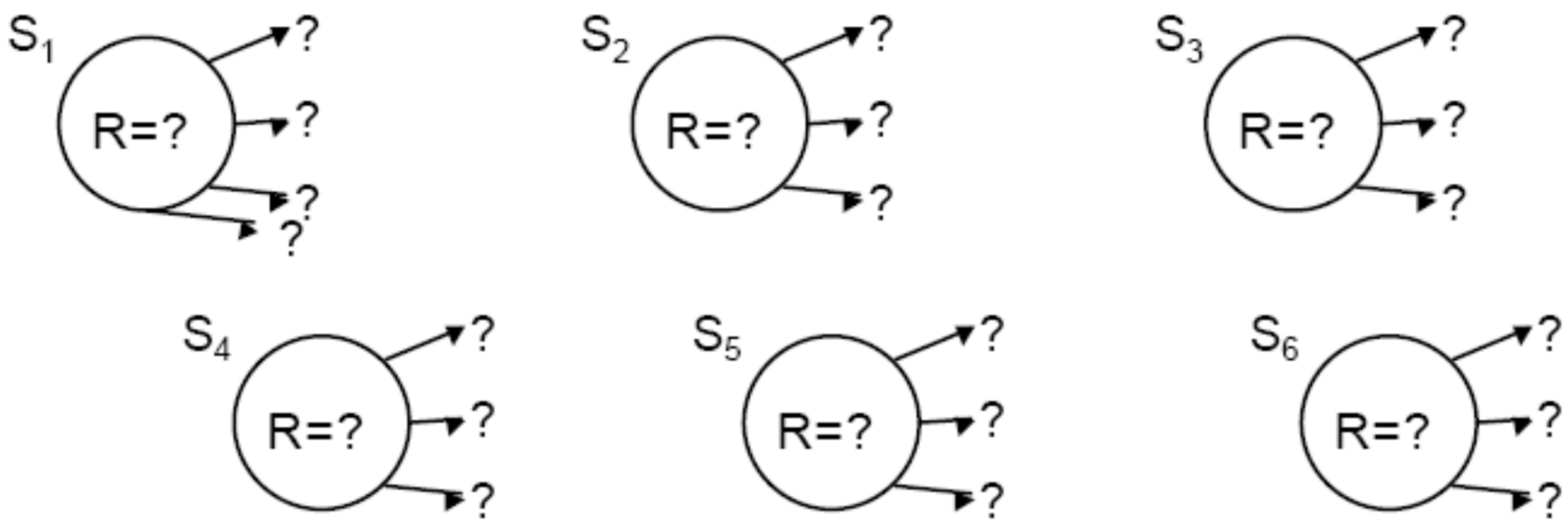
Introduction to Machine Learning

CS307 --- Fall 2022

Reinforcement Learning

Reading:
Chapter 21, R&N
Sections 13.1-13.2, 13.5, Mitchell

Learning Delayed Rewards



All you can see is a series of states and rewards:

$S_1(r=0) \rightarrow S_2(r=0) \rightarrow S_3(r=4) \rightarrow S_2(r=0) \rightarrow S_4(r=0) \rightarrow S_5(r=0)$

Task: Based on this sequence, estimate $J^*(S_1), J^*(S_2) \cdots J^*(S_6)$

Idea 1: Supervised Learning

Assume $\gamma = 0.5$.

$S_1(r=0) \rightarrow S_2(r=0) \rightarrow S_3(r=4) \rightarrow S_2(r=0) \rightarrow S_4(r=0) \rightarrow S_5(r=0)$

At t=1 we were in state S_1 and eventually got a long term discounted reward of $0+\gamma 0+\gamma^2 4+\gamma^3 0+\gamma^4 0\ldots= 1$

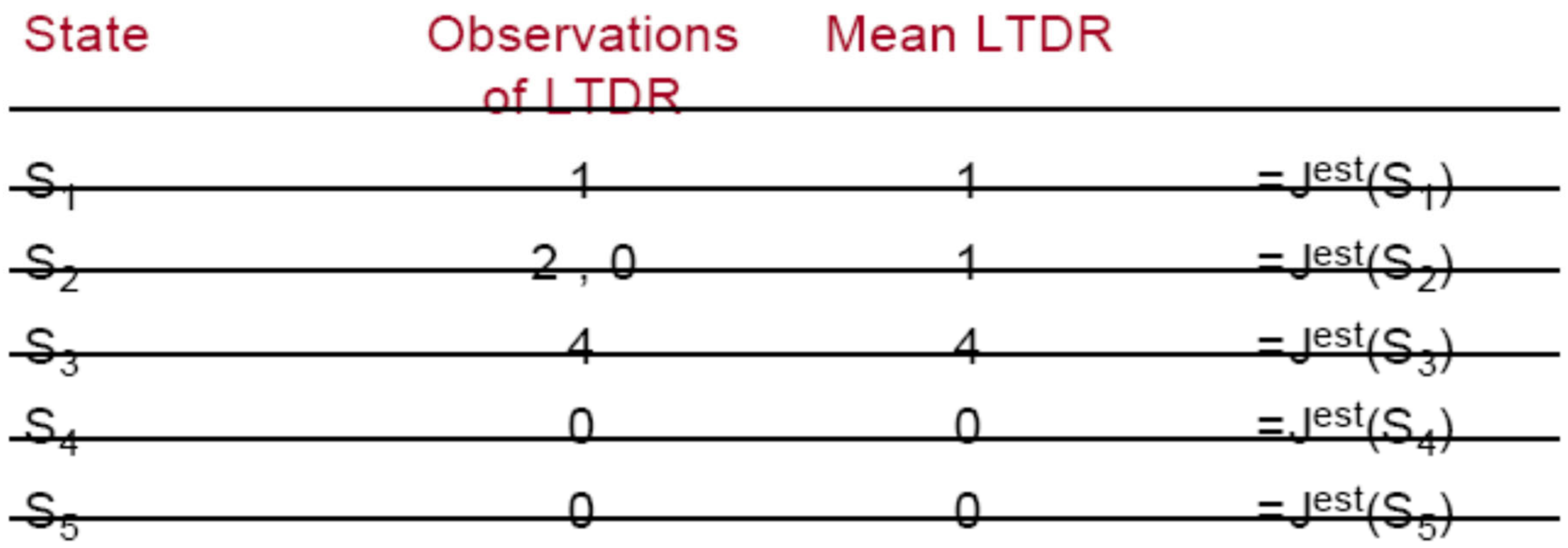
At t=2 in state S_2 ltdr = 2

At t=3 in state S_3 ltdr = 4

At t=4 in state S_2 ltdr = 0

At t=5 in state S_4 ltdr = 0

At t=6 in state S_5 ltdr = 0



State	Observations of LTDR	Mean LTDR	
S_1	1	1	$=J^{est}(S_1)$
S_2	2 , 0	1	$=J^{est}(S_2)$
S_3	4	4	$=J^{est}(S_3)$
S_4	0	0	$=J^{est}(S_4)$
S_5	0	0	$=J^{est}(S_5)$

Supervised Learning Algorithm

- Watch a trajectory

  S[0] r[0] S[1] r[1] ···· S[T]r[T]

- For t=0,1, ··· T , compute $\mathrm{J}[t] = \sum_{i=0}^{\infty} \gamma^{i} r[t+i]$

- Compute

$$\mathrm{J}^{est}(S_i) = \left(\begin{array}{c} \text{mean value of } \mathrm{J}[t] \\ \text{among all transitio ns beginning} \\ \text{in state } S_i \text{ on the trajector y} \end{array} \right)$$

Let $\mathrm{MATCHES}(S_i) = \{t \mid S[t] = S_i\}$, then define

$$\mathrm{J}^{est}(S_i) = \frac{\sum_{t \in \mathrm{MATCHES}(S_i)} \mathrm{J}[t]}{|\mathrm{MATCHES}(S_i)|}$$

- You're done!

Online Supervised Learning Algorithm

```
Initialize: $\text{Count}[S_i] = 0 \quad \forall S_i$
            $\text{SumJ}[S_i] = 0 \quad \forall S_i$
            $\text{Eligibility}[S_i] = 0 \; \forall S_i$
Observe:
        When we experience $S_i$ with reward $r$ do this:
$\forall j \; \text{Elig}[S_j] \leftarrow \gamma \text{Elig}[S_j]$
   $\text{Elig}[S_i] \leftarrow \text{Elig}[S_i] + 1$
$\forall j \; \text{SumJ}[S_j] \leftarrow \text{SumJ}[S_j] + r \times \text{Elig}[S_j]$
   $\text{Count}[S_i] \leftarrow \text{Count}[S_i] + 1$
```

Then at any time,

$J^{est}(S_j) = \text{SumJ}[S_j]/\text{Count}[S_j]$

Online Supervised Learning Economics

Given N states $S_1 \cdots S_N$, OSL needs O(N) memory.

Each update needs O(N) work since we must update all Elig[ ] array elements

Idea: Be sparse and only update/process Elig[ ] elements with values > ε for tiny ε

Easy to prove:

$$\text{As } T \to \infty,\ J^{est}(S_i) \to J^*(S_i) \quad \forall S_i$$

Online Supervised Learning

Let's grab OSL off the street, bundle it into a black van, take it to a bunker and interrogate it under 600 Watt lights.

$S_1(r=0) \rightarrow S_2(r=0) \rightarrow S_3(r=4) \rightarrow S_2(r=0) \rightarrow S_4(r=0) \rightarrow S_5(r=0)$

State	Observations of LTDR	$\hat{J}(S_i)$
S_1	1	1
S_2	2 , 0	1
S_3	4	4
S_4	0	0
S_5	0	0

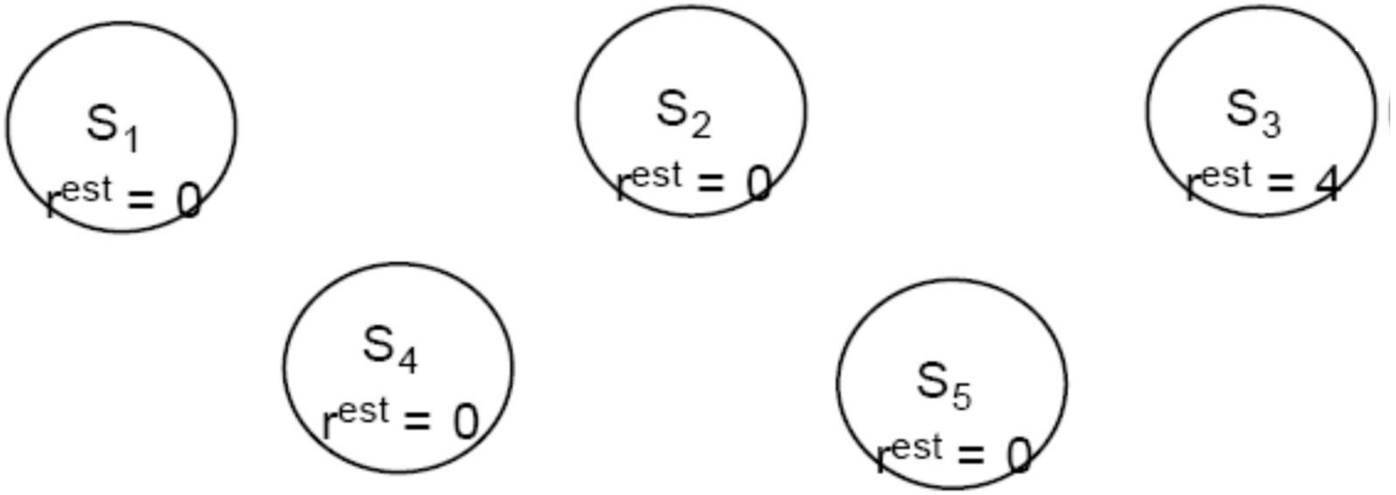
Certainty-Equivalent (CE) Learning

Idea: Do model-based learning (i.e., use your data to estimate the underlying Markov system, instead of trying to estimate J directly.

$S_1(r=0) \rightarrow S_2(r=0) \rightarrow S_3(r=4) \rightarrow S_2(r=0) \rightarrow S_4(r=0) \rightarrow S_5(r=0)$

Estimated Markov System:

You draw in the transitions +probs



What're the estimated J values?

C.E. Method for Markov Systems

Initialize:

$$\left.\begin{array}{l} \text{Count}[S_i] = 0 \\ \text{SumR}[S_i] = 0 \\ \text{Trans}[S_i,S_j] = 0 \end{array}\right\} \forall S_{i,,} S_j \qquad \begin{array}{l} \#\text{Times visited } S_i \\ \text{Sum of rewards from } S_i \\ \#\text{Times transitioned from } S_i \rightarrow S_j \end{array}$$

When we are in state S_i, and we receive reward r, and we move to S_j …

```
    $\text{Count}[S_i] \leftarrow \text{Count}[S_i] + 1$
    $\text{SumR}[S_i] \leftarrow \text{SumR}[S_i] + r$
    $\text{Trans}[S_i,S_j] \leftarrow \text{Trans}[S_i,S_j] + 1$
```

Then at any time

$r^{est}(S_i) = \text{SumR}[S_i] / \text{Count}[S_i]$

$P^{est}_{ij} = \text{Estimated Prob}(\text{next} = S_j \mid \text{this} = S_i)$

$\qquad = \text{Trans}[S_i,S_j] / \text{Count}[S_i]$

C.E. for Markov Systems (cont'd)

So at any time we have $\forall S_{i,,} S_j$

$r^{est}(S_i)$ and $P^{est}(\text{next}=S_j \mid \text{this}=S_i) = P^{est}_{ij}$

So at any time we can solve the set of linear equations

$$J^{est}(S_i) = r^{est}(S_i) + \gamma \sum_{S_j} P^{est}(S_j | S_i) J^{est}(S_j)$$

C.E. Online Economics

Memory: $O(N^2)$

Time to update counters: $O(1)$

Time to re-evaluate J^{est}

- $O(N^3)$ if use matrix inversion
- $O(N^2 k_{CRIT})$ if use value iteration and we need k_{CRIT} iterations to converge
- $O(N k_{CRIT})$ if use value iteration, and k_{CRIT} to converge, and Markov System is Sparse (i.e. mean # successors is constant)

Certainty Equivalent Learning: Complaint

Memory use could be $O(N^2)$!

And time per update could be $O(Nk_{CRIT})$ up to $O(N^3)$!

Too expensive for some people.

Where Are We?

Trying to do online J^{est} prediction from streams of transitions

	Space	J^{est} Update Cost	Data Efficiency:
Supervised Learning	$O(N_s)$	$O(\frac{1}{\log(1/?)})$	☹
Full C.E. Learning	$O(N_{so})$	$O(N_{so}N_s)$ $O(N_{so}k_{CRIT})$	☺

N_{so}= # state-outcomes (number of arrows on the M.S. diagram)

N_s= # states

Temporal Difference Learning

Model-free learning: learning without trying to estimate the parameters of a Markov system

Only maintain a J^{est} array … nothing else

So you've got $J^{est}(S_1)$, $J^{est}(S_2)$, ⋯, $J^{est}(S_N)$

and you observe

$$S_i \; r \rightarrow S_j$$

A transition from i that receives an immediate reward of r and jumps to j

what should you do?

TD Learning

$S_i \; r \rightarrow S_j$

We update = $J^{est}(S_i)$

We nudge it to be closer to expected future rewards

$$\left.\begin{aligned} J^{est}(S_i) \leftarrow (1-\alpha) J^{est}(S_i) + \\ \alpha\left[\text{Expected future rewards}\right] \end{aligned}\right\} \text{WEIGHTED SUM}$$

$$= (1-\alpha) J^{est}(S_i) + \alpha\left[r + \gamma J^{est}(S_j)\right]$$

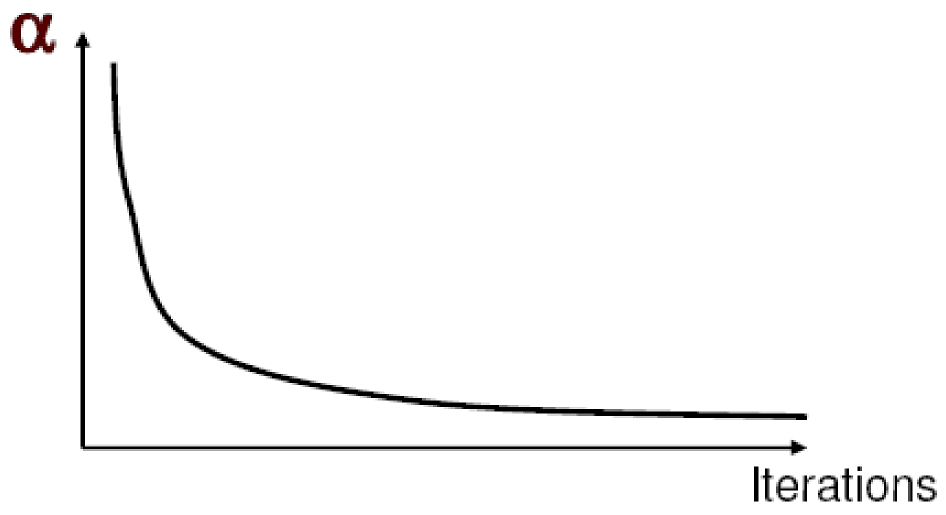
α is the learning rate.

Decaying Learning Rate

This is: $\alpha_t = 1/t$

This isn't: $\alpha_t = \alpha_0$

This is: $\alpha_t = ß/(ß+t)$ [e.g. ß=1000]



- Start with large α

  → Not confident in our current estimate so we can change it a lot

- Decrease α as we explore more

  → We are more and more confident in our estimate so we don't want to change it a lot

Learning Markov Systems: Summary

		Space	J Update Cost	Data Efficiency
MODEL-BASED	Supervised Learning	$0(N_s)$	$0\left(\frac{1}{\log 1/\gamma}\right)$	😐
	Full C.E. Learning	$0(N_{so})$	$0(N_{so}N_s)$ $0(N_{so}k_{CRIT})$	🙂
MODEL FREE	TD(0)	$0(N_s)$	$0(1)$	☹

Learning Policies for MDPs

The heart of Reinforcement Learning

General Problem

World: You are in state 34.
Your immediate reward is 3. You have 3 actions.
Robot: I’ll take action 2.
World: You are in state 77.
Your immediate reward is -7. You have 2 actions.
Robot: I’ll take action 1.
World: You’re in state 34 (again).
Your immediate reward is 3. You have 3 actions.

The Markov property means once you’ve selected an action the P.D.F. of your next state is the same as the last time you tried the action in this state.

Learning from experience …

The Task

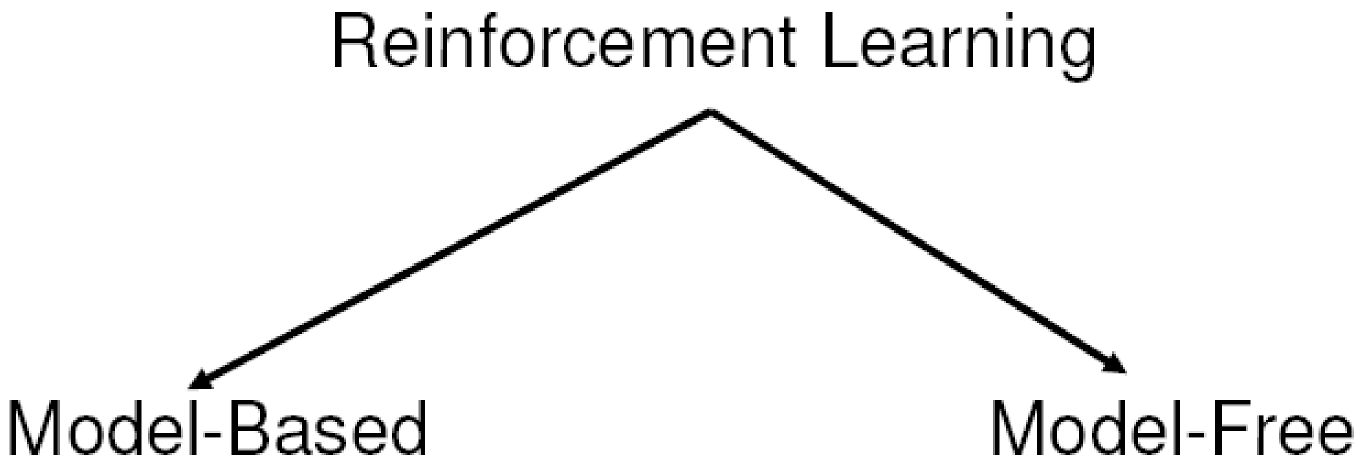
- Hence, all you can see is a series of states and rewards, except that each arrow is associated with an action.

$$S(r=1) \xrightarrow{a1} S_2(r=0) \xrightarrow{a2} S_3(r=1) \xrightarrow{a3} S_4(r=0) \xrightarrow{a4} S_5(r=1) \xrightarrow{a5} S_6(r=0)$$

  - We don’t know the model of the environment
  - We don’t know the P(. |. , a): transition probabilities
  - We don’t know the R(.): reward associated with a state

- Task is still the same as in an MDP: Find an optimal policy

Classes of Techniques



- Try to learn an explicit model of P(.|., a) and R(.)

- Recover an optimal policy without ever estimating a model

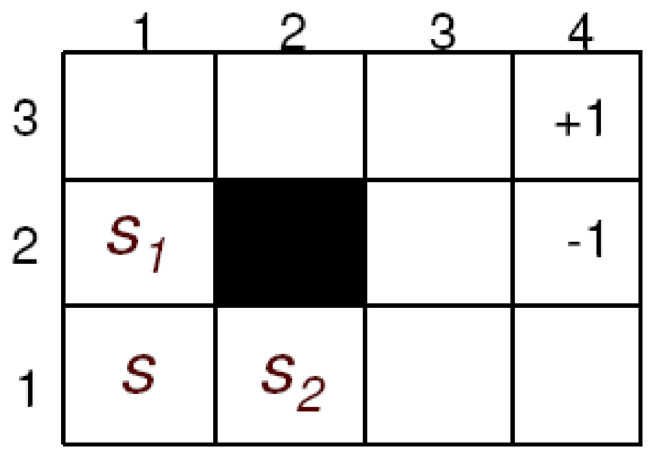
Model-Based Methods

- The C.E. methods are very similar to the Markov System case, except now do value-iteration-for-MDP backups

$$J^{est}(S_i) = \max_a \left[r_i^{est} + \gamma \sum_{S_j \in SUCCS(S_i)} P^{est}(S_j \mid S_i, a) J^{est}(S_j) \right]$$

	Space	Update Cost	Data Efficiency
Full C.E. Learning	$O(N_{sAo})$	$O(N_{sAo} k_{CRIT})$	🙂

Model Estimation: An Example



I observed a trajectory during which, when I moved Up from *s* = (1,1), I ended up in

s1 = (1,2) 10 times

s2 = (2,1) 2 times

P(s1 | s, Up) = 10/(10+2) = 0.83

P(s2 | s, Up) = 2/(10+2) = 0.17

But … how to explore the environment? We have not said how we generate the actions *a*

Exploration Strategy

- In principle, we can compute a current estimate of the best policy:

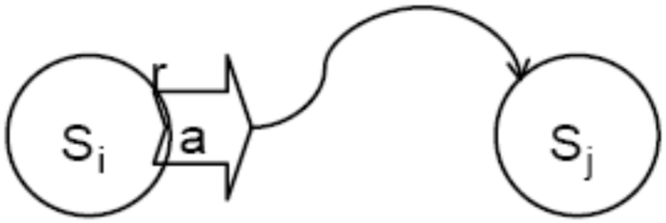
$$\pi^*(S) = \operatorname{argmax}_a \left(\sum_{S' \in Succ(S)} P(S' | S, a) J^{est}(S')\right)$$

- What is then the best strategy for exploration?
  - Greedy: Always use π*(*s*) when in state *s*?
  - Random
  - Mixed: ε-greedy
    - Choose the (current) best one with probability 1-ε
    - Choose another one randomly with probability ε

Model-Free Reinforcement Learning

Why not use T.D. ?

Observe



update

$$J^{est}(S_i) \leftarrow \alpha\left(r_i + \gamma J^{est}(S_j)\right) + (1-\alpha)J^{est}(S_i)$$

What's wrong with this?

Q-Learning: Model-Free Reinforcement Learning

Define

$Q^*(S_i,a)$= Expected sum of discounted future rewards if I start in state S_i, if I then take action a, and if I'm subsequently optimal

Question:

Define $J^*(S_i)$ in terms of Q^*

Q-Learning Update

Note that

$$Q^*(S,a) = r_i + \gamma \sum_{S_j \in SUCC(S_i)} P(S_j \mid S_i, a) \max_{a'} Q^*(S_j, a')$$

In Q-learning we maintain a table of Q^{est} values instead of J^{est} values … .

When you see $$\begin{tikzcd} S_i \arrow[r, bend left, "\text{reward}", "\text{action a}"'] & S_j \end{tikzcd}$$ do

$$Q^{est}(S_i, a) = \alpha\left[r_i + \gamma \max_{a'} Q^{est}(S_j, a')\right] + (1-\alpha) Q^{est}(S_i, a)$$

Policy estimation: $\pi(s) = \text{argmax}_a\, Q(s,a)$

Q-Learning: Convergence

- Q-learning guaranteed to converge to an optimal policy
- Very general procedure (because completely model-free)
- May be slow (because completely model free)

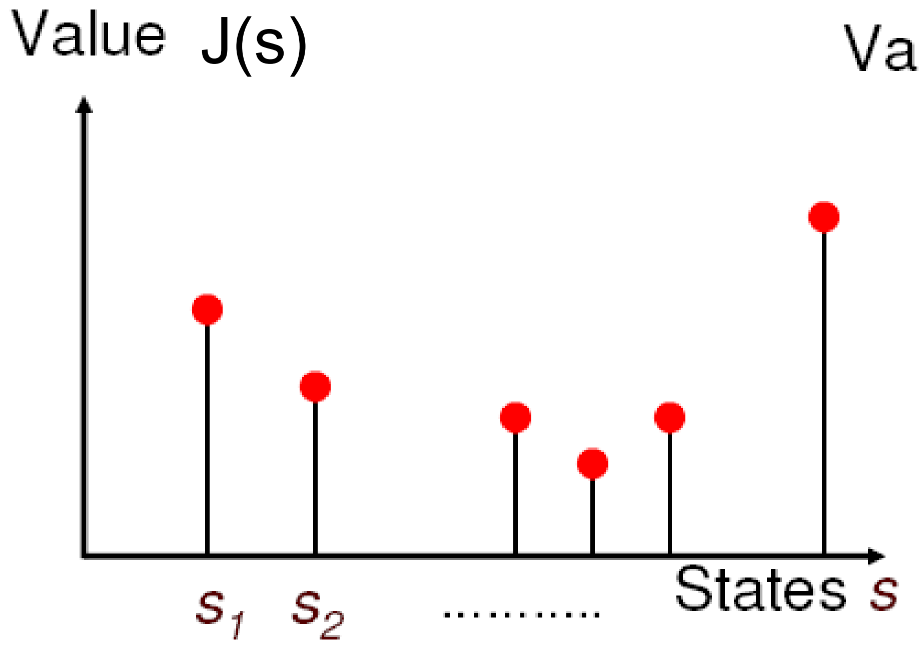
Q-Learning: Exploration Strategies

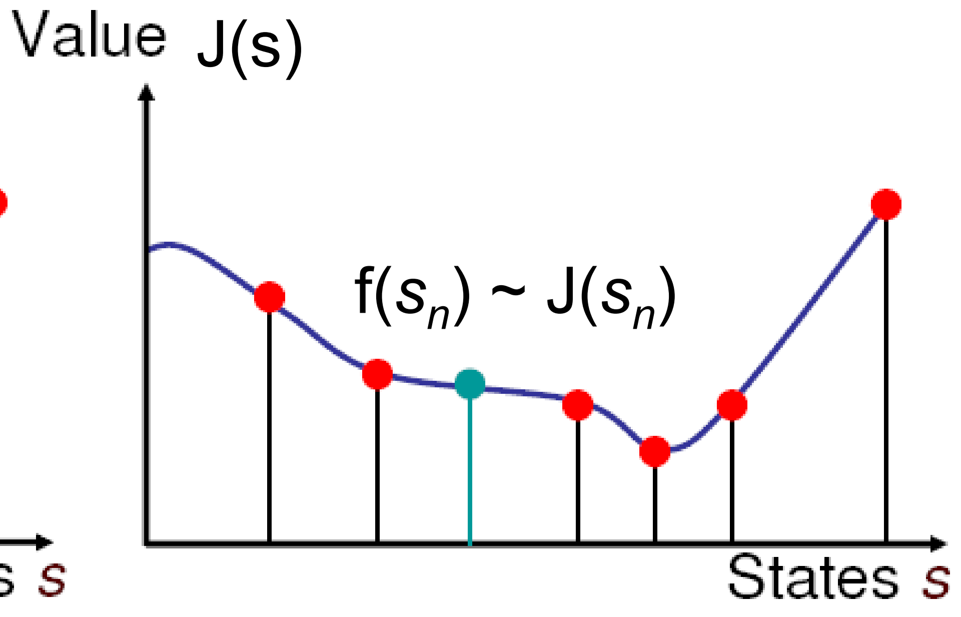
- How to choose the next action while we are learning?
  - Random
  - Greedy: Always choose the estimated best action $\pi(s)$
  - ε-Greedy: Choose the estimated best with probability $1-\varepsilon$

Generalization

- In real problems: Too many states (or state-action pairs) to store in a table
- Example: Backgammon → 10^{20} states!
- Need to:
  - Store J for a subset of states $\{s_1,..,s_K\}$
  - Generalize to compute J(s) for any other states s

Generalization



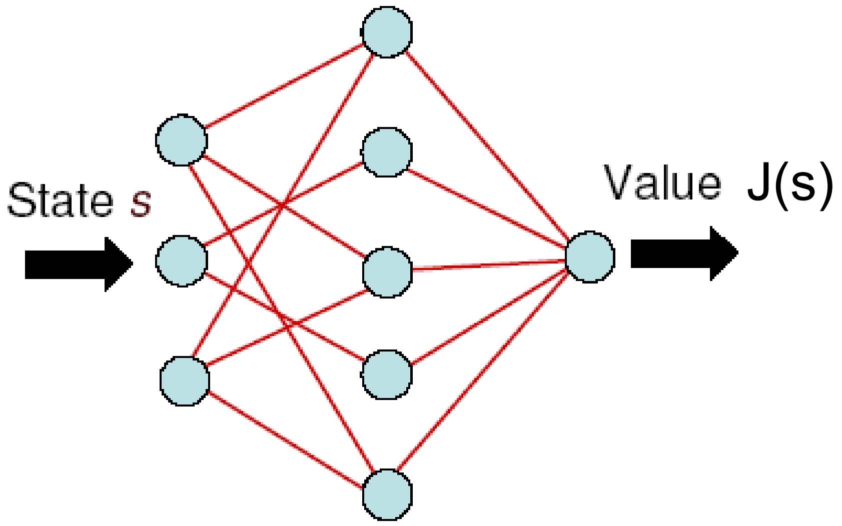


We have sample values of J for some of the states s_1, s_2

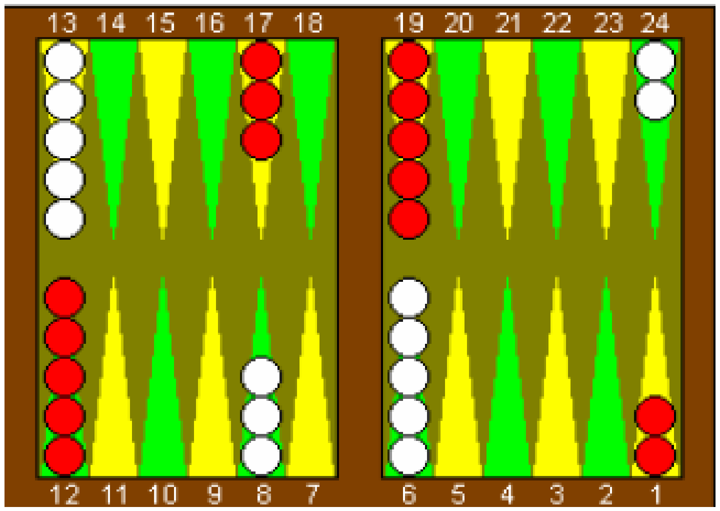
We interpolate a function f(.), such that for any query state s_n, $f(s_n)$ approximates $J(s_n)$

Generalization

- Possible function approximators:
  - Neural networks
  - Memory-based methods
- …… and many others solutions to representing J over large state spaces:
  - Decision trees
  - Clustering



Example: Backgammon



- States: Number of white and black checkers at each location
  → Order 10^{20} states!!!!
  → Branching factor prevents direct search
- Actions: Set of legal moves from any state

Example: Backgammon

- Represent mapping from states to expected outcomes by multilayer neural net
- Run a large number of “training games”
  - For each state *s* in a training game:
  - Update using temporal differencing
  - At every step of the game Choose best move according to current estimate of J
- Initially: Random moves
- After learning: Converges to good selection of moves

Performance

- Can learn starting with no knowledge at all!
- Example:
  - 1,500,000 training games
  - 80 hidden units
  - -1 pt/40 games (against world-class opponent)
- Enhancements use better encoding and additional hand-designed features

Summary

- Learning exploring environment and recording received rewards

- Model-Based techniques
  - Evaluate transition probabilities and apply previous MDP techniques to find values and policies

- Model-Free techniques
  - Temporal differencing for estimation of future rewards
  - Q-Learning for estimation of future rewards and optimal policy
  - Parameter: Learning rate should decay over iterations
  - Exploration strategies and selection of actions