Reinforcement Learning

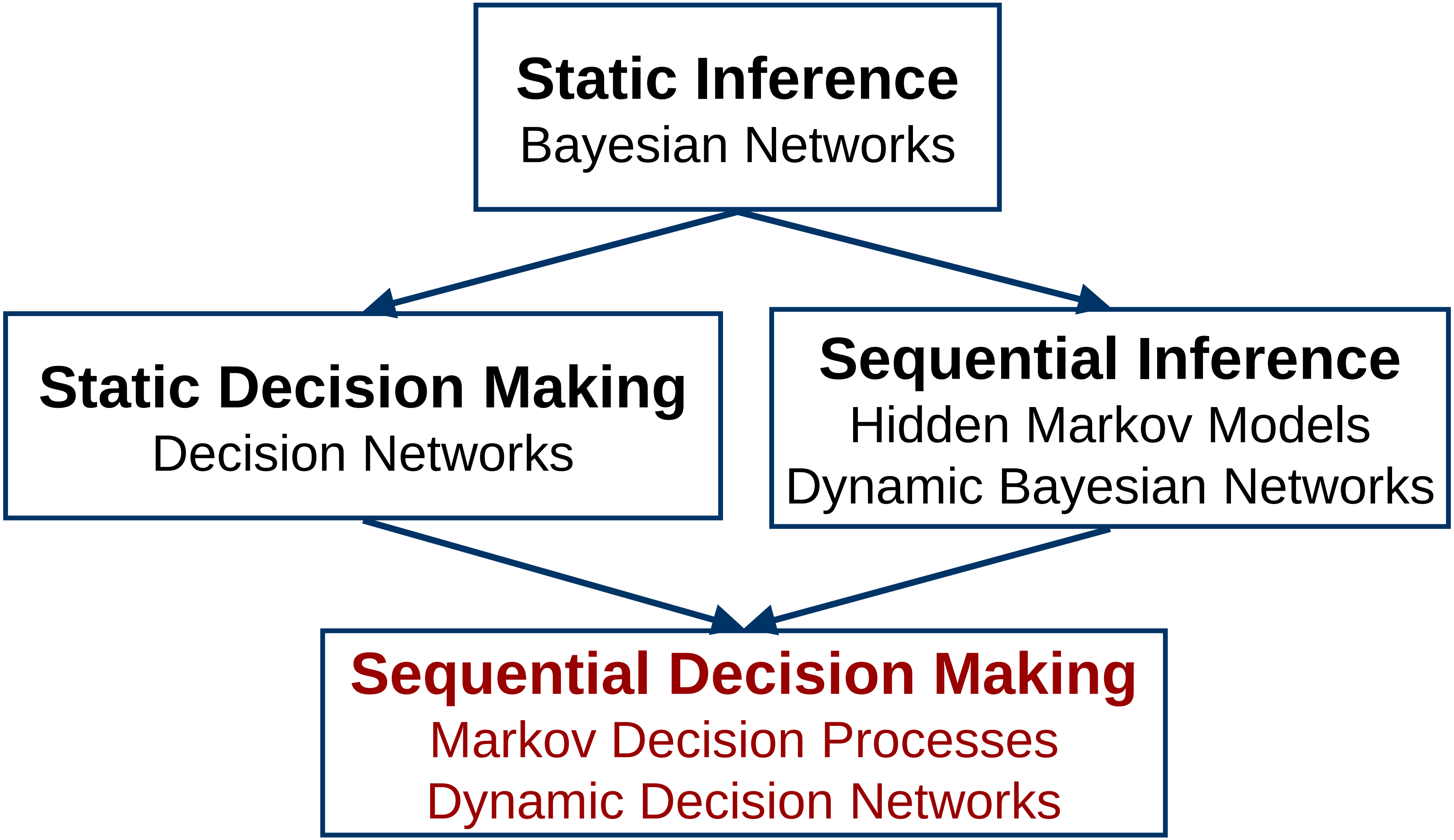
November 23, 2010

CS 886

Outline

- Markov Decision Processes
  - Dynamic Decision Networks
  - Russell and Norvig: Sect 17.1, 17.2 (up to p. 620), 17.4, 17.5
- Reinforcement learning
  - Temporal-Difference learning
  - Q-learning
  - Russell & Norvig Sect 21.1-21.3

Sequential Decision Making



Static Inference
Bayesian Networks

Static Decision Making
Decision Networks

Sequential Inference
Hidden Markov Models
Dynamic Bayesian Networks

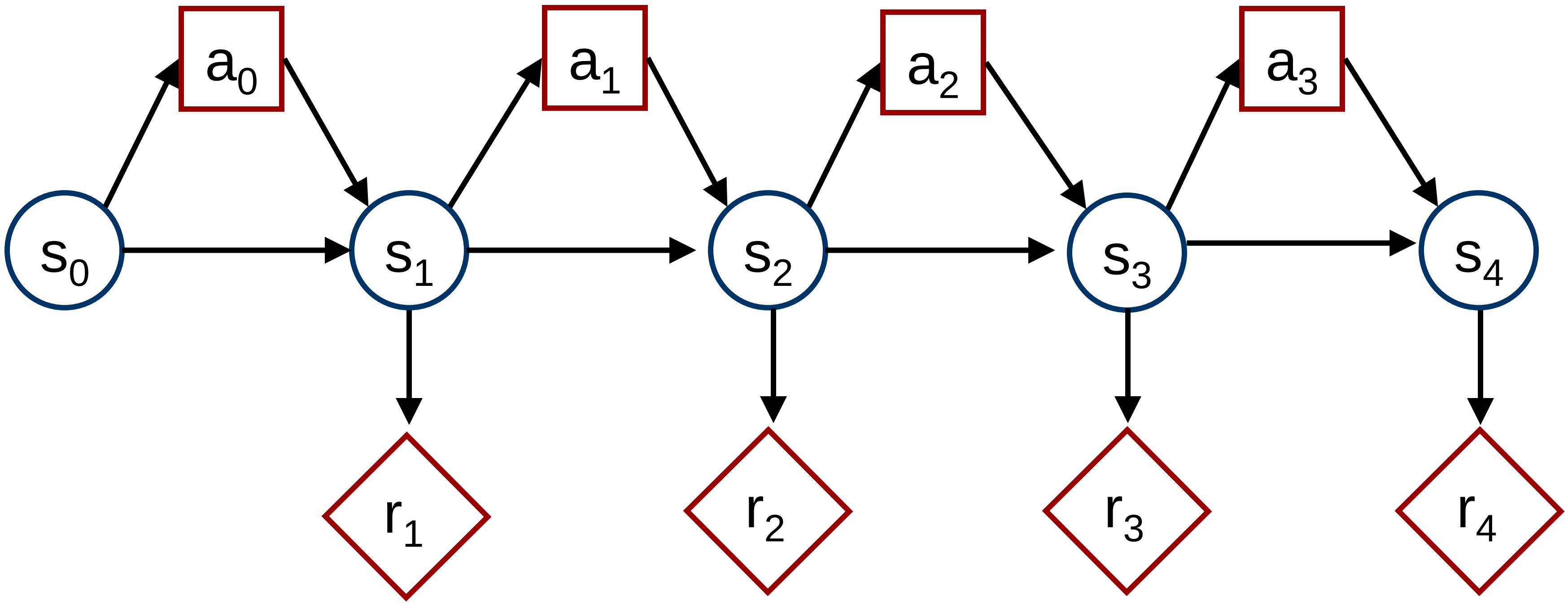
Sequential Decision Making
Markov Decision Processes
Dynamic Decision Networks

Sequential Decision Making

- Wide range of applications
  - Robotics (e.g., control)
  - Investments (e.g., portfolio management)
  - Computational linguistics (e.g., dialogue management)
  - Operations research (e.g., inventory management, resource allocation, call admission control)
  - Assistive technologies (e.g., patient monitoring and support)

Markov Decision Process

- Intuition: Markov Process with…
  - Decision nodes
  - Utility nodes



Stationary Preferences

- Hum… but why many utility nodes?

- $U(s_0,s_1,s_2,\ldots)$
  - Infinite process → infinite utility function

- Solution:
  - Assume stationary and additive preferences
  - $U(s_0,s_1,s_2,\ldots) = \Sigma_t R(s_t)$

Discounted/Average Rewards

- If process infinite, isn't Σ_t R(s_t) infinite?

- Solution 1: discounted rewards
  - Discount factor: 0 ≤ γ ≤ 1
  - Finite utility: Σ_t γ^tR(s_t) is a geometric sum
  - γ is like an inflation rate of 1/γ - 1
  - Intuition: prefer utility sooner than later

- Solution 2: average rewards
  - More complicated computationally
  - Beyond the scope of this course

Markov Decision Process

- Definition
  - Set of states: S
  - Set of actions (i.e., decisions): A
  - Transition model: $\Pr(s_t|a_{t-1},s_{t-1})$
  - Reward model (i.e., utility): $R(s_t)$
  - Discount factor: $0 \leq \gamma \leq 1$
  - Horizon (i.e., # of time steps): h

- Goal: find optimal policy

Inventory Management

- Markov Decision Process
  - States: inventory levels
  - Actions: {doNothing, orderWidgets}
  - Transition model: stochastic demand
  - Reward model: Sales – Costs - Storage
  - Discount factor: 0.999
  - Horizon: ∞
- Tradeoff: increasing supplies decreases odds of missed sales but increases storage costs

Policy

- Choice of action at each time step
- Formally:
  - Mapping from states to actions
  - i.e., $\delta(s_t) = a_t$
  - Assumption: fully observable states
    - Allows a_t to be chosen only based on current state s_t. Why?

Policy Optimization

- Policy evaluation:
  - Compute expected utility
  - $EU(\delta) = \Sigma_{t=0}^{h} \gamma^{t} Pr(s_t|\delta) R(s_t)$
- Optimal policy:
  - Policy with highest expected utility
  - $EU(\delta) \leq EU(\delta^*)$ for all δ

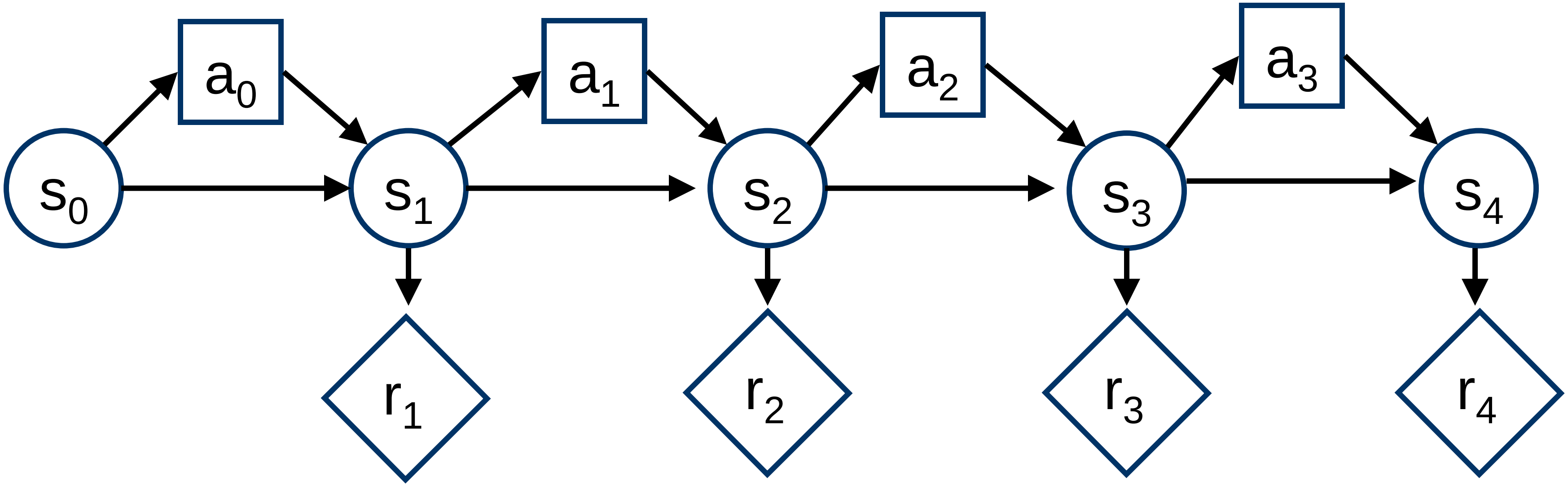
Policy Optimization

- Three algorithms to optimize policy:
  - Value iteration
  - Policy iteration
  - Linear Programming

- Value iteration:
  - Equivalent to variable elimination

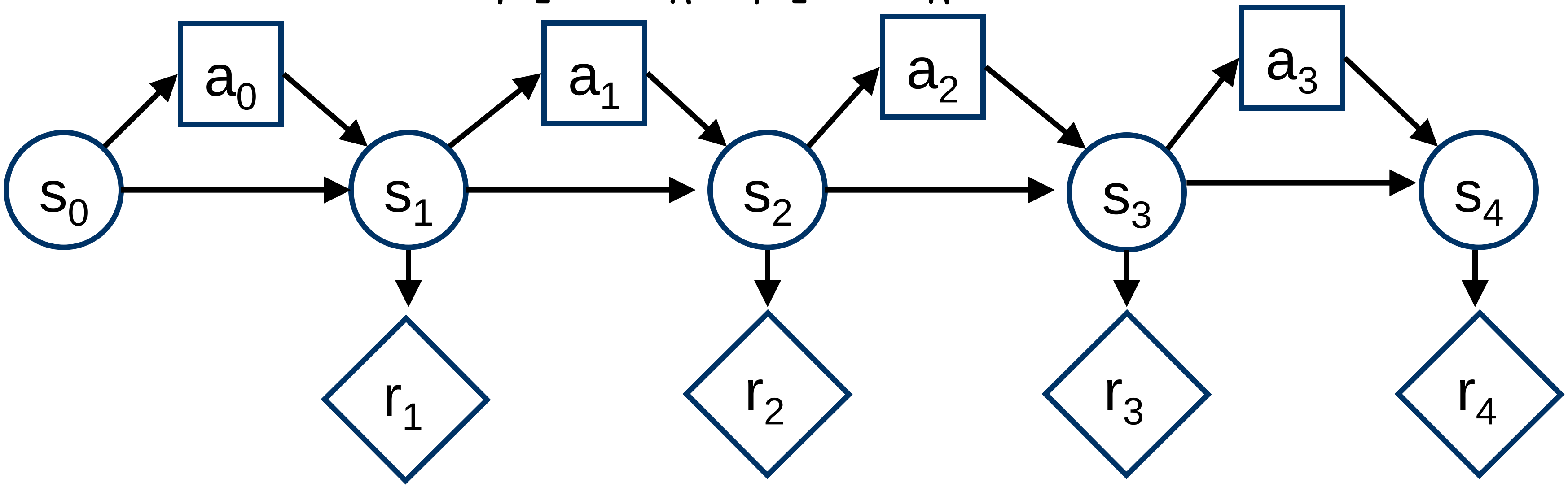
Value Iteration

- Nothing more than variable elimination
- Performs dynamic programming
- Optimize decisions in reverse order



Value Iteration

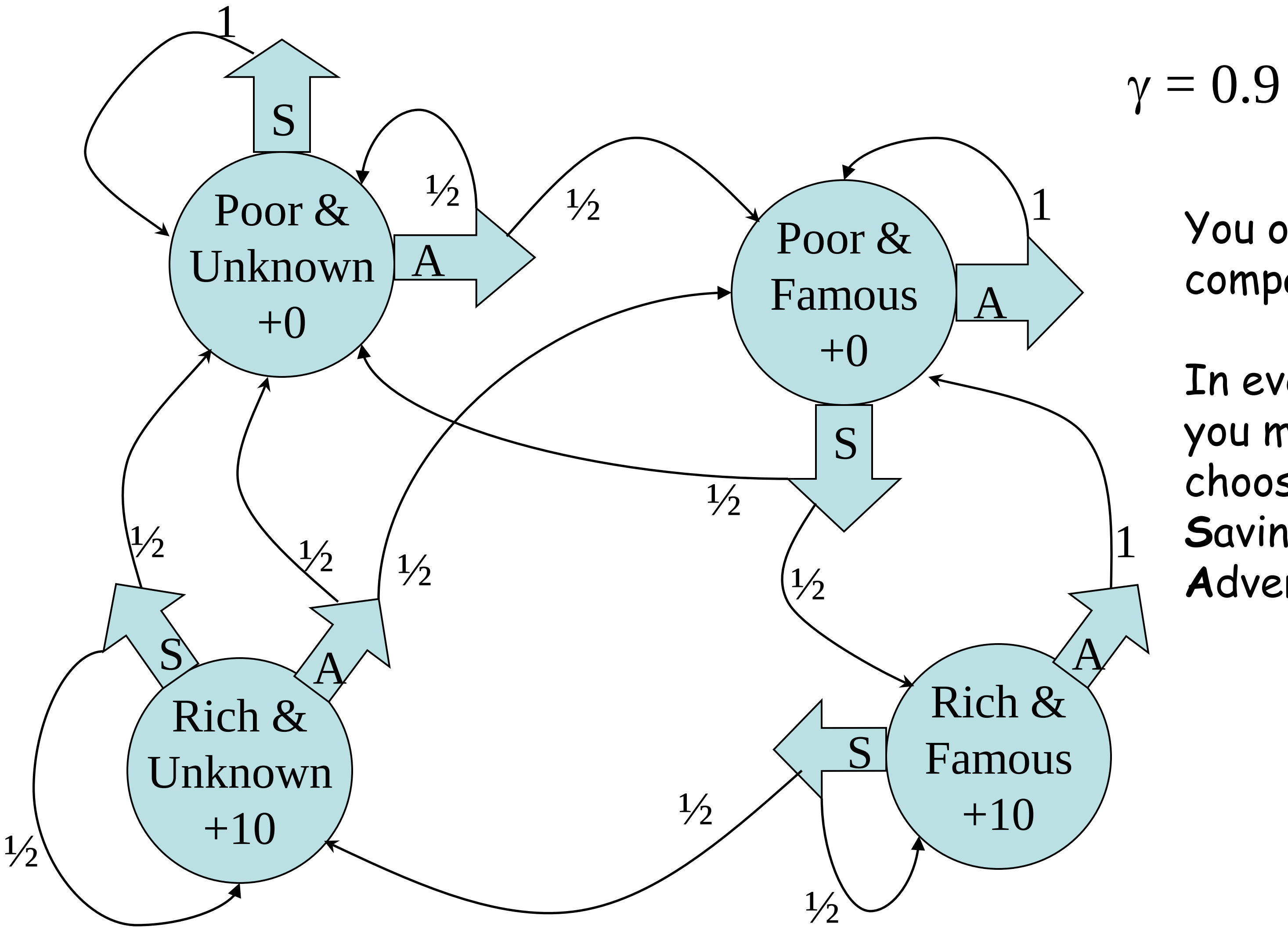
- At each t, starting from t=h down to 0:
  - Optimize a_t: $EU(a_t|s_t)$?
  - Factors: $Pr(s_{i+1}|a_i,s_i)$, $R(s_i)$, for $0 \leq i \leq h$
  - Restrict s_t
  - Eliminate $s_{t+1},\dots,s_h,a_{t+1},\dots,a_h$



Value Iteration

- Value when no time left:
  - $V(s_h) = R(s_h)$
- Value with one time step left:
  - $V(s_{h-1}) = \max_{a_{h-1}} R(s_{h-1}) + \gamma \Sigma_{s_h} Pr(s_h|s_{h-1},a_{h-1}) V(s_h)$
- Value with two time steps left:
  - $V(s_{h-2}) = \max_{a_{h-2}} R(s_{h-2}) + \gamma \Sigma_{s_{h-1}} Pr(s_{h-1}|s_{h-2},a_{h-2})V(s_{h-1})$
- …
- Bellman's equation:
  - $V(s_t) = \max_{a_t} R(s_t) + \gamma \Sigma_{s_{t+1}} Pr(s_{t+1}|s_t,a_t) V(s_{t+1})$
  - $a_t^* = \text{argmax}_{a_t} R(s_t) + \gamma \Sigma_{s_{t+1}} Pr(s_{t+1}|s_t,a_t) V(s_{t+1})$

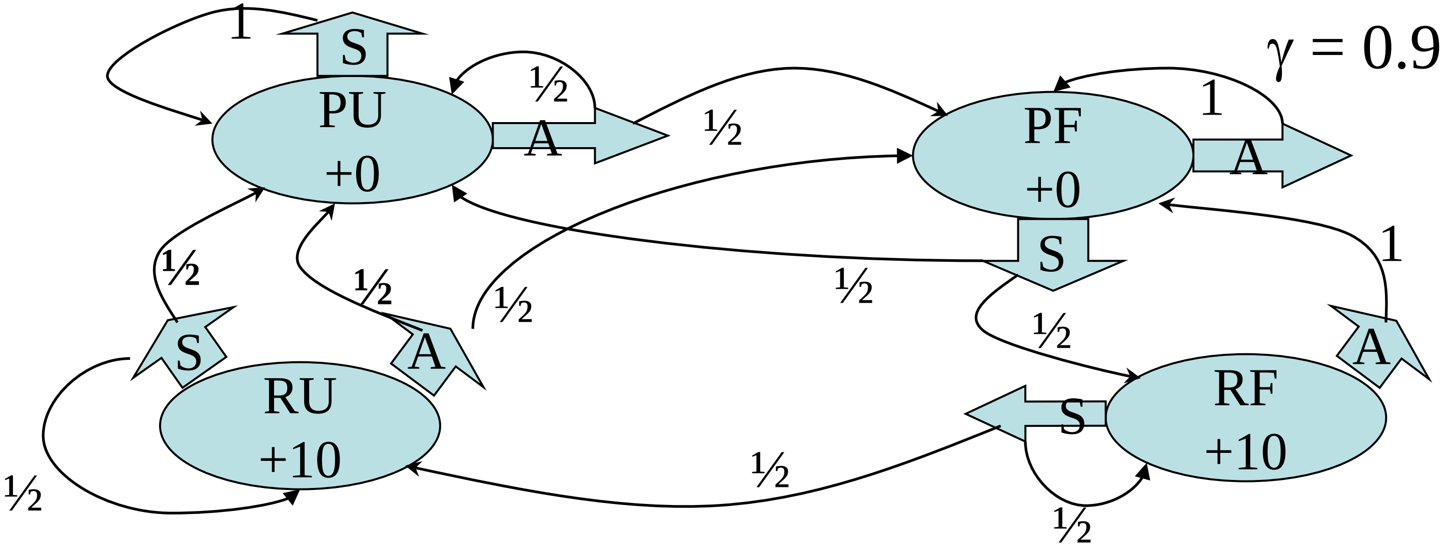
A Markov Decision Process



γ = 0.9

You own a company

In every state you must choose between **S**aving money or **A**dvertising



t	V(PU)	V(PF)	V(RU)	V(RF)
h	0	0	10	10
h-1	0	4.5	14.5	19
h-2	2.03	8.55	16.53	25.08
h-3	4.76	12.20	18.35	28.72
h-4	7.63	15.07	20.40	31.18
h-5	10.21	17.46	22.61	33.21

Finite Horizon

- When h is finite,
- Non-stationary optimal policy
- Best action different at each time step
- Intuition: best action varies with the amount of time left

Infinite Horizon

- When h is infinite,
- Stationary optimal policy
- Same best action at each time step
- Intuition: same (infinite) amount of time left at each time step, hence same best action

- Problem: value iteration does an infinite number of iterations…

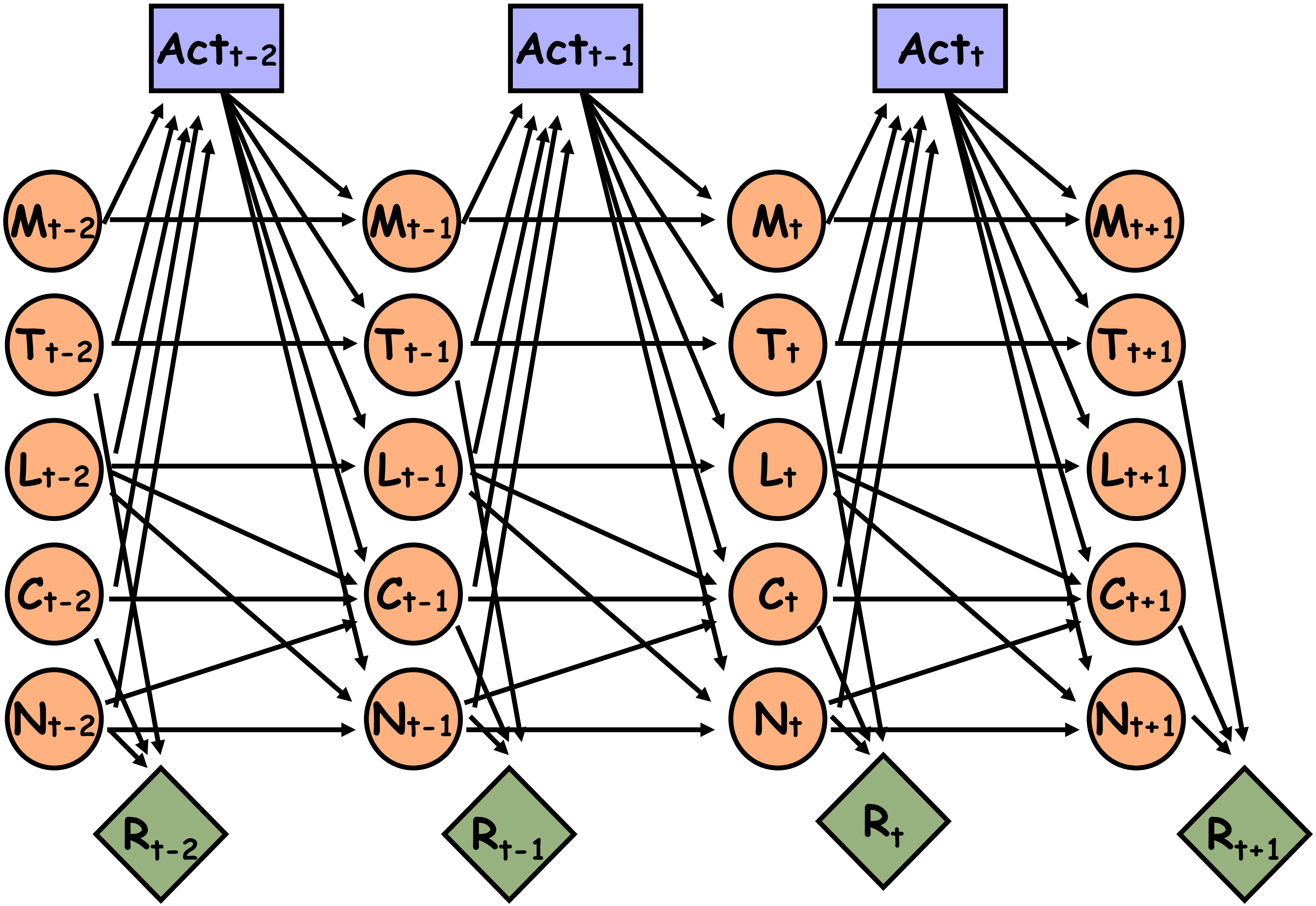
Infinite Horizon

- Assuming a discount factor γ, after k time steps, rewards are scaled down by γ^k
- For large enough k, rewards become insignificant since $\gamma^k \rightarrow 0$
- Solution:
  - pick large enough k
  - run value iteration for k steps
  - Execute policy found at the k^{th} iteration

Computational Complexity

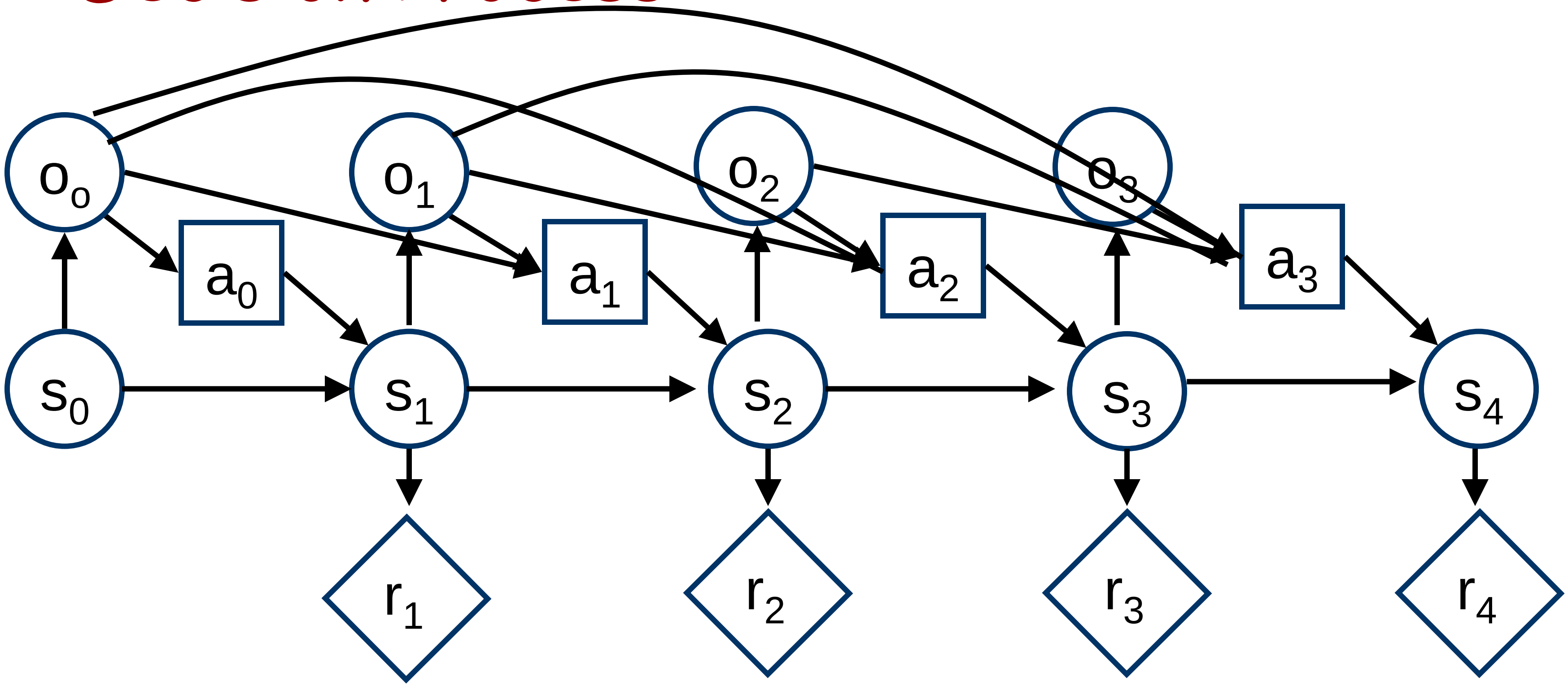
- Space and time: $O(k|A||S|^2)$ ☺
  - Here k is the number of iterations
- But what if |A| and |S| are defined by several random variables and consequently exponential?
- Solution: exploit conditional independence
  - Dynamic decision network

Dynamic Decision Network



Partial Observability

- What if states are not fully observable?
- Solution: Partially Observable Markov Decision Process



Partially Observable Markov Decision Process (POMDP)

- Definition
  - Set of states: S
  - Set of actions (i.e., decisions): A
  - Set of observations: O
  - Transition model: $\Pr(s_t|a_{t-1},s_{t-1})$
  - Observation model: $\Pr(o_t|s_t)$
  - Reward model (i.e., utility): $R(s_t)$
  - Discount factor: $0 \leq \gamma \leq 1$
  - Horizon (i.e., # of time steps): h
- Policy: mapping from past obs. to actions

POMDP

- Problem: action choice generally depends on all previous observations…
- Two solutions:
  - Consider only policies that depend on a finite history of observations
  - Find stationary sufficient statistics encoding relevant past observations

Reinforcement Learning

- Definition:
  - Markov decision process with unknown transition and reward models

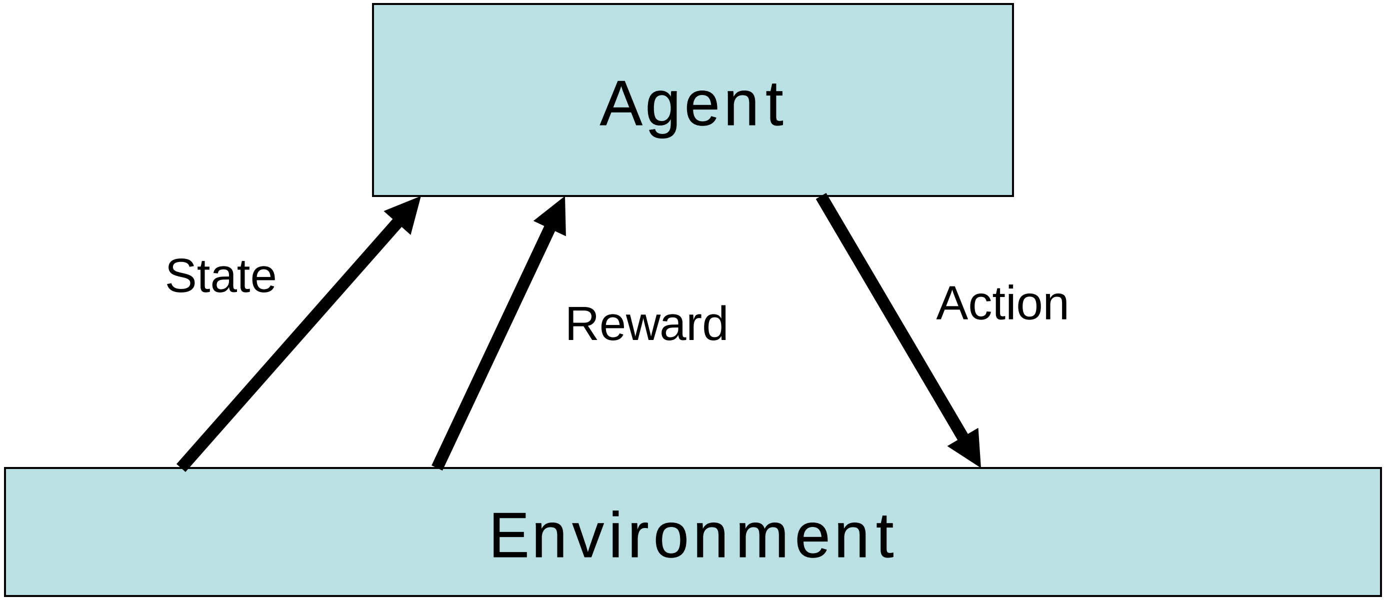
- Set of states S
- Set of actions A
  - Actions may be stochastic
- Set of reinforcement signals (rewards)
  - Rewards may be delayed

Policy optimization

- Markov Decision Process:
  - Find optimal policy given transition and reward model
  - Execute policy found

- Reinforcement learning:
  - Learn an optimal policy while interacting with the environment

Reinforcement Learning Problem

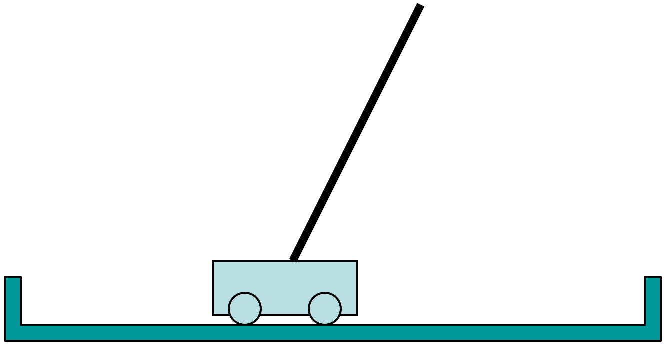


$$\begin{tikzcd} s0 \arrow[r, "a0", "r0"'] & s1 \arrow[r, "a1", "r1"'] & s2 \arrow[r, "a2", "r2"'] & \ldots \end{tikzcd}$$

Goal: Learn to choose actions that maximize $r_0+\gamma\, r_1+\gamma^2 r_2+\ldots$, where $0\leq \gamma <1$

Example: Inverted Pendulum

- State: $x(t)$, $x'(t)$, $\theta(t)$, $\theta'(t)$
- Action: Force F
- Reward: 1 for any step where pole balanced



Problem: Find $\delta: S \rightarrow A$ that maximizes rewards

RL Examples

- Game playing (backgammon, solitaire)
- Operations research (pricing, vehicule routing)
- Elevator scheduling
- Helicopter control

- http://neuromancer.eecs.umich.edu/cgi-bin/twiki/view/Main/SuccessesOfRL

Types of RL

- Passive vs Active learning
  - Passive learning: the agent executes a fixed policy and tries to evaluate it
  - Active learning: the agent updates its policy as it learns

- Model based vs model free
  - Model-based: learn transition and reward model and use it to determine optimal policy
  - Model free: derive optimal policy without learning the model

Passive Learning

- Transition and reward model known:
  - Evaluate δ:
  - $V^{\delta}(s) = R(s) + \gamma \Sigma_{s'} Pr(s'|s,\delta(s)) V^{\delta}(s')$
- Transition and reward model unknown:
  - Estimate policy value as agent executes policy: $V^{\delta}(s) = E_{\delta}[\Sigma_t \gamma^t R(s_t)]$
  - Model based vs model free

Passive learning

3	r	r	r	+1
2	u	■	u	-1
1	u	l	l	l
	1	2	3	4

$\gamma = 1$

r_i = -0.04 for non-terminal states

Do not know the transition probabilities

(1,1)→ (1,2)→ (1,3)→ (1,2)→ (1,3)→ (2,3)→ (3,3)→ $(4,3)_{+1}$
(1,1)→ (1,2)→ (1,3)→ (2,3)→ (3,3)→ (3,2)→ (3,3)→ $(4,3)_{+1}$
(1,1)→ (2,1)→ (3,1)→ (3,2)→ $(4,2)_{-1}$

What is the value V(s) of being in state s?

Passive ADP

- Adaptive dynamic programming (ADP)
    - Model-based
    - Learn transition probabilities and rewards from observations
    - Then update the values of the states

ADP Example

$\gamma = 1$

3	r	r	r	+1
2	u	■	u	-1
1	u	l	l	l
	1	2	3	4

r_i = -0.04 for non-terminal states

$$V^{\delta}(s) = R(s) + \gamma\, \Sigma_{s'} \Pr(s'|s,\delta(s))\, V^{\delta}(s')$$

(1,1)→ (1,2)→ **(1,3)**→ (1,2)→ **(1,3)**→ (2,3)→ (3,3)→ $(4,3)_{+1}$

(1,1)→ (1,2)→ **(1,3)**→ (2,3)→ (3,3)→ (3,2)→ (3,3)→ $(4,3)_{+1}$

(1,1)→ (2,1)→ (3,1)→ (3,2)→ $(4,2)_{-1}$

$$\left.\begin{array}{l} P((2,3)|(1,3),r) = 2/3 \\ P((1,2)|(1,3),r) = 1/3 \end{array}\right\}$$ Use this information in

We need to learn all the transition probabilities!

Passive TD

- Temporal difference (TD)
  - Model free

- At each time step
  - Observe: s,a,s',r
  - Update $V^\delta(s)$ after each move
  - $V^\delta(s) = V^\delta(s) + \underset{\text{Learning rate}}{\alpha}\ \underbrace{(R(s) + \gamma\ V^\delta(s') - V^\delta(s))}_{\text{Temporal difference}}$

TD Convergence

Thm: If α is appropriately decreased with number of times a state is visited then $V^{\delta}(s)$ converges to correct value

- α must satisfy:
  - $\Sigma_t \alpha_t \rightarrow \infty$
  - $\Sigma_t (\alpha_t)^2 < \infty$
- Often $\alpha(s) = 1/n(s)$
  - $n(s)$ = # of times s is visited

Active Learning

- Ultimately, we are interested in improving δ
- Transition and reward model known:
  - $V^*(s) = \max_a R(s) + \gamma \Sigma_{s'} Pr(s'|s,a) V^*(s')$
- Transition and reward model unknown:
  - Improve policy as agent executes policy
  - Model based vs model free

Q-learning (aka active temporal difference)

- Q-function: $Q{:}S \times A \rightarrow \Re$
  - Value of state-action pair
  - Policy $\delta(s) = \mathrm{argmax}_a\, Q(s,a)$ is the optimal policy

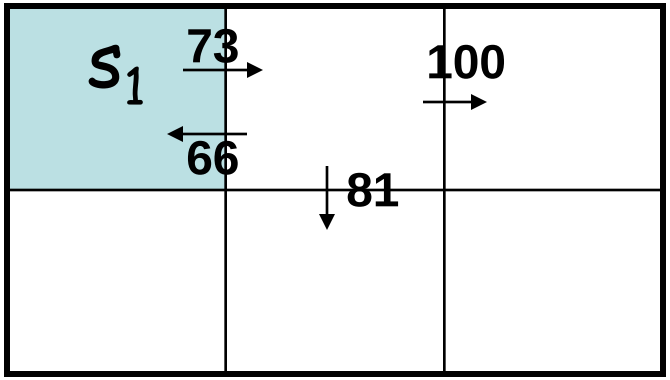
- Bellman's equation:

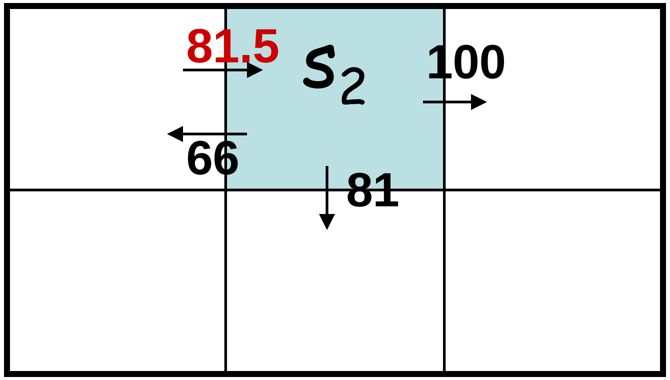
$$\boxed{Q^*(s,a) = R(s) + \gamma\, \Sigma_{s'} \Pr(s'|s,a) \max_{a'} Q^*(s',a')}$$

Q-learning

- For each state s and action a initialize Q(s,a) (0 or random)
- Observe current state
- Loop
  - Select action a and execute it
  - Receive immediate reward r
  - Observe new state s'
  - Update Q(a,s)
    - $Q(s,a) = Q(s,a) + \alpha(r(s)+\gamma \max_{a'} Q(s',a') - Q(s,a))$
  - s=s'

Q-learning example





r=0 for non-terminal states
γ=0.9
α=0.5

$$
\begin{aligned}
Q(s_1,\text{right}) &= Q(s_1,\text{right}) + \alpha\ (r(s_1) + \gamma\ \max_{a'} Q(s_2,a') - Q(s_1,\text{right})) \\
&= 73 + 0.5\ (0 + 0.9\ \max[66,81,100] - 73) \\
&= 73 + 0.5\ (17) \\
&= 81.5
\end{aligned}
$$

Q-learning

- For each state s and action a initialize Q(s,a) (0 or random)
- Observe current state
- Loop
  - Select action a and execute it
  - Receive immediate reward r
  - Observe new state s'
  - Update Q(a,s)
    - $Q(s,a) = Q(s,a) + \alpha(r(s)+\gamma \max_{a'} Q(s',a') - Q(s,a))$
  - s=s'

Exploration vs Exploitation

- If an agent always chooses the action with the highest value then it is exploiting
  - The learned model is not the real model
  - Leads to suboptimal results
- By taking random actions (pure exploration) an agent may learn the model
  - But what is the use of learning a complete model if parts of it are never used?
- Need a balance between exploitation and exporation

Common exploration methods

- ε-greedy:
  - With probability ε execute random action
  - Otherwise execute best action a*
    $a^* = \text{argmax}_a\, Q(s,a)$

- Boltzmann exploration

$$P(a) = \frac{e^{Q(s,a)/T}}{\Sigma_a e^{Q(s,a)/T}}$$

Exploration and Q-learning

- Q-learning converges to optimal Q-values if
  - Every state is visited infinitely often (due to exploration)
  - The action selection becomes greedy as time approaches infinity
  - The learning rate a is decreased fast enough but not too fast

A Triumph for Reinforcement Learning: TD-Gammon

- Backgammon player: TD learning with a neural network representation of the value function:

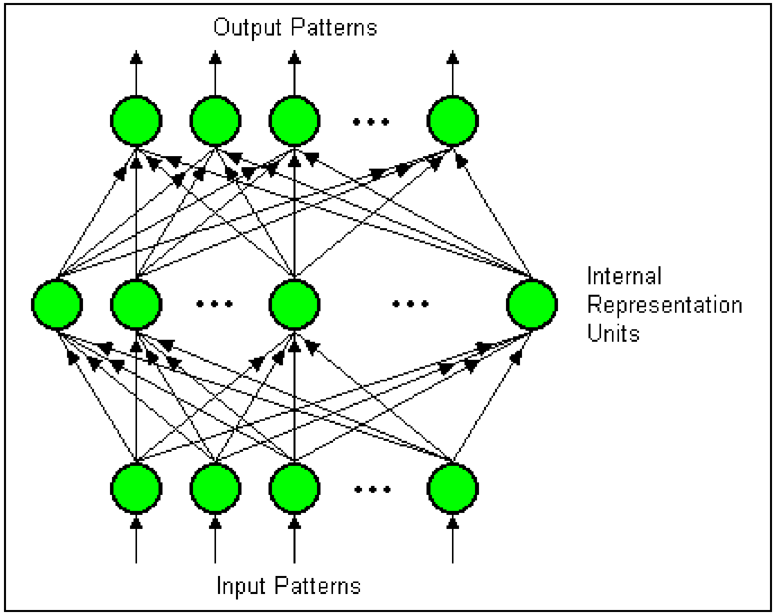


Figure 1. An illustration of the multilayer perception architecture used in TD-Gammon's neural network. This architecture is also used in the popular backpropagation learning procedure. Figure reproduced from [9].