Reinforcement Learning

Yishay Mansour

MSR & Tel-Aviv University

Outline

- Goal of Reinforcement Learning
- Mathematical Model (MDP)
- Planning
- Learning
- Large MDP
- POMDP
- Lets see how much we can really cover ☺

Reinforcement Learning - origins

Artificial Intelligence

Control Theory

Operation Research

Cognitive Science & Psychology

Solid foundations; well established research.

Typical Applications

- Robotics
  - Helicopter control [NKJS].
  - Robo-soccer [SV].
- Board games
  - checkers [S].
  - backgammon [T],
  - Go/Atari
- Scheduling
  - Dynamic channel allocation [SB].
  - Inventory problems.

An important modeling tool

Goal of Reinforcement Learning

Goal oriented learning through interaction

Control of large scale stochastic environments with partial knowledge.

Supervised / Unsupervised Learning

Learn from labeled / unlabeled examples

Contrast with Supervised Learning

The system has a “state”.

The algorithm influences the state distribution.

Impact on both rewards and observations

Inherent Tradeoff: Exploration versus Exploitation.

Mathematical Model - Motivation

Model of uncertainty:

Environment, actions, our knowledge.

Focus on decision making.

Maximize long term reward.

Markov Decision Process (MDP)

Mathematical Model - MDP

Markov decision processes

S- set of states

A- set of actions

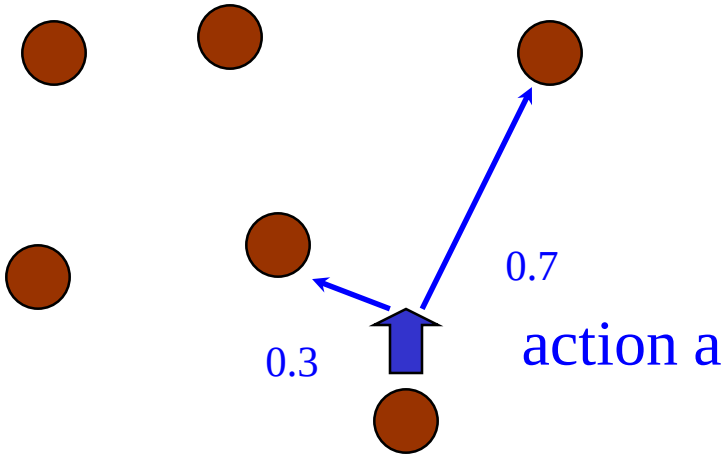
δ - Transition probability

R - Reward function

Similar to DFA!

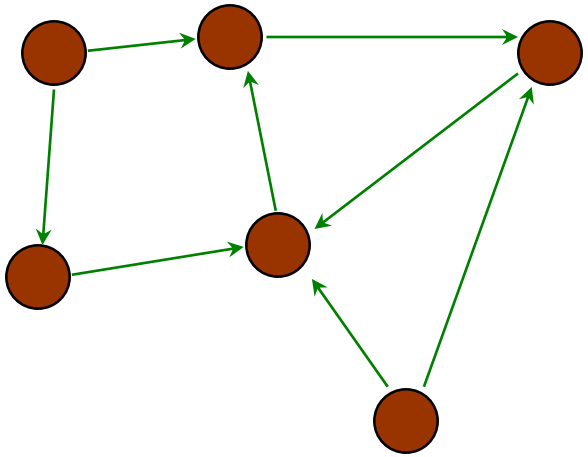
MDP model - states and actions

Environment = states



Actions = transitions $\delta(s, a, s')$

MDP model - rewards



R(s,a) = reward at state ***s***

for doing action ***a***

(a random variable).

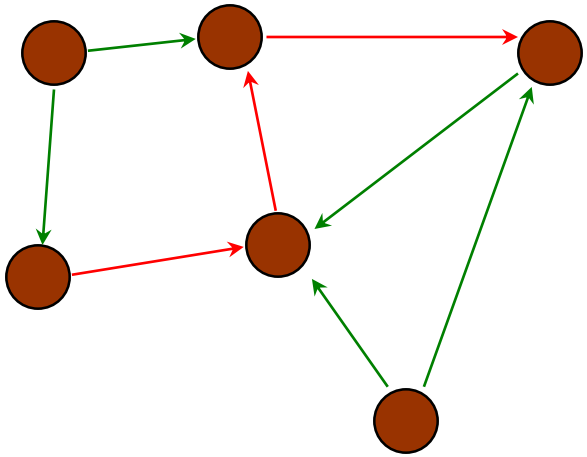
Example:

R(s,a) = -1 with probability 0.5

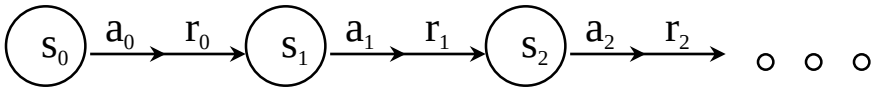
+10 with probability 0.35

+20 with probability 0.15

MDP model - trajectories



trajectory:



MDP - Return function.

Combining all the immediate rewards to a single value.

Modeling Issues:

Are early rewards more valuable than later rewards?

Is the system “terminating” or continuous?

Usually the return is linear in the immediate rewards.

MDP model - return functions

Finite Horizon - parameter H

$$return = \sum_{1 \le i \le H} R(s_i, a_i)$$

Infinite Horizon

discounted - parameter $\gamma<1$.

$$return = \sum_{i=0}^{\infty} \gamma^i R(s_i, a_i)$$

undiscounted

$$\frac{1}{N} \sum_{i=0}^{N-1} R(s_i, a_i) \xrightarrow{N \to \infty} return$$

Terminating MDP

MDP model - action selection

AIM: Maximize the expected return.

Fully Observable - can "see" the "entire" state.

Policy - mapping from states to actions

Optimal policy: optimal from any start state.

THEOREM: There exists an optimal policy which is history-independent and deterministic

MDP model - summary

$s \in S$ - set of states, $|S|=n$.

$a \in A$ - set of k actions, $|A|=k$.

$\delta(s_1, a, s_2)$ - transition function.

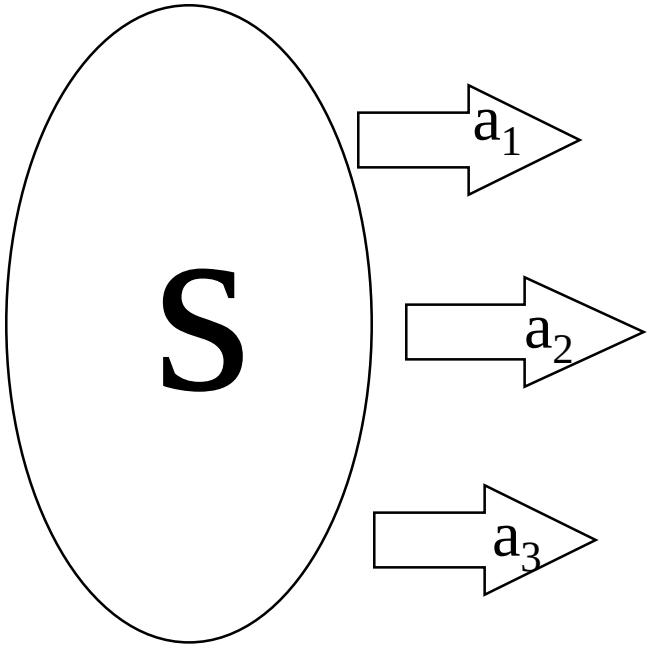
R(s,a) - immediate reward function.

$\pi : S \to A$ - policy.

$\sum_{i=0}^{\infty} \gamma^i r_i$ - discounted cumulative return.

Simple example: N- armed bandit

Single state.



Goal: Maximize sum of immediate rewards.

Given the model:
Greedy action.

Difficulty:
unknown model.

N-Armed Bandit: Models

- Stochastic/i.i.d. per action/Finite Horizon
  - Upper Confidence Bound (UCB) [A-CB-F]
- Stochastic/Markov per action/Discounted
  - Gittins Index [G]
- Adversarial
  - Exponential weights EXP3 [A-CB-F-S]
  - Uses importance sampling
    - Unbiased estimate
    - Bounded 2nd moment

Stochastic bandit: Model

- Model:
  - K actions
  - Each action a has reward distribution Z_a
    - Bounded in [0,1]
    - $\mu_a = E[Z_a]$
  - rewards
    - Optimal action
      - $a^* = \arg\max_a \mu_a$
      - $\mu^* = \max \mu_a$
    - $\Delta_a = \mu^* - \mu_a$
- Online Algorithm
  - Selects dist x_t
  - Reward: $r_t = \langle x_t, \mu_t \rangle$
  - Cumulative:

    $R_T = \sum_{t=1}^{T} r_t$
- Regret:
  - $T\mu^* - R_T$

Upper Confidence Bound (UCB)

- UCB algorithm:
  - Sample each action once (first K times)
  - $\hat{\mu}_a$ = average action a
  - T_a = num. action a
- At time t:
  - $U_a = \hat{\mu}_a + \sqrt{\frac{2\log T}{T_a}}$
  - $a_t = \max \arg_a U_a$

- Conceptually:
  - Per action
    $[\ldots\ldots \hat{\mu}_a \ldots\ldots]$
  - Selecting:
    $[\ldots\ldots \hat{\mu}_{a_.} \ldots [\hat{\mu}_{a^*}].]$
- Theorem:
- $Regret = \sum_{a=2}^{K} \Delta_a E[T_a]$
  $\leq \sum_{a=2}^{K} \frac{16 \log T}{\Delta_a} + c \sum_{a=1}^{K} \Delta_a$

Proof idea (UCB)

- With probability $1\text{-}T^{-4}$
  - For all $s \leq T$

$$|\mu_a - \hat{\mu}_a| \leq \sqrt{\frac{2\log T}{s}}$$

- Assume it holds:
  - $\mu_{a^*} - \sqrt{\frac{2\log T}{m_{a^*}}} \leq \hat{\mu}_{a^*}^t$
  - $\hat{\mu}_a^t \leq \mu_a + \sqrt{\frac{2\log T}{m_a}}$

- Bounding T_a
  - $\hat{\mu}_{a^*}^t \leq \hat{\mu}_a^t$
  - $\Delta_a = \mu_{a^*} - \mu_a$
  - $\Delta_a \leq \sqrt{\frac{2\log T}{m_{a^*}}} + \sqrt{\frac{2\log T}{m_a}}$
  - $m_a \leq 8\Delta_a^{-2} \log T$, or $m_{a^*} \leq 8\Delta_a^{-2} \log T$
  - $E[T_a] \leq 16\Delta_a^{-2} \log T + c$

Planning - Basic Problems.

Given a complete MDP model.

Policy evaluation - Given a policy π, estimate its return.

Optimal control - Find an **optimal policy π^*** (maximizes the return from any start state).

Planning - Value Functions

$V^{\pi}(s)$ The expected return starting at state s following π.

$Q^{\pi}(s,a)$ The expected return starting at state s with action a and then following π.

$V^*(s)$ and $Q^*(s,a)$ are define using an optimal policy π^*.

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

Planning - Policy Evaluation

Discounted infinite horizon (Bellman Eq.)

$$V^{\pi}(s) = E_{s' \sim \pi(s)} [\, R(s, \pi(s)) + \gamma V^{\pi}(s') \,]$$

Rewrite the expectation

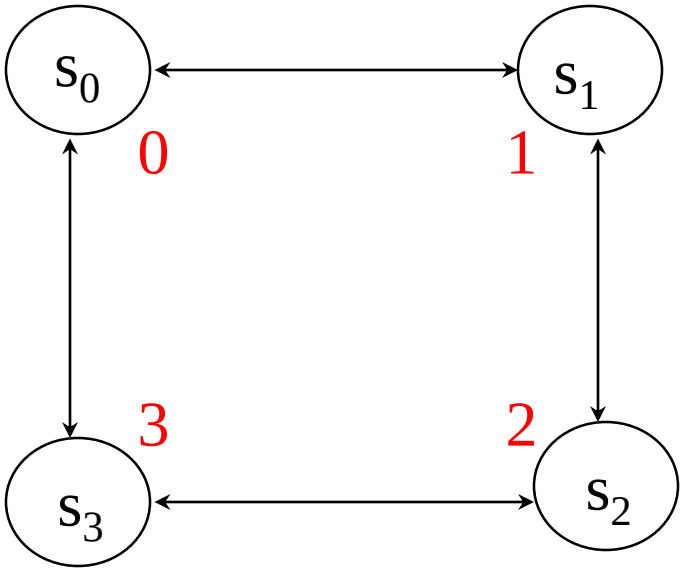
$$V^{\pi}(s) = E[R(s, \pi(s))] + \gamma \sum_{s'} \delta(s, \pi(s), s') V^{\pi}(s')$$

Linear system of equations.

Algorithms - Policy Evaluation
Example

A={+1,-1}
$\gamma = 1/2$
$\delta(s_i,a) = s_{i+a}$
π random

$\forall a: R(s_i,a) = i$

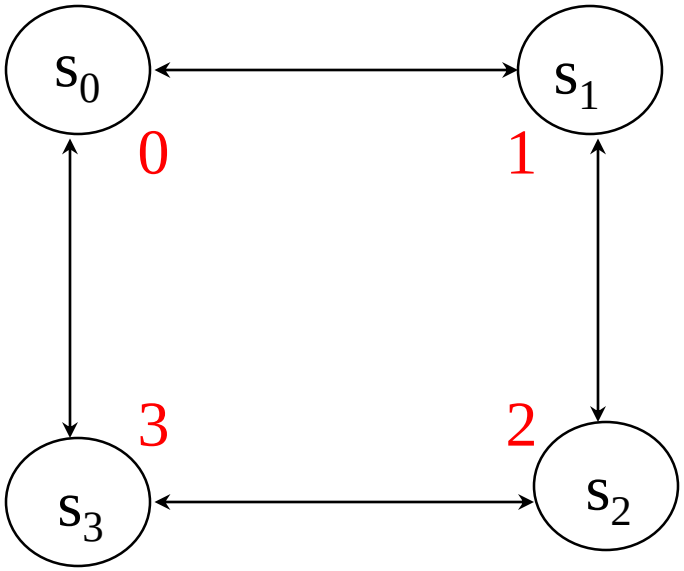


$V^{\pi}(s_0) = 0 + \gamma [\pi(s_0,+1)V^{\pi}(s_1) + \pi(s_0,-1) V^{\pi}(s_3)]$

Algorithms -Policy Evaluation
Example

$A=\{+1,-1\}$
$\gamma = 1/2$
$\delta(s_i,a)= s_{i+a}$
π random

$\forall a: R(s_i,a) = i$



$V^{\pi}(s_0) = 5/3$
$V^{\pi}(s_1) = 7/3$
$V^{\pi}(s_2) = 11/3$
$V^{\pi}(s_3) = 13/3$

$V^{\pi}(s_0) = 0 + (V^{\pi}(s_1) + V^{\pi}(s_3))/4$

Algorithms - optimal control

State-Action Value function:

$$Q^{\pi}(s,a) = E\,[\,R(s,a)] + \gamma\, E_{s' \sim (s,a)}\,[\,V^{\pi}(s')]$$

Note $V^{\pi}(s) = Q^{\pi}(s, \pi(s))$

For a deterministic policy π.

Algorithms - optimal control

CLAIM: A policy π is optimal if and only if at each state *s*:

$$V^{\pi}(s) = MAX_a \{Q^{\pi}(s,a)\} \qquad \textbf{(Bellman Eq.)}$$

PROOF: (part I) Assume there is a state *s* and action *a* s.t.,

$$V^{\pi}(s) < Q^{\pi}(s,a).$$

Then the strategy of performing *a* at state *s* (the first time) is better than π.

This is true each time we visit *s*, so the policy that performs action *a* at state *s* is better than π. □

Algorithms - optimal control

PROOF: (part II, sketch) The main issue is that V^* is unique.
We define a non-linear operator T:

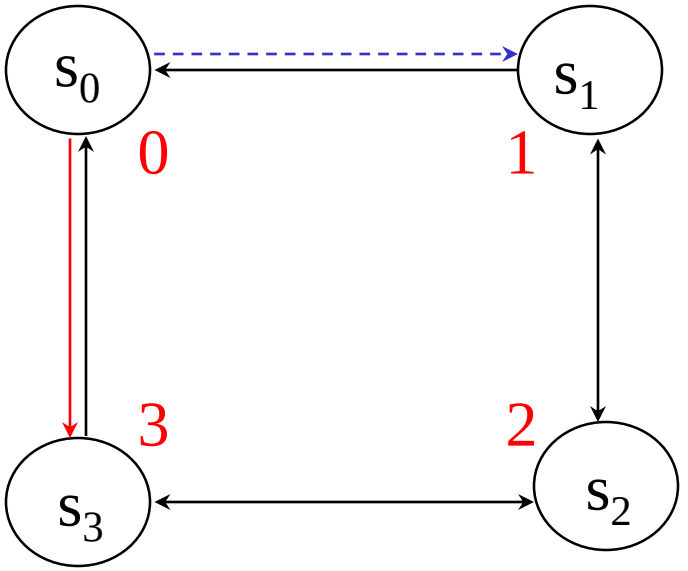
$$TV(s) \leftarrow \max_a \{R(s,a) + \gamma \sum_{s'} \delta(s,a,s') V(s')\}$$

The operator T is contracting: $\|TV_1 - TV_2\|_\infty \leq \gamma \|V_1 - V_2\|_\infty$
For any optimal policy $TV^* = V^*$
This implies that V^* is unique!
Similarly Q^* is unique!
Therefore if for π we have:
$V^\pi(s) = MAX_a \{Q^\pi(s,a)\}$
Then ***$V^\pi = TV^\pi$***

Algorithms -Optimal control
Example

A={+1,-1}
$\gamma = 1/2$
$\delta(s_i,a) = s_{i+a}$
π random

$R(s_i,a) = i$



$Q^{\pi}(s_0,+1) = 5/6$
$Q^{\pi}(s_0,-1) = 13/6$

$$Q^{\pi}(s_0,+1) = 0 + \gamma V^{\pi}(s_1)$$

Algorithms -optimal control
Example

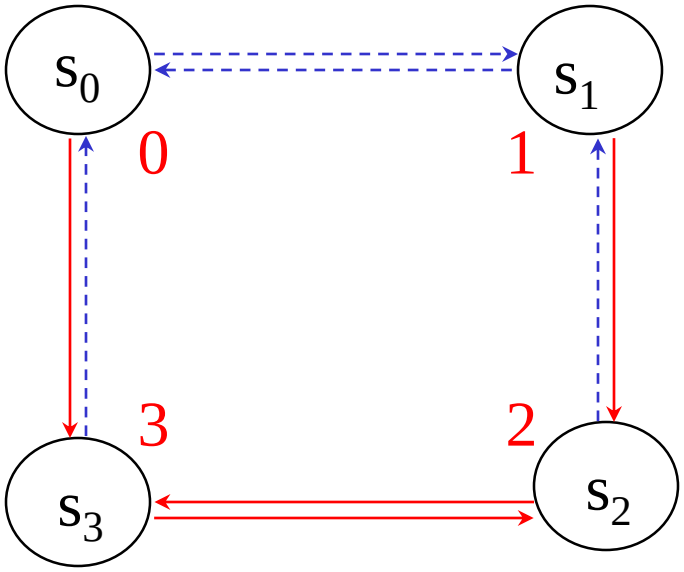
A={+1,-1}

$\gamma = 1/2$

$\delta(s_i,a) = s_{i+a}$

π random

$R(s_i,a) = i$



Changing the policy using the state-action value function.

MDP - computing optimal policy

1. Linear Programming

2. Value Iteration method.

$$V^{i+1}(s) \leftarrow \max_a \{R(s,a) + \gamma \sum_{s'} \delta(s,a,s') V^i(s')\}$$

3. Policy Iteration method.

$$\pi_i(s) = \arg\max_a \{Q^{\pi_{i-1}}(s,a)\}$$

Convergence: Value Iteration

- Distance of V^i from the optimal V^* (in L_∞)

$$Q^i(s,a) = R(s,a) + \gamma \sum_{s'} \delta(s,a,s') V^i(s')$$

$$V^*(s) - Q^i(s,a^*) = \gamma \sum_{s'} \delta(s,a^*,s')[V^*(s') - V^i(s')]$$

$$\leq \gamma \| V^* - V^i \|_\infty$$

$$V^*(s) - V^{i+1}(s) \leq V^*(s) - Q^i(s,a^*)$$

$$\| V^* - V^{i+1} \|_\infty \leq \gamma \| V^* - V^i \|_\infty$$

Convergence Rate: $1/(1-\gamma)$ ONLY Pseudo Polynomial

Convergence: Policy Iteration

- Policy Iteration Algorithm:
  - Compute $Q^{\pi}(s,a)$
  - Set $\pi(s) = \arg\max_a Q^{\pi}(s,a)$
  - Reiterate
- Convergence:
  - Policy can only improve

    $\forall s \; V^{t+1}(s) \geq V^{t}(s)$
    - Less iterations then Value Iteration,
      - But more expensive iterations.
- Complexity: worse case both poly and exp
  - Exponential – arbitrary discount factor [HDJ]
  - Polynomial – constant discount factor [Y]

Outline

- Done
  - Goal of Reinforcement Learning
  - Mathematical Model (MDP)
  - Planning
    - Value iteration
    - Policy iteration
- Now: Learning Algorithms
  - Model based
  - Model Free

Learning Algorithms

Given access only to actions perform:

1. policy evaluation.
2. control - find optimal policy.

Two approaches:

1. Model based (Dynamic Programming).
2. Model free (Q-Learning).

Learning - Model Based

Estimate the model from the observation.
(Both transition probability and rewards.)

Use the estimated model as the true model,
and find optimal policy.

If we have a “good” estimated model, we should have a “good” estimation.

Learning - Model Based: off policy

- Let the policy run for a “long” time.
  - what is “long” ?!
  - Assuming some “exploration”
- Build an “observed model”:
  - Transition probabilities
  - Rewards
    - Both are independent!
- Use the “observed model” learn optimal policy.

Approximate Model

- Assume:
  - Probabilities: For all s,a
    $$||p_1(\cdot\,|s,a) - p_2(\cdot\,|s,a)||_1 \leq \beta$$
  - Rewards: $0 \leq R_i(s,a) \leq M$
  - Claim: for any policy π:
    $$|V_1^\pi(s) - V_2^\pi(s)| \leq \frac{M\gamma\beta}{(1-\gamma(1-\beta))(1-\gamma)}$$

Approximate Optimal policy

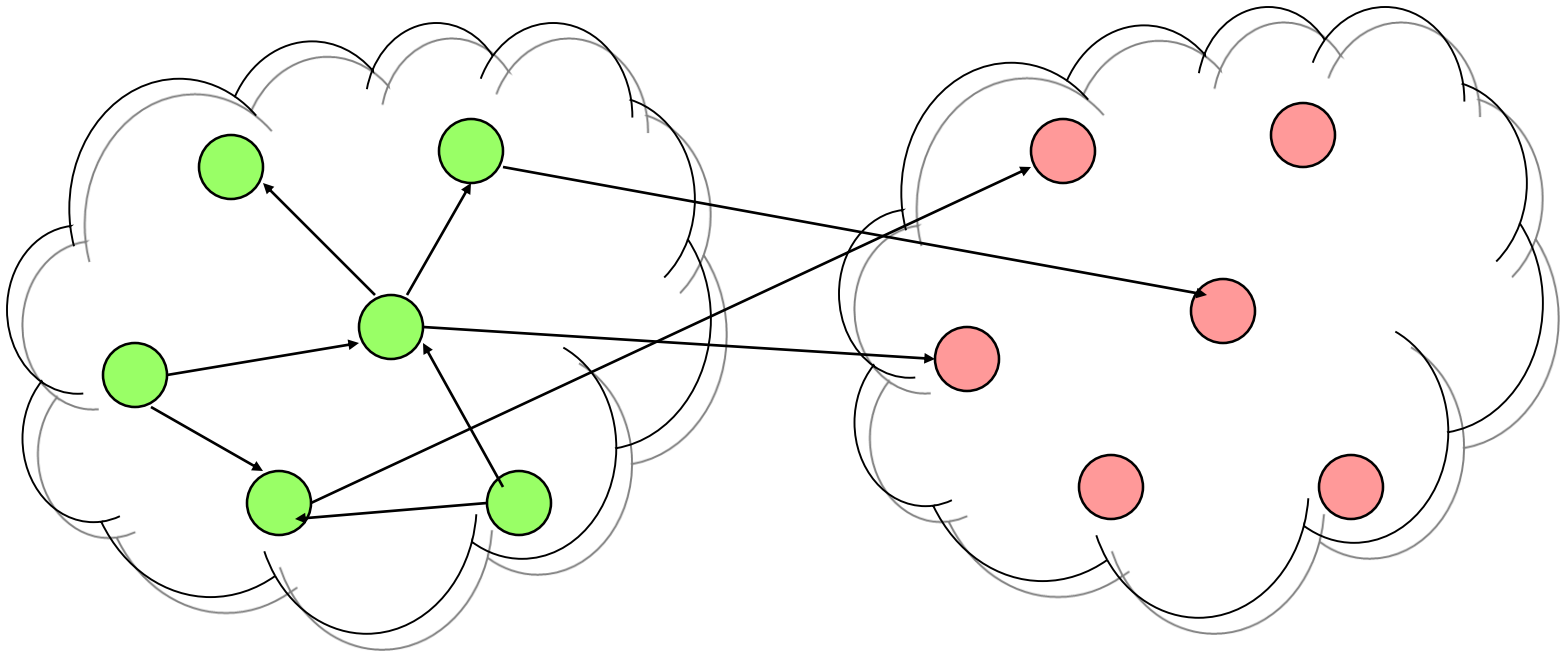
- Assume each s,a sampled at least $O(\frac{|S|+\log\frac{|S||A|}{\delta}}{\beta^2})$ times
  - $P(||\hat{p}(\cdot|s,a) - p(\cdot|s,a)||_1 \geq \beta) \leq \frac{\delta}{|S||A|}$
    - Bretagnolle-Huber-Carol inequality
- With probability $1-\delta$: an optimal policy on $\hat{p}(\cdot|s,a)$ is ε-near optimal for $p(\cdot|s,a)$ for

$$\epsilon = \frac{M\gamma\beta}{(1-\gamma)((1-\gamma(1-\beta))}$$

Learning - Model Based: on policy

- The learner has control over the action.
  - The immediate goal is to lean a model
- As before:
  - Build an “observed model”:
    - Transition probabilities and Rewards
  - Use the “observed model” to estimate value of the policy.
- Accelerating the learning:
  - How to reach “new” places ?!

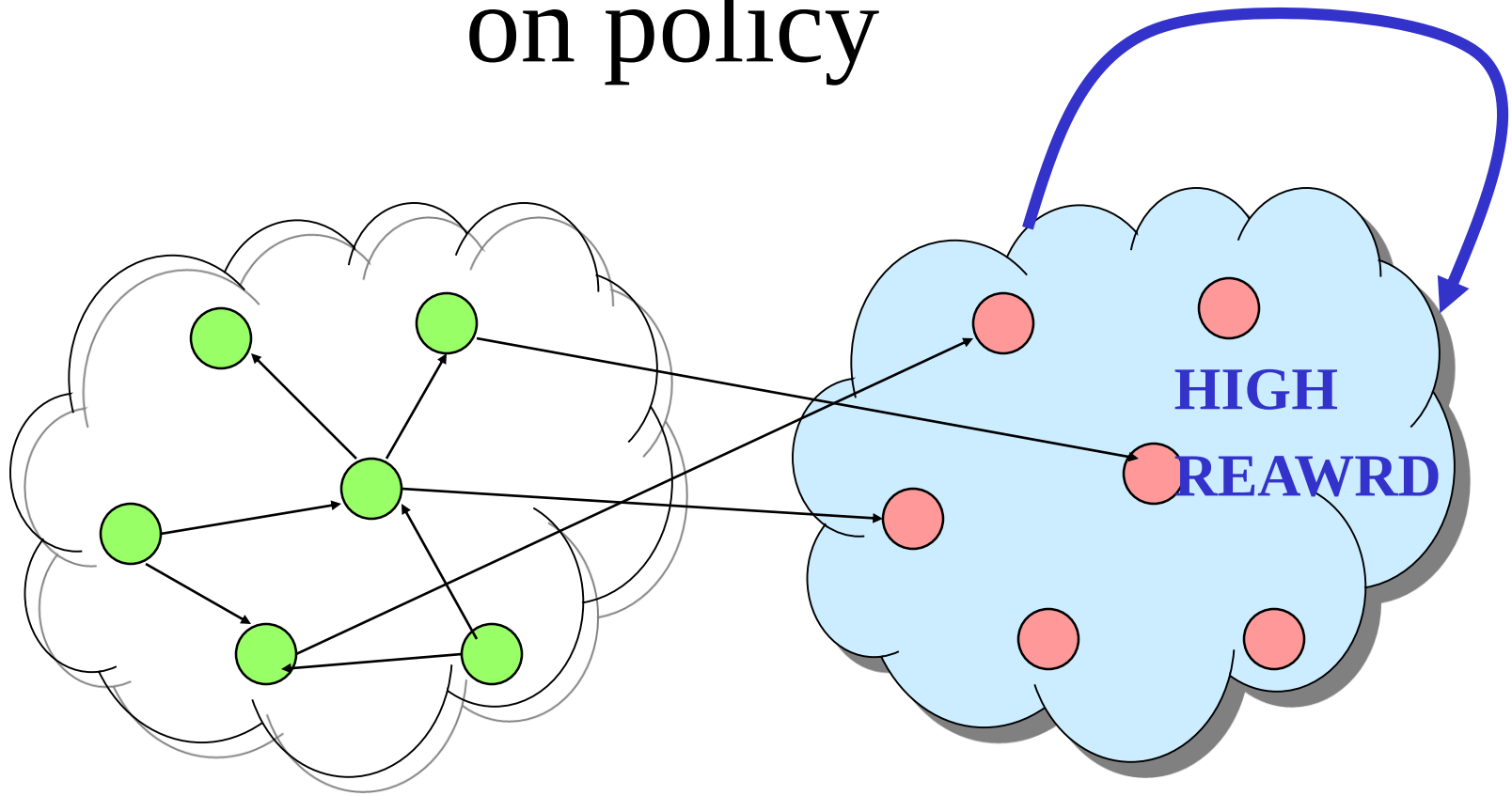
Learning - Model Based: on policy



Well sampled nodes

Relatively unknown nodes

Learning - Model Based: on policy



Well sampled nodes

Relatively unknown nodes

Exploration → Planning in new MDP

Model-Free learning

Q-Learning: off policy

Basic Idea: Learn the Q-function.

On a move (s,a) → s' update:

Old estimate

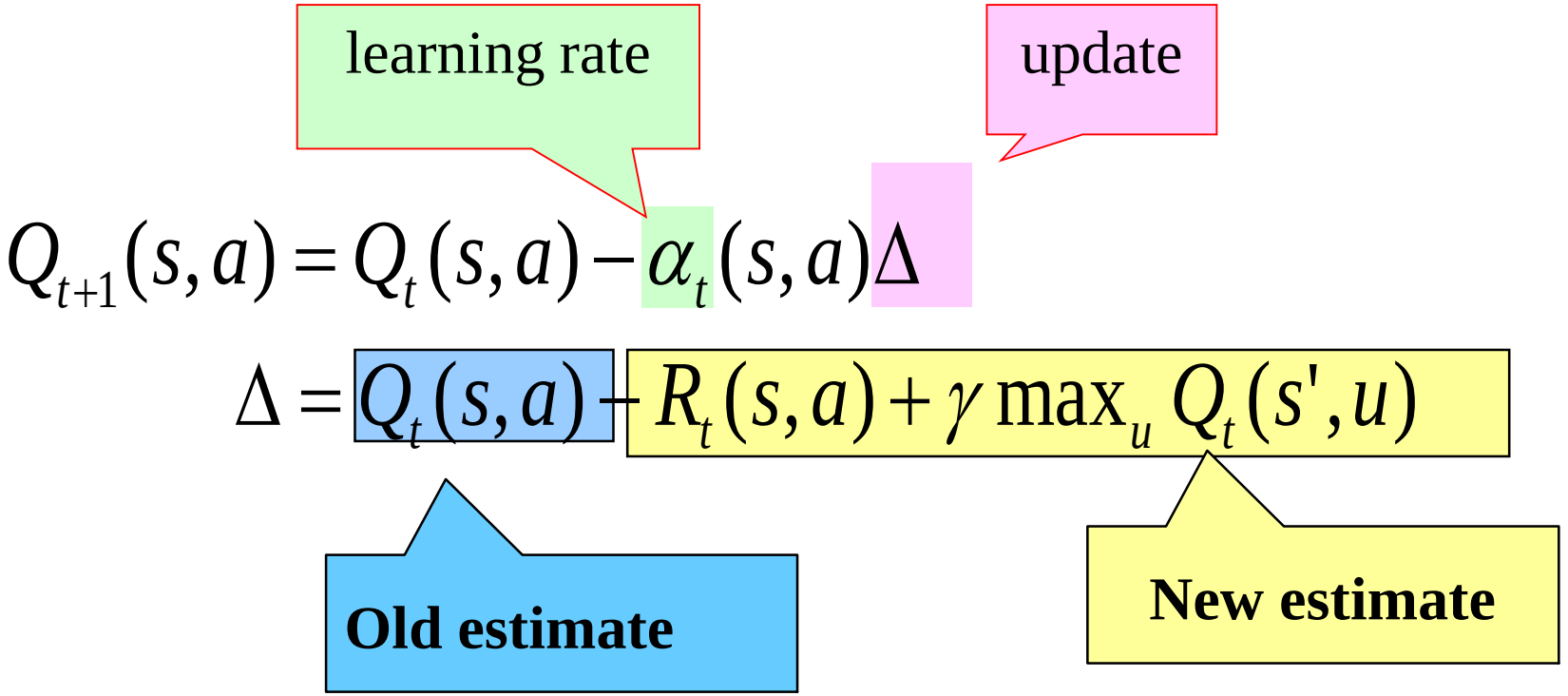
$$Q_{t+1}(s,a) = (1-\alpha_t(s,a))Q_t(s,a) +$$

$$\alpha_t(s,a)[R_t(s,a) + \gamma \max_u Q_t(s',u)]$$

New estimate

Learning rate $\alpha_t(s,a)=1/t^w$

Q-Learning: update equation



Q-Learning: Intuition

- Updates based on the difference:

$$\Delta = Q_t(s,u) - [R_t(s,a) + \gamma \max_u Q_t(s',u)]$$

- Assume we have the right Q function
- Good news: The expectation is Zero !!!
- Challenge: understand the dynamics
  - stochastic process

QL: Classical Convergence Results

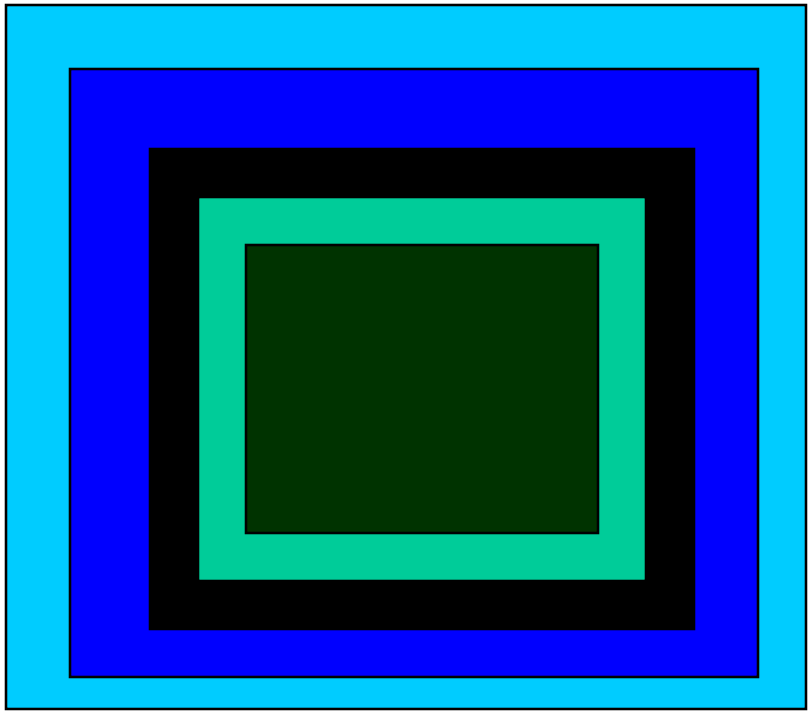
- Convergence in the limit (off policy)
  - If every *(s,a)* performed infinitely often,
  - Learning rates, for every *(s,a):*

$$\sum_t \alpha_t(s,a) = \infty \quad \text{and} \quad \sum_t \alpha_t^2(s,a) = O(1)$$

- **$Q_t(s,a)$ Converges with probability 1 to $Q^*(s,a)$**

QL: Classical Proof Methodology

Let $r_t = Q_t - Q^*$



$$D_0 = V_{\max}$$

$$D_1 = (1-\beta)D_0$$

$$D_2 = (1-\beta)D_1$$

$$D_3 = (1-\beta)D_2$$

$$D_4 = (1-\beta)D_3$$

QL: Classical Proof Methodology

- Divide the error to two parts:
  - One which contracts.
  - One which is noise
    - the difference of the expected and sampled values.
- The first drops deterministically (as expected).
- The second has zero expectation,
  - bound it using law of large numbers.
    - needs to be bounded for an entire phase.

Q-Learning: Qualitative Bounds

- For the contraction part:
  - Compute the time until the error shrinks to $(1-(3/2)\beta)D_i$.
- Derive a bound on the noise.
  - Has to hold during **all** the next iteration.
- Duration of an iteration i:
  - Depends on the initial time T_i
  - An upper bound on the time $T_{k+1} < T_k + T_k^w$

Model Free Algorithms

- Many algorithms have similar structure:
  - Q Learning

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha[r_t + \gamma \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t)]$$

  - SARSA

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha[r_t + \gamma Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)]$$

  - TD(0)

$$V_{t+1}(s_t) = V_t(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)]$$

  - TD(λ)

MDP: Large state space restricted value function

- Need to create state attributes
- Modify the notation
- Rather than $Q(s,a)$ have $Q_a(s)$
- Each action has a function $Q_a(s)$
- Greedy(Q) = $MAX_a\ Q_a(s)$
- Learn each $Q_a(s)$ independently!
  - Almost a ML problem

Function Approximation

- Use a restricted model for $Q_a(s)$
- Have an attribute vector:
  - Each state s has a vector $vec(s)=x_1 \ldots x_k$
  - Normally $k << |S|$
- Examples:
  - Neural Network
  - Decision tree
  - Linear Function
    - Weights $\theta = \theta_1 \ldots \theta_k$
    - Value $\Sigma \theta_i x_i$

Gradient Decent

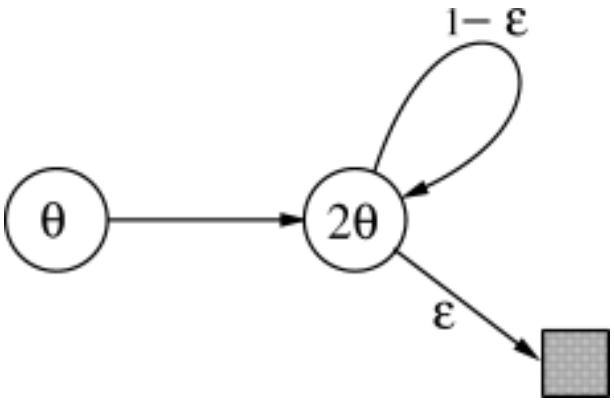
- Minimize Squared Error
  - Square Error = ½ Σ P(s) $[V^{\pi}(s) - V_{\theta}(s)]^2$
  - P(s) is a weighting on the states
- Algorithm:
  - $\theta(t+1) = \theta(t) + \alpha [V^{\pi}(s_t) - V_{\theta(t)}(s_t)] \nabla_{\theta(t)} V_{\theta(t)}(s_t)$
  - $\nabla_{\theta(t)}$ = partial derivatives
  - Replace $V^{\pi}(s_t)$ by a sample
    - Monte Carlo: use R_t for $V^{\pi}(s_t)$
    - TD(0) use A_t for $[V^{\pi}(s_t) - V_{\theta(t)}(s_t)]$

Linear Functions

- Linear function: $\Sigma\ \theta_i\ x_i = < \theta, x >$
- Derivative $\nabla_{\theta(t)}\ V_t(s_t) = vec(s_t)$
- Update Rule:
  - $\theta_{t+1} = \theta_t + \alpha\ [V^{\pi}(s_t) - V_t(s_t)]\ vec(s_t)$
  - MC: $\theta_{t+1} = \theta_t + \alpha\ [\ R_t - < \theta_t\ , s_t >]\ vec(s_t)$
  - TD: $\theta_{t+1} = \theta_t + \alpha\ A_t\ vec(s_t)$

Linear functions: non-convergence [T-vR]

- MDP:



- Minimizing SQE

$$\theta_{k+1} = \arg\min_{\theta\in\Re} \sum_{s\in\mathcal{S}} \Big[V_\theta(s) - E_\pi\{r_{t+1} + \gamma V_{\theta_k}(s_{t+1}) | s_t = s\}\Big]^2$$

$$= \arg\min_{\theta\in\Re} \Big[\theta - \gamma 2\theta_k\Big]^2 + \Big[2\theta - (1-\varepsilon)\gamma 2\theta_k\Big]^2$$

$$= \frac{6-4\varepsilon}{5}\gamma\theta_k,$$

Linear Function Approximation

optimal control

- Q-Learning with LFA might diverge. [Baird,Gordon]
- Sarsa with LFA converges [NR,S].
- Monte Carlo with LFA converges.

Large Scale MDP

Previous Methods: Tabular. (small state space).

Large scale  Exponential size.

CS view of computing.

Large scale MDP

Approaches:

1. Restricted Value function.
2. Restricted policy class.
3. Restricted model of MDP.

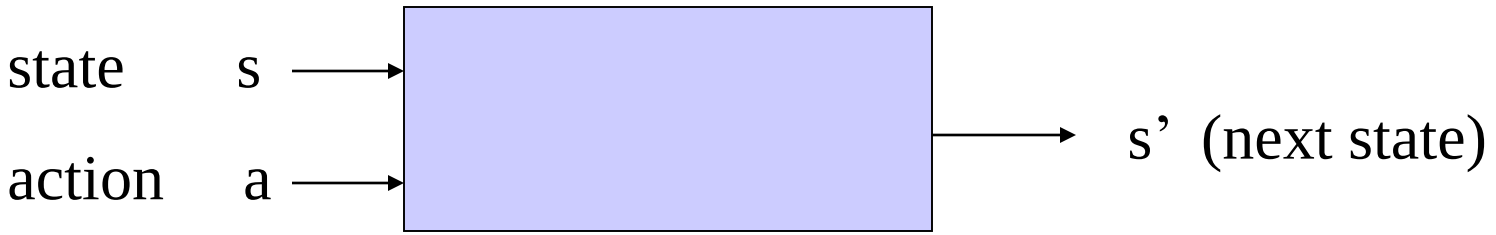
4. Generative model: a different MDP representation.

Large MDP - model

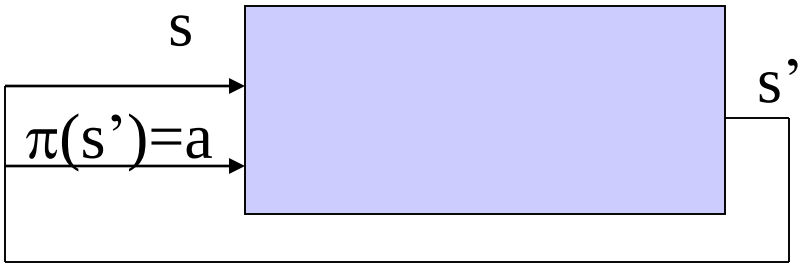
Exponential (infinite) size MDP.

How is the MDP described?

Generative model



Generative Model:
Policy Evaluation



Generative Model:
near optimal policy

AIM: build a (near) optimal policy π efficiently.

$$\forall s \qquad V^{\pi}(s) \geq V^{*}(s) - \varepsilon$$

Theorem [KMN]:

Compute a "near" optimal stochastic policy

and runs in time subexponential in ε and

R_{max} and exponential in $1/(1-\gamma)$.

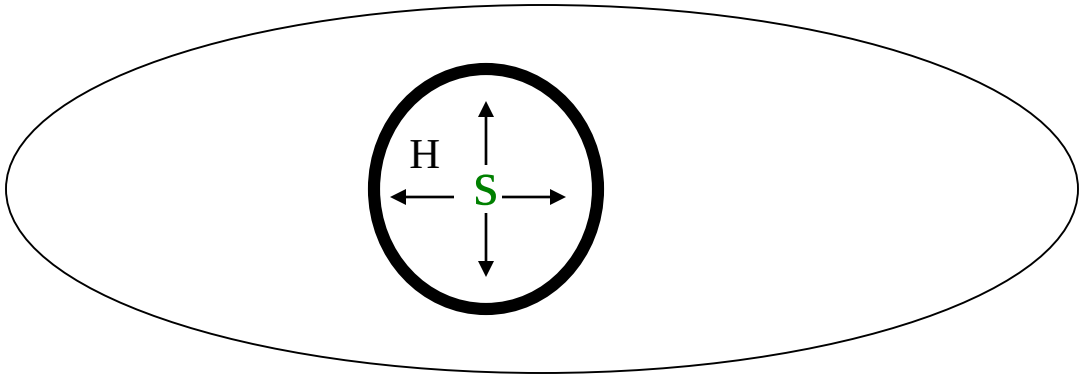
(Running time is independent from number of states!)

Generative Model:
Computing near optimal policy

Input : s (state)

Output : a (action)

Consider only states distance at most H



Easy case:

Every state and action has only few next states.

Consider the sub-MDP which includes only the states at distance < H from s.

Find an optimal policy in the sub-MDP.

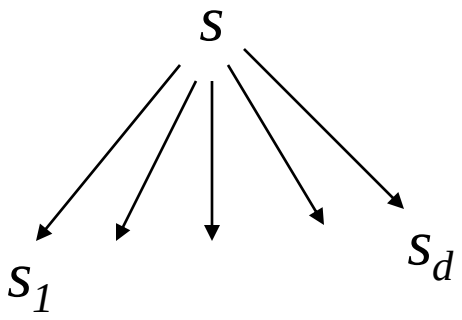
That policy is near optimal, for state s, in the original MDP.

General Case:

Action a from state s might reach many states.

(E.g. uniform distribution over all states.)

Main Idea: Use sampling to approximate return.



To approximate $Q^*(s,a)$ draw d states s_i and approximate $V^*(s_i)$

Problem: we replace one state with d states ?!

Progress: We can tolerate more error!

Large MDP - Algorithm.

EstimateQ(0,s,a): return 0

EstimateQ(n,s,a) :

sample $s_1, \ldots, s_d$ using $\delta(s,a,s_i)$

return $R(s,a) + \gamma \frac{1}{d} \sum_{i=1}^{d} EstimateV(n-1, s_i)$

EstimateV(n,s):

return MAX_a {*EstimateQ(n,s,a)*}

Output: action *a* that maximizes *EstimateQ(H,s,a).*

Running time ~ $(kd)^H$

Need to show: w. h. p. A chooses a near optimal action.

Two sources of error for *EstimateQ(n,s,a):*

1. Finite sample (only *d* states).

2. Error in approximating the *EstimateV(n-1,s)* function.

Proof idea:

1. Choose *d* such that finite sample error is small.

2. Show that estimation error decreases with *n*.

Proof Sketch

$$|Q^*(s,a) - EstimateQ(n,s,a)|$$

$$= \gamma\,|V^*(s) - \frac{1}{d}\sum_i EstimateV(n-1,s_i)|$$

$$\leq \gamma\,|V^*(s) - \frac{1}{d}\sum_i V^*(s_i)|$$

finite sample error

$$+ \gamma\,\frac{1}{d}\sum_i |V^*(s_i) - EstimateV(n-1,s_i)|$$

estimation error

$$\leq \gamma[\varepsilon + b_{n-1}] = b_n$$

Define b_n as follows: $b_0 = V_{max}$ & $b_{n+1} = \gamma(\varepsilon + b_n)$

solving for b_n: $b_n = (\sum_{i=0}^{n} \gamma^i \varepsilon) + \gamma^n V_{\max} \leq \frac{\varepsilon}{1-\gamma} + \gamma^n V_{\max}$

Theorem:

Given a generative model,

there exists an algorithm A,

that defines a "near" optimal stochastic policy, i.e.,

$V^*(s_0) - V^A(s_0) < \varepsilon$

and runs in time $(dk)^H$, where $d = \tilde{O}\left(\frac{V_{\max}^2}{(1-\gamma)\varepsilon^2} \log \frac{k}{\delta}\right)$

and $H = O\left(\log_\gamma \left(\frac{\varepsilon}{V_{\max}}\right)\right) = O\left(\frac{1}{1-\gamma} \log\left(\frac{V_{\max}}{\varepsilon}\right)\right)$

(Running time is independent from number of states!)

Partially Observable MDP

Rather than observing the state we observe some function of the state.

Ob - Observable function.
a random variable for each states.

Example: (1) Ob(s) = s+noise. (2) Ob(s) = first bit of s.

Problem: different states may “look” similar.

The optimal strategy needs to consider the entire history!

POMDP - Belief State Algorithm

- Bayesian approach: Assume known model
- Belief state: Distribution over states
- Next state

$$\Pr[s_{t+1} = s' | h_t, a_t] = \sum_s \Pr[s_t = s | h_t, a_t] \Pr[s' | s, a_t]$$

$$\Pr[s_{t+1} = s' | h_t, a_t, ob_{t+1}] = \frac{\Pr[ob_{t+1} | s_{t+1} = s', h_t, a_t] \Pr[s_{t+1} = s' | h_t, a_t]}{\Pr[ob_{t+1} | h_t, a_t]}$$

POMDP - Belief State Algorithm

- Reduction to MDP
  - States= Belief States
    - distributions over S
  - Actions: A
  - Transition function
    - Given a belief state
    - action and observation
    - return new belief state
- Problem: Infinite state space !

POMDP -

Hard computational problems.

Computing an infinite (polynomial) horizon undiscounted optimal strategy for a deterministic POMDP is P-space-hard (NP-complete) [PT,L].

Computing an infinite (polynomial) horizon undiscounted optimal strategy for a stochastic POMDP is EXPTIME-hard (P-space-complete) [PT,L].

Computing an infinite (polynomial) horizon undiscounted optimal policy for an MDP is P-complete [PT] .

Conclusion: MDPs

- Powerful modeling tool
- Efficient algorithms
  - For table look-up
- Learning algorithm
  - Unknown MDP
- Large state space
- POMDP

The End