efficient Order

e.g. Prioritized Sweeping [Moore 93]
Q-Dyna [Peng & Williams 93]

# Policy Iteration

Another way to compute optimal policies

Write $\pi(S_i)$ = action selected in the $i$'th state. Then $\pi$ is a policy.

Write $\pi^t$ = $t$'th policy on $t$'th iteration

Algorithm:

$\pi^\circ$ = Any randomly chosen policy

$\forall i$ compute $J^\circ(S_i)$ = Long term reward starting at $S_i$ using $\pi^\circ$

$\pi_1(S_i)$ = $\arg\max_a \left[ r_i + \gamma \sum_j P_{ij}^a J^\circ(S_j) \right]$

$J_1$ = ….

$\pi_2(S_i)$ = ….

… Keep computing $\pi^1$ , $\pi^2$ , $\pi^3$ …. until $\pi^k = \pi^{k+1}$ . You now have an optimal policy.

# Policy Iteration & Value Iteration: Which is best ???

It depends.

Lots of actions? Choose Policy Iteration

Already got a fair policy? Policy Iteration

Few actions, acyclic? Value Iteration

Best of Both Worlds:

Modified Policy Iteration [Puterman]

…a simple mix of value iteration and policy iteration

3rd Approach

Linear Programming

# Time to Moan

What’s the biggest problem(s) with what we’ve seen so far?

# Dealing with large numbers of states

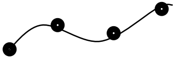
Don't use a Table…

| STATE | VALUE |
|---|---|
| $s_1$ | |
| $S_2$ | |
| : | |
| $S_{15122189}$ | |

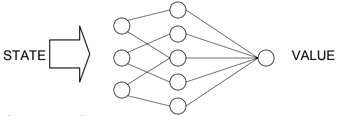
use…

(Generalizers)

Splines

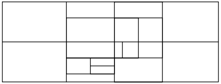


A Function Approximator



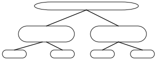

(Hierarchies)

Variable Resolution

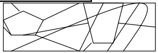


[Munos 1999]

Multi Resolution



Memory Based



# Function approximation for value functions

Polynomials → [Samuel, Boyan, Much O.R. Literature]

Neural Nets → [Barto & Sutton, Tesauro, Crites, Singh, Tsitsiklis]

Backgammon, Pole Balancing, Elevators, Tetris, Cell phones

Checkers, Channel Routing, Radio Therapy

Splines → Economists, Controls

**Downside:** All convergence guarantees disappear.

# Memory-based Value Functions

J("state") = J(most similar state in memory to "state")

or

Average J(20 most similar states)

or

Weighted Average J(20 most similar states)

[Jeff Peng, Atkenson & Schaal,

Geoff Gordon, ← proved stuff

Scheider, Boyan & Moore 98]

"Planet Mars Scheduler"

# Hierarchical Methods

Continuous State Space: "Split a state when statistically significant that a split would improve performance"

Discrete Space:

Chapman & Kaelbling 92, McCallum 95 (includes hidden state)

A kind of Decision Tree Value Function

Continuous Space

e.g. Simmons et al 83, Chapman & Knelbling 92, Mark Ring 94 …, Munos 96

with interpolation!

"Prove needs a higher resolution"

Moore 93, Moore & Atkeson 95

Multiresolution

A hierarchy with high level "managers" abstracting low level "servants"

<u>Many</u> O.R. Papers, Dayan & Sejnowski's Feudal learning, Dietterich 1998 (MAX-Q hierarchy) Moore, Baird & Kaelbling 2000 (airports Hierarchy)

# What You Should Know

- Definition of a Markov System with Discounted rewards
- How to solve it with Matrix Inversion
- How (and why) to solve it with Value Iteration
- Definition of an MDP, and value iteration to solve an MDP
- Policy iteration
- Great respect for the way this formalism generalizes the deterministic searching of the start of the class
- But awareness of what has been sacrificed.