Reinforcement Learning

[Chapter 13]
[R&N, Ch20]; [Barto/Sutton]; [Kaelbling *et al.* survey]
[TGD, "4 Current Trends", 1997]

- Motivation: *Control learning (backgammon)*

- Framework
  - Markov Decision Process
  - Computing Optimal Policy (given Model)
    - Policy Iteration
      (Evaluating fixed policy)
    - Value Iteration

- Extensions ⇒ Reinforcement Learning
  1. Stochastic approx to backups
     TD-learning, $TD(\lambda)$-learning
  2. Value function approximation
  3. Model-free learning
     Q-learning

- Topics

Reinforcement Learning

So far... Learning $\equiv$ "classification learning"

$\approx$ "function approx":

Agent receives explicit training data $\{\langle x_i, f(x_i)\rangle\}_i$

seeks $h(\cdot) \approx f(\cdot)$

- What if feedback is not so clear/immediate?

Less generous environment:

(Agent receives no examples, but can "explore")

Agent has { no *a priori* model
no pre-defined utility function

Eg: Playing a game:

1. No "teacher" providing examples
2. Feedback only after many actions

Final "reward" ("reinforcement", "feedback"):

win / loss / draw

Agent NEVER told

- correct action, nor even
- which individual actions good/bad

Backgammon World

Task: Learn to play backgammon

1. Given { $\langle$board$_i$, optAction$_i\rangle$ }
   Standard learning... f: board $\mapsto$ optAction

2. Given { $\langle$board$_i$, utility$_i\rangle$ }
   ...utility $\approx$ chance of winning
   Standard learning... u: board $\mapsto \Re$
   (then take action producing best utility)

3. Given { $\langle$**FINAL**board$_i$, utility$_i\rangle$ }:
   Only feedback in final state: { Win, Loss }
   Hmmm...

$\Rightarrow$ Use **Reinforcement learning**
to compute utility for intermediate states!

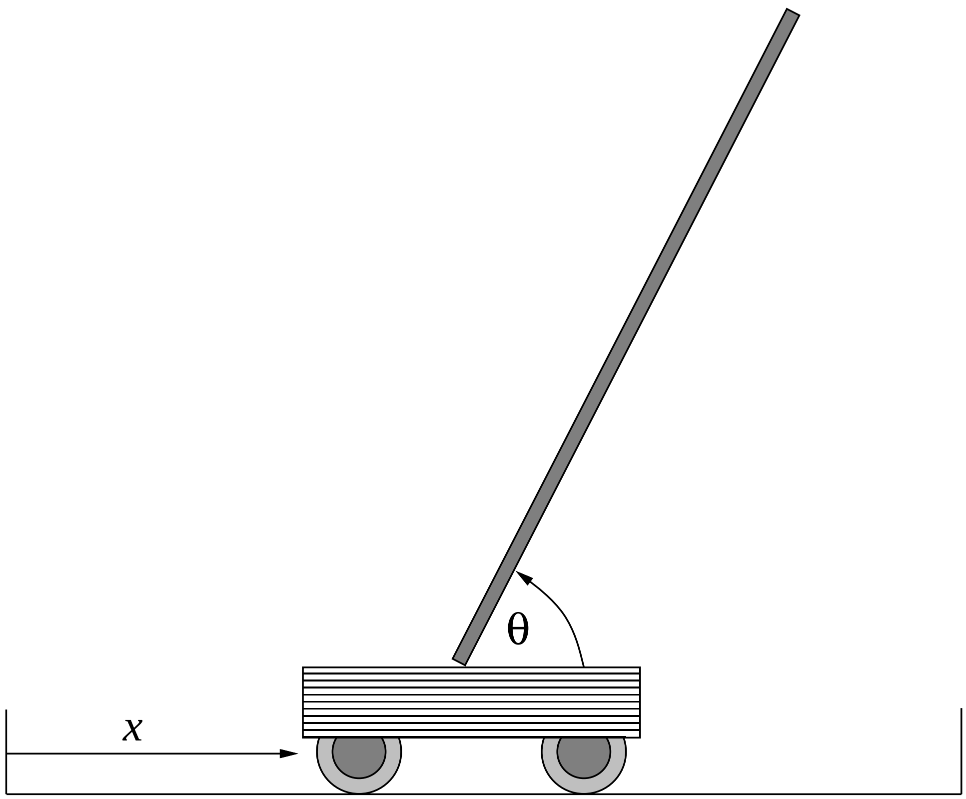
TD-Gammon

[Tesauro, 1995]

- Learn to play Backgammon
- Immediate reward
  - $+100$ if win
  - -100 if lose
  - 0 for all other states
- Starting from random play;
  Trained by . . .
  playing 1.5 million games against itself

Now: $\approx$ best human player

Pole Balancing



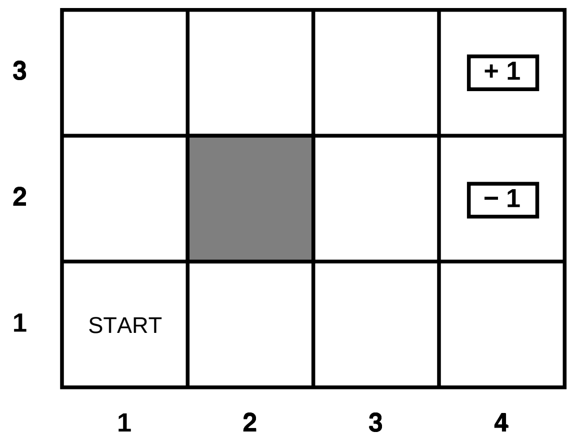
Control Learning, in General

- *Consider learning to choose actions,* eg,
  - Learning to play game (Backgammon)
  - Robot learning to dock on battery charger
  - Learning to choose actions to optimize factory output

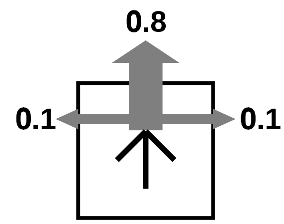
- *Several problem characteristics:*
  - Delayed reward
  - Opportunity for active exploration
  - State is only partially observable
  - May need to learn multiple tasks with same sensors/effectors

- **Successes:**
  + World-class Backgammon [Tesauro'95]
  + Checkers program [Samuels'59]
  + Job-shop scheduling [Zhang/Dietterich'95]
  + Real-time scheduling of passenger elevators [Crites/Barto'95]

Simple Sequential Decision Problem



(a)



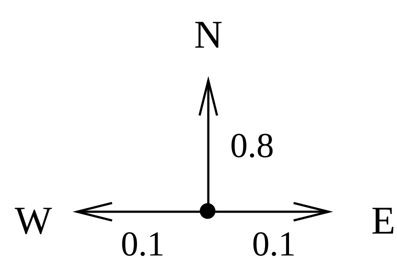
(b)

- Agent can move $\{$ `North`, `South`, `East`, `West` $\}$
  Terminates on reaching $[4,2]$ or $[4,3]$

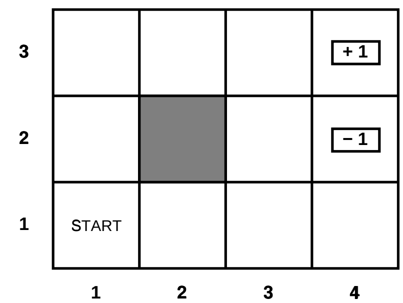
Transition Model: $\boxed{M_{ij}^a \quad = \quad P(\, S_{t+1} = j \,|\, S_t = i,\ A = a\,)}$
$=$ prob of reaching state j if perform action a
from state i

If actions have DETERMINISTIC EFFECTS ($M_{i,j}^a \in \{0,1\}$)
$\Rightarrow$ (std) Planning Problem

but... Actions NOT reliable:
Action $=$ `North` $\equiv$



Immediate Reward & Total Utility



- **Immediate reward:**

$$R(s_{t+1}) = \begin{cases} -\frac{1}{25} & \text{if } S_{t+1} \neq [4,2] \text{ or } [4,3] \\ 1-\frac{1}{25} & \text{if } S_{t+1} = [4,3] \\ -1-\frac{1}{25} & \text{if } S_{t+1} = [4,2] \end{cases}$$

- **Utility function** of *sequence* of actions+states
  $\equiv$ cumulative immediate reward:

$$U([s_0, a_1, s_1, \ldots, a_n, s_n]) = \sum_t R(s_{t+1})$$

$$= \begin{cases} 1-\frac{n}{25} & \text{if } s_n = [4,3] \\ -1-\frac{n}{25} & \text{if } s_n = [4,2] \end{cases}$$

Note: Utility depends on EPISODE

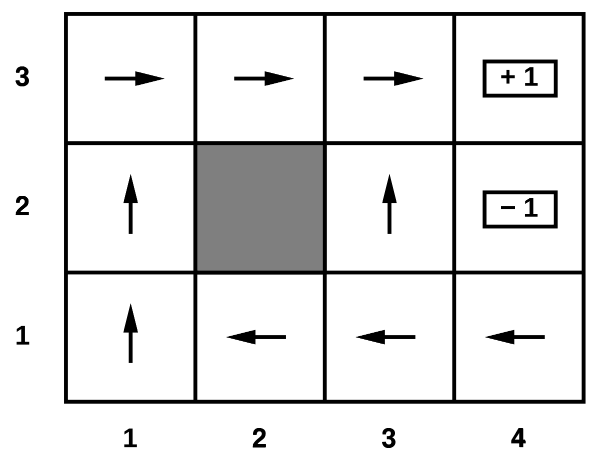
Most reward after SERIES of states
($\equiv$ sequence of actions)
not single state

Observable (Accessible) Environment

- After each action,
  agent can determine resulting state s_{t+1}

$\Rightarrow$ Agent just needs to know
optimal action for each state.

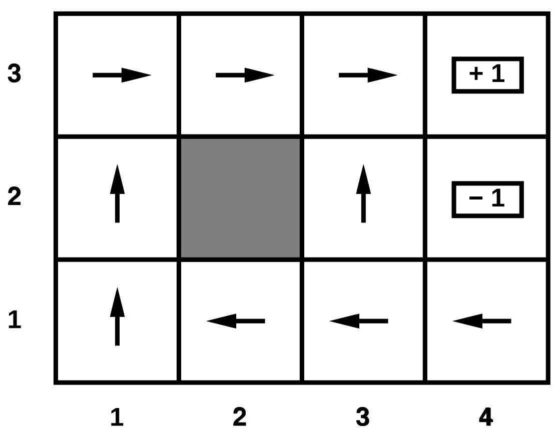
- Agent $\approx$ "**Policy**" π : State $\mapsto$ Action



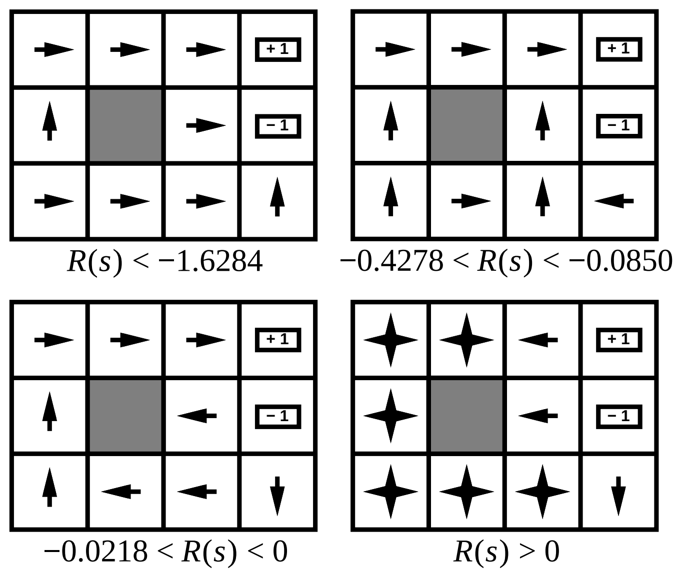
Note: Don't need optimal action sequence
just need optimal policy!

- Agent, using π, is "deterministic" ("reflex")

Different Policies



(a)



(b)

Model

- Agent wanders around environment
  - observing various inputs
  - performing various actions

  ...occasionally gets reward

- Challenge: Figure out what action to take, in each situation.

- Useful to know **utility** of each state

  ...can then avoid bad states, etc

  But utility not known initially!

- **Reinforcement learning** addresses this!

Markov Decision Processes

Assume:
- finite set of states S
- finite set of actions A

At each discrete time, agent . . .
observes state $s_t \in S$, and
chooses action $a_t \in A$

then receives (immediate) reward $r_t = R(s_t)$
and state changes to $s_{t+1} \sim P(S_{t+1} \mid s_t, a_t)$

Markov assumption:

r_t, s_{t+1} depend only on *current* $\left\{ \begin{array}{c} \text{state(s)} \\ \text{action} \end{array} \right.$

Use of $\boxed{M^a_{ij}}$ suggests
(time-independent) Markovian

Agent's Learning Task

- Execute actions in environment, observe results, and ...
  - learn action policy $\pi : S \rightarrow A$ that maximizes
    $$E[r_t + r_{t+1} + r_{t+2} + \ldots]$$
    from any starting state in S
  - Or...
    $$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots]$$
    $0 \leq \gamma < 1$ is "discount factor for future rewards"
    (relative importance of short-term vs long-term rewards)
    Assuming "accessible" (so know state)

Note:
- Target function is $\pi : S \rightarrow A$
- but training examples NOT of form $\langle s, a^* \rangle$
  (a^* = optimal action)
- instead: $\langle \langle s, a \rangle,\ r \rangle$

Defining Utility Function Over States

$U_\pi(S_i) =$ expected cumulative reward of executing policy π, starting in state S_i

(aka "*value function*")

$$\begin{aligned} U_\pi(S_i) &= R(S_i) + \sum_j P(S_j \mid S_i, \pi(S_i)) \cdot U_\pi(S_j) \\ &= R(S_i) + \sum_j M_{ij}^{\pi(S_i)} \cdot U_\pi(S_j) \end{aligned}$$

Key Point: Can use $U_\pi(S_j)$ as observe S_j after taking action $\pi(S_i)$

Theorem (Bellman and Dreyfus) [MEU]

Optimal policy $\pi^*(S)$ = action that maximizes expected utility $U^*(\cdot)$

$$\begin{aligned} \pi^*(S_i) &= \operatorname*{argmax}_a \sum_j M_{ij}^a \; U^*(S_j) \\ U^*(S_i) &= R(S_i) + \max_a \sum_j M_{ij}^a \; U^*(S_j) \end{aligned}$$

Challenges

Goal: Computing *optimal* policy

$$\pi^* = \underset{\pi}{\text{argmax}}\, E_s[\ U_\pi(s)\ ]$$

Challenge#1: Finding optimal policy given model $[P(s' \mid s, a),\ r(s)]$

- ◦ Value Iteration
- ◦ Policy Iteration
  - ⇒ Challenge#1A: Evaluating *fixed* policy
  - – Adaptive Dynamic Programming
  - – Sampling
  - • TD-learning; TD(λ)-learning

Challenge#2: Scaling to large spaces

- • Generalization

Challenge#3: Finding optimal policy *NOT* given model $[P(s' \mid s, a),\ R(s)]$

- ◦ Estimation + "Challenge#1"
- • Q-learning

Finding Optimal Policy

- Easy to find optimal policy π^*, given $U^*(s_i)$

$$\pi^*(s_i) \quad = \quad \underset{a}{\text{argmax}} \sum_j M_{ij}^a \; U^*(s_j)$$

- Easy to find $U^*(s_i)$, given optimal policy π^*:

$$U^*(s_i) \quad = \quad R(\, s_i \,) + \sum_j M^{\pi^*(s_i)} \; U^*(s_j)$$

- Circular:

  Given U^*, can find π^*

  But need π^* to find U^* values

  as need to know $\pi^*(s_j)$ to determine $U^*(s_j)$

  ...to know $U^*(s_i)$

- Answers ...

Value Iteration: Algorithm for computing U^*

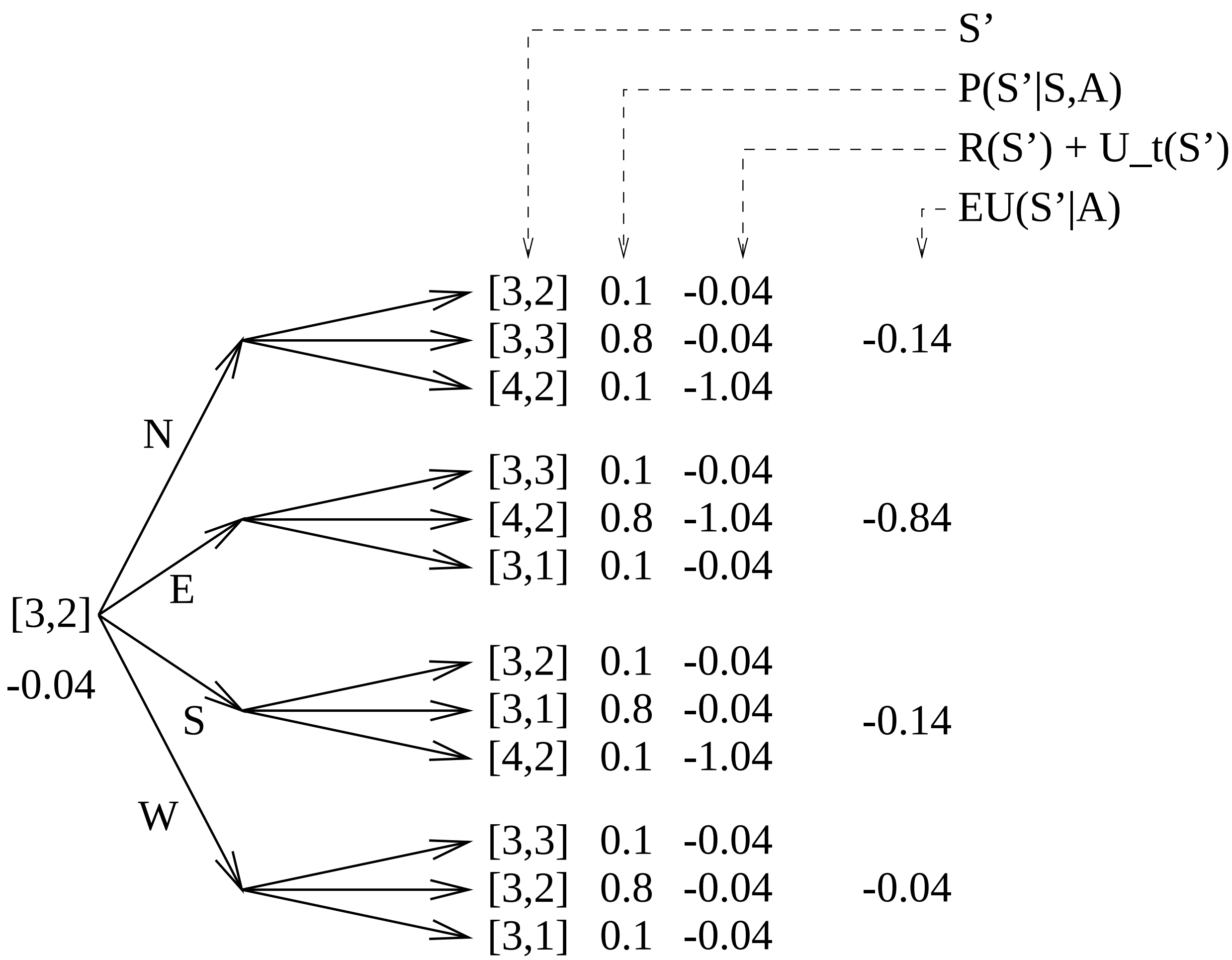
function *ValueIteration*$(\cdots)$
$U(S)$ `:=` $R(S)$ `for all` S
while U `is changing` **do**
for `each state` s_i **do**

$$U(s_i) := R(s_i) + \max_a \sum_j M_{ij}^a \cdot U(s_j)$$

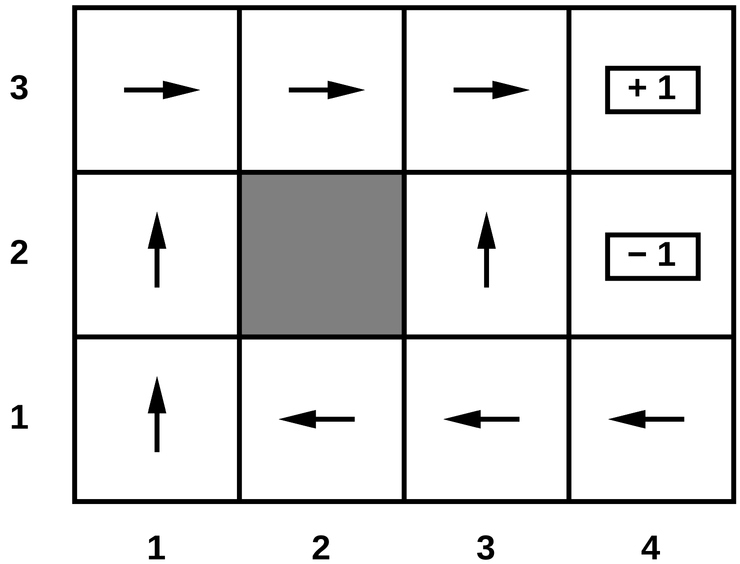
end
end
return($U(\cdot)$)

- $U(\cdot)$ converges to stable values
- Each update of U is called *Bellman backup*

Example of Bellman Backup

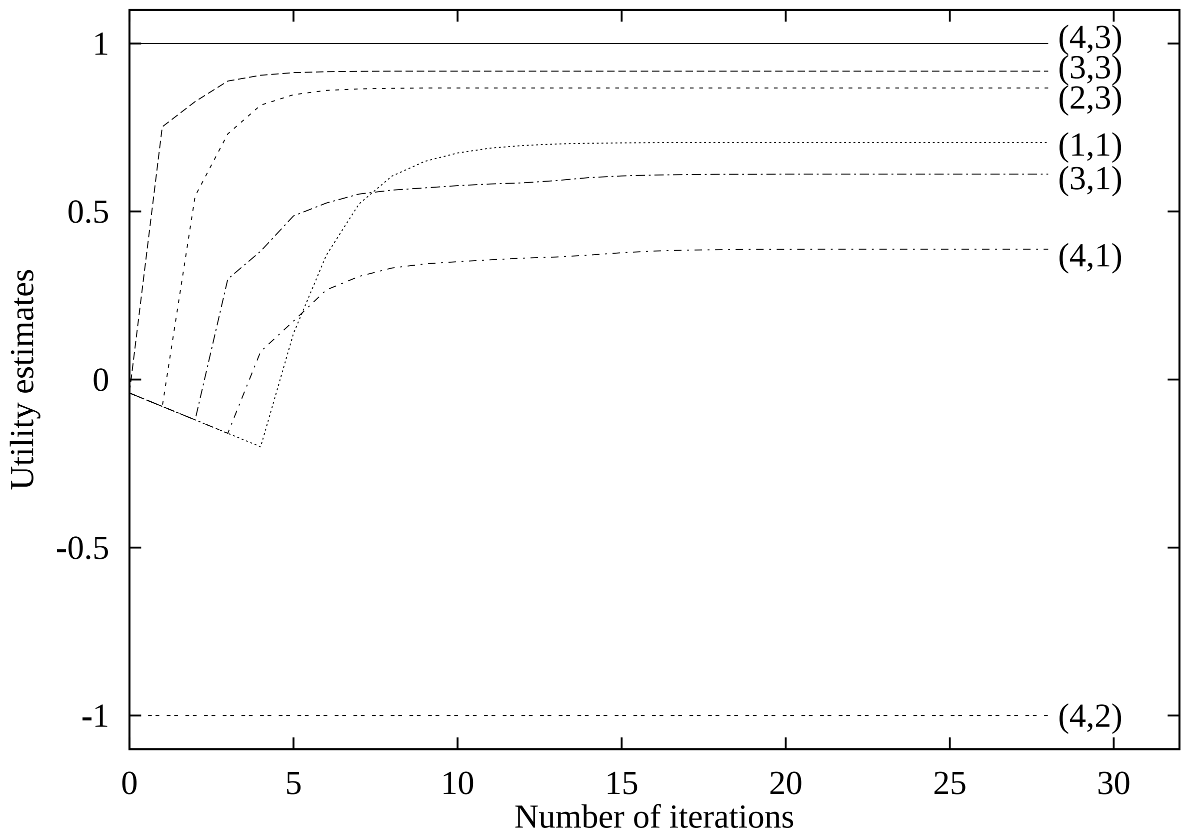


Optimal Value Function and Policy



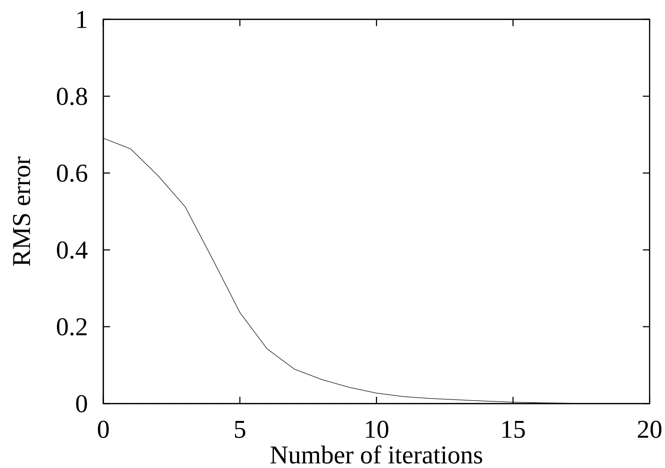
	1	2	3	4
3	0.812	0.868	0.912	+ 1
2	0.762		0.660	− 1
1	0.705	0.655	0.611	0.388

Behavior of Value Iteration

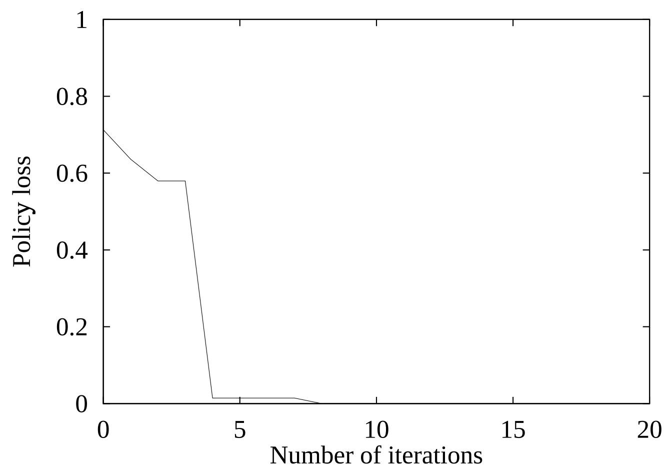


Utility value for selected states
at each iteration step in *ValueIteration*

Convergence of Value Iteration



$$RMS(\widehat{U}_k) = \frac{1}{|S|}\sum_{S}\left(U^*(S) - \widehat{U}_k(S) \right)^2$$



$$PolicyLoss(\widehat{\pi}_k) = \sum_{s_j \in Terminal} U(s_j) \cdot [\, P(\pi^* \text{ leads } s_0 \text{ to } s_j) - P(\widehat{\pi}_k \text{ leads } s_0 \text{ to } s_j)\,]^2$$

Policy Iteration

Note: Policy $\hat{\pi}$ not particularly sensitive to utility estimate $\hat{U}$

- Why not just estimate *Policy*, on each iteration?

function *PolicyIteration*$(R(\cdot),\ M^a_{ij})$

```
Choose initial policy π_0;   t := 0
Repeat
        % Value Determination
  Compute utility U_{π_t}(s),   ∀s
        % Policy Improvement
  Compute new policy
    ∀s_i:  π_{t+1}(s_i) := argmax_a Σ_j M^a_ij U_{π_t}(s_j)
until convergence  (π_{t+1} ≈ π_t)
return( π_t )
```

$$\forall s_i:\quad \pi_{t+1}(s_i)\ :=\ \underset{a}{\operatorname{argmax}} \sum_j M^a_{ij}\ U_{\pi_t}(s_j)$$

Implementing `ValueDetermination`

- Evaluating Fixed Policy

  $$\equiv$$

  Finding $\{U(s)\}$ s.t.

  $$U(s_i) \;=\; R(\,s_i\,) + \sum_j M_{ij}^{\pi(s_i)} \cdot U(s_j)$$

  (in known accessible environment)

- Just solve set of equations:

$$\begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_n \end{pmatrix} = \begin{pmatrix} R_1 \\ R_2 \\ \vdots \\ R_n \end{pmatrix} + \begin{pmatrix} M_{11}^{\pi(s_1)} & M_{12}^{\pi(s_1)} & \dots & M_{1n}^{\pi(s_1)} \\ M_{21}^{\pi(s_2)} & M_{22}^{\pi(s_2)} & \dots & M_{2n}^{\pi(s_2)} \\ \vdots & \vdots & \ddots & \vdots \\ M_{n1}^{\pi(s_n)} & M_{n2}^{\pi(s_n)} & \dots & M_{nn}^{\pi(s_n)} \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_n \end{pmatrix}$$

$$\vec{U} = -(M - I)^{-1}\vec{R}$$

(Note: most $M_{ij}^a = 0$)

but . . . too many equations/unknowns!

Eg, for backgammon:

$\approx 10^{50}$ equations w/ $\approx 10^{50}$ unknowns!

Infeasibly large!

The Curse of Dimensionality

- Computational cost scales with
  - ⋆ number of states $|S|$
  - ⋆ number of actions $|A|$

- $|S|$ is *exponential* in
  number of "dimensions" ($\approx$ sensors)

- "Curse of Dimensionality" [Bellman]
  - **Value Iteration**: $O(\, n\,|S|\,|A|\,B\,)$
  - **Policy Iteration**: $O(\, n'\,|S|\,|A|\,)$
  - **Value Determination**: $O(\, n''\,|S|\,B)$
    or $O(\,|S|^3)$

(n, n', n'' is number of iterations;
B is time for each Bellman Update)

Simple Extensions

- So far,
  - $R(\, s\,)$
    (reward just depends on current state)
  - $P(\, s_{t+1} \,|\, s_t, a_t\,)$
    (transition probability independent of WHEN)
  - $\sum_{s_t} R_{s_t}$
    (total cost just sum of rewards)
- Easy extensions to . . .
  - $R(\, s_{t+1} \,|\, s_t, a_t\,)$
    (reward also depends on previous state, action)
  - $P^q(\, s_{t+1} \,|\, s_t, a_t\,)$
    (transition probability depends of time of transition)
  - $\sum_{s_t} \gamma^t R_{s_t}$
    (total cost is discounted sum of rewards; $0 < \gamma < 1$)