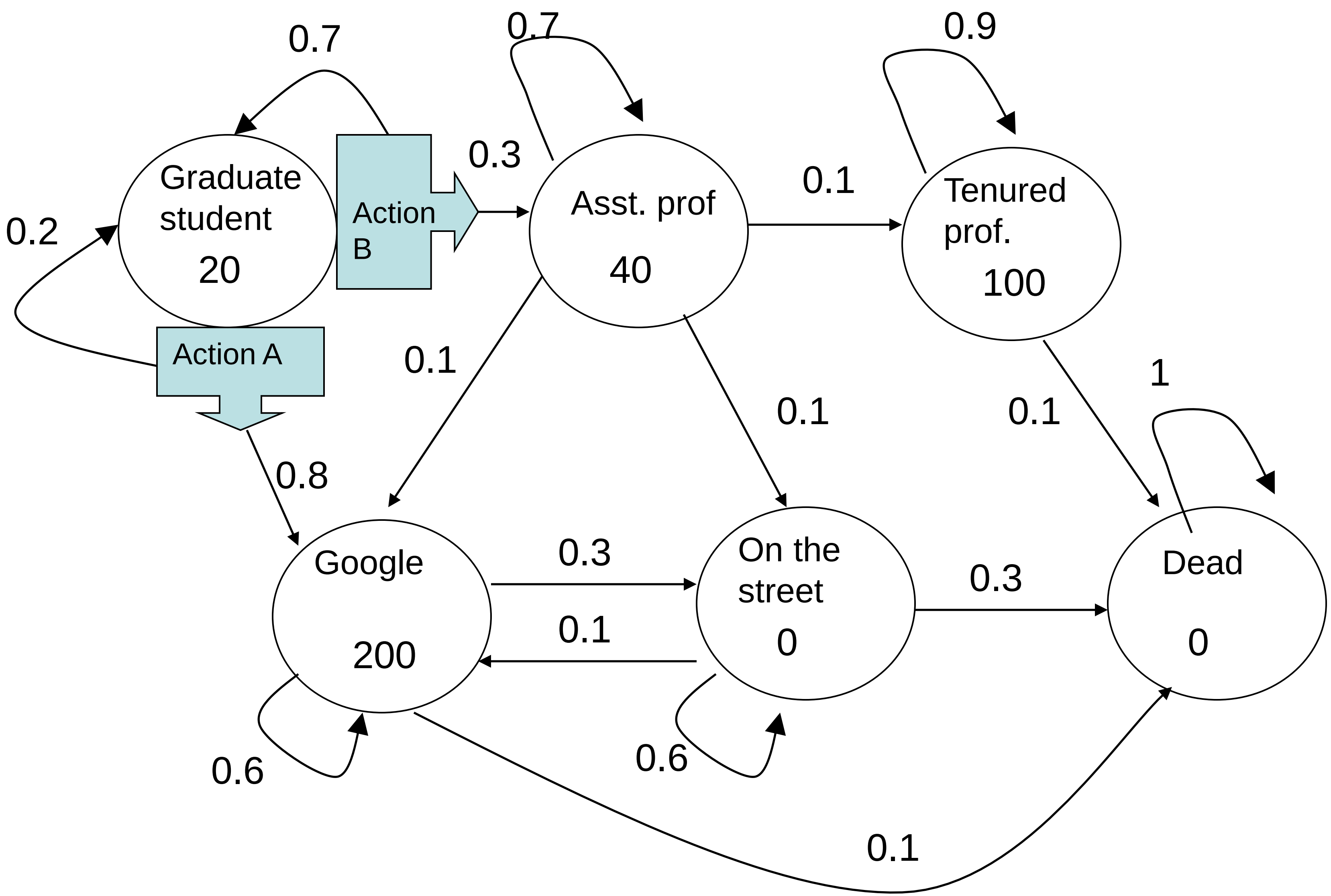
10-701
Machine Learning

Reinforcement learning (RL)

Markov decision process (MDP) with actions



Value computation

- An obvious question for such models is what is combined expected value for each state
- What can we expect to earn over our life time if we become Asst. prof.?
- What if we go to industry?

Before we answer this question, we need to define a model for future rewards:

- The value of a current award is higher than the value of future awards
  - Inflation, confidence
  - Example: Lottery

Discounted rewards

- The discounted rewards model is specified using a parameter γ
- Total rewards = current reward +

γ (reward at time t+1) +

γ^2 (reward at time t+2) +

………….

γ^k (reward at time t+k) +

infinite sum

Discounted awards

- The discounted award model is specified using a parameter γ
- Total awards = current award +

  γ (award at time t+1) +

  γ^2 (award at time t+2) +

  Converges if $0<\gamma<1$

) +

  infinite sum

Determining the total rewards in a state

- Define $J^*(s_i)$ = expected discounted sum of rewards when starting at state s_i
- How do we compute $J^*(s_i)$?

Factors expected pay for all possible transitions for step *i*

$$J^*(s_i) = r_i + \gamma X$$

$$= r_i + \gamma(p_{i1}J^*(s_1) + p_{i2}J^*(s_2) + \cdots p_{in}J^*(s_n))$$

How can we solve this?

Iterative approaches

- Solving in closed form is possible, but may be time consuming.
- It also doesn't generalize to non-linear models
- Alternatively, this problem can be solved in an iterative manner
- Lets define $J^t(s_i)$ as the expected discounted rewards after k steps
- How can we compute $J^t(s_i)$?

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

$$J^2(S_i) = r_i + \gamma\left(\sum_k p_{i,k} J^1(s_k)\right)$$

$$J^{t+1}(S_i) = r_i + \gamma\left(\sum_k p_{i,k} J^t(s_k)\right)$$

Iterative approaches

We know how to solve this!

Lets fill the dynamic programming table

- Lets define $J^k(s_i)$ as the expected discounted awards after k steps

But wait …

This is a never ending task!

$$J^2(S_i) = r_i + \gamma \left(\sum_k p_{i,k} J^1(s_k) \right)$$

$$J^{t+1}(S_i) = r_i + \gamma \left(\sum_k p_{i,k} J^t(s_k) \right)$$

When do we stop?

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

$$J^2(S_i) = r_i + \gamma\left(\sum_k p_{i,k} J^1(s_k)\right)$$

$$J^{t+1}(S_i) = r_i + \gamma\left(\sum_k p_{i,k} J^t(s_k)\right)$$

Remember, we have a converging function

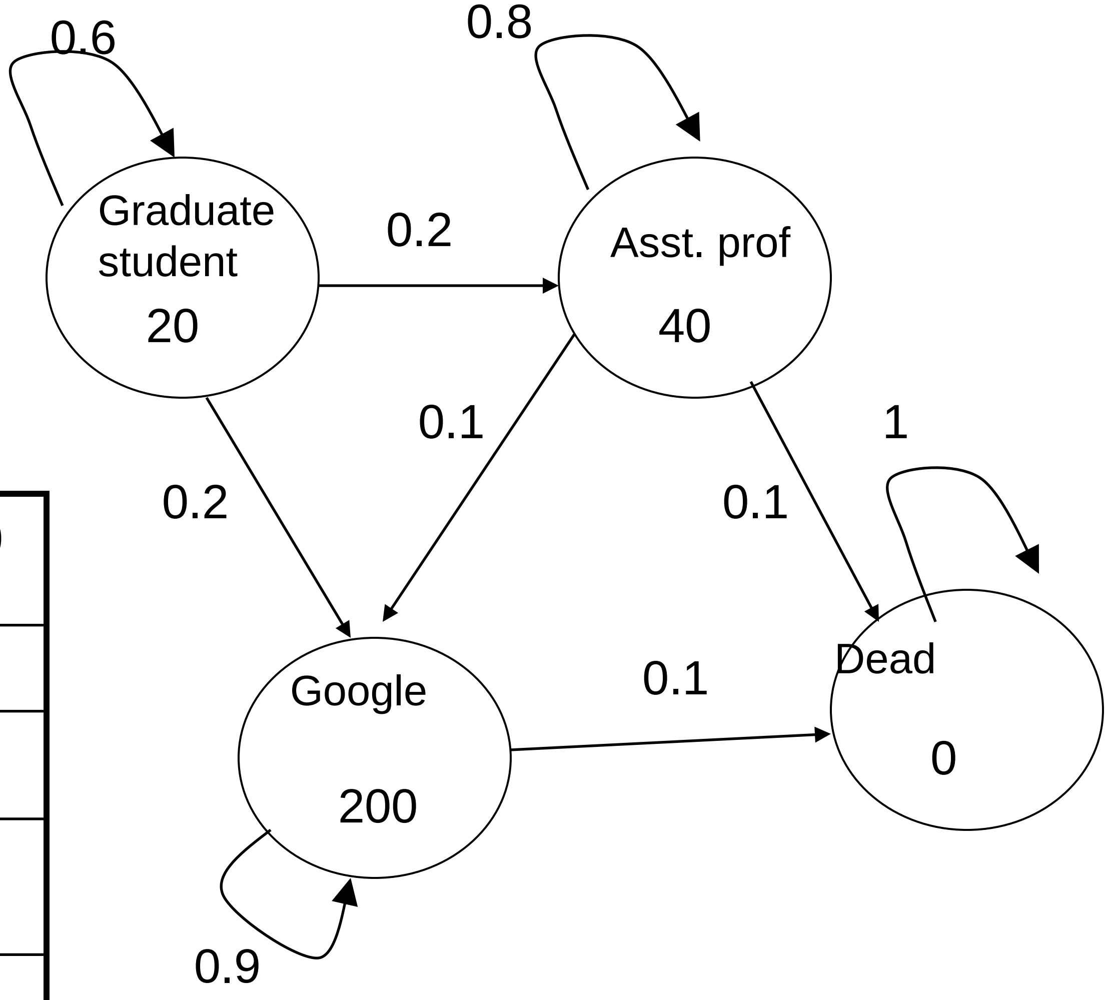
We can stop when $|J^{t-1}(s_i) - J^t(s_i)|_\infty < \varepsilon$

Infinity norm selects maximal element

Example for γ=0.9

$J^2(Gr)=20+0.9*(0.6*20+0.2*40+0.2*200)$

t	$J^t(Gr)$	$J^t(P)$	$J^t(Goo)$	$J^t(D)$
1	20	40	200	0
2	74	87	362	0
3	141	135	493	0
4	209	182	600	0



From MDPs to RL

- We still use the same Markov model with rewards and actions
- But there are a few differences:

  1. We do not assume we know the Markov model

  2. We adapt to new observations (online vs. offline)
- Examples:

  - Game playing

  - Robot interacting with enviroment

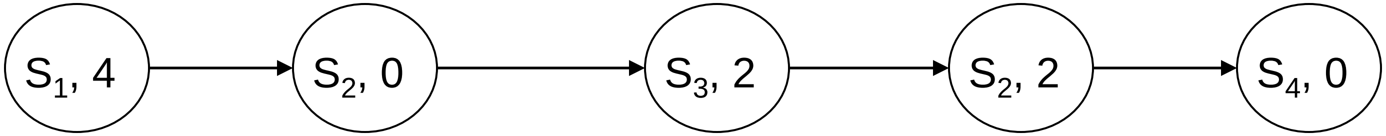
  - Agents

RL

- No actions
- With actions

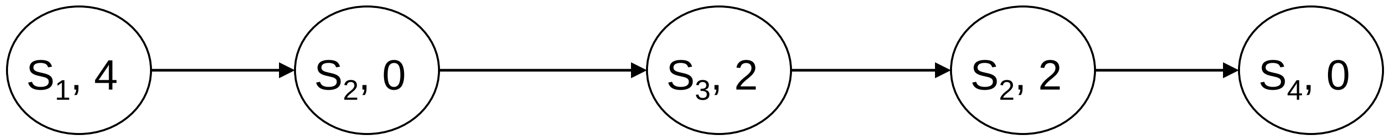
Scenario

- You wonder the world
- At each time point you see a state and a reward
- Your goal is to compute the sum of discounted rewards for each state



Scenario

- You wonder the world
- At each time point you see a state and a reward
- Your goal is to compute the sum of discounted rewards for each state
- We will denote these by $J^{est}(S_i)$



Discounted rewards: γ=0.9

- Lets compute the discounted rewards for each time point:

t1: $4 + 0.9*0 + 0.9^2*2 + 0.9^3*2 = 7.1$

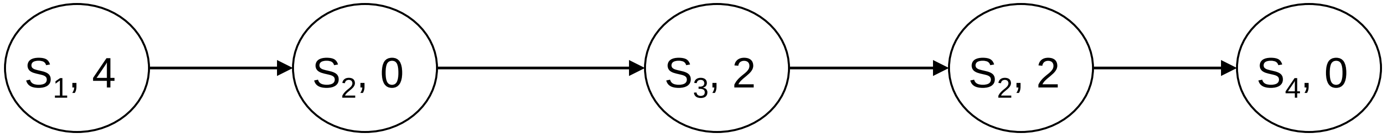
t2: $0 + 0.9*2 + 0.9^2*2 = 3.4$

t3: $2 + 0.9*2 = 3.8$

t4: $2 + 0 = 2$

t5: $0 = 0$

State	Observations	Mean
S_1	7.1	7.1
S_2	3.4, 2	2.7
S_3	3.8	3.8
S_4	0	0



Supervised learning for RL

- Type equation here.Observe set of states and rewards: (s(0),r(0)) …(s(T),r(T))
- For t=0 … T compute discounted sum:

$$J[t] = \sum_{i=t}^{T} \gamma^{i-t} r_i$$

- Compute $J^{est}(s_i)$ = (mean of J(t) for t such that s(t) = s_i)

$$\text{Jest}[s_i] = \frac{\sum_{t|s[t]=s_i} J[t]}{\# s[t] = s_i}$$

We assume that we observe each state frequently enough and that we have many observations so that the final observations do not have a big impact on our prediction

Algorithm for supervised learning

1. Initialize $Counts(s_i) = J(s_i)= Disc(s_i) = 0$
2. Observe a state s_i and a reward r
3. $Counts(s_i) = Counts(s_i) + 1$
4. $Disc(s_i) = Disc(s_i) + 1$
5. For all states j

   $J(s_j)= J(s_j) + r*Disc(s_j)$

   $Disc(s_j) = \gamma*Disc(s_j)$

6. Go to 2

At any time we can estimate J* by setting:
$J^{est}(s_i)= J(s_i) / Counts(s_i)$

Running time and space

- Each update takes O(n) where n is the number of states, since we are updating vectors containing entries for all states
- Space is also O(n)

1. Convergences to true J* can be proven

2. Can be more efficient by ignoring states for which Disc() is very low already.

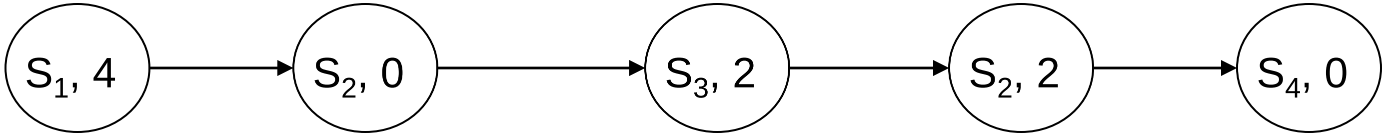
Problems with supervised learning

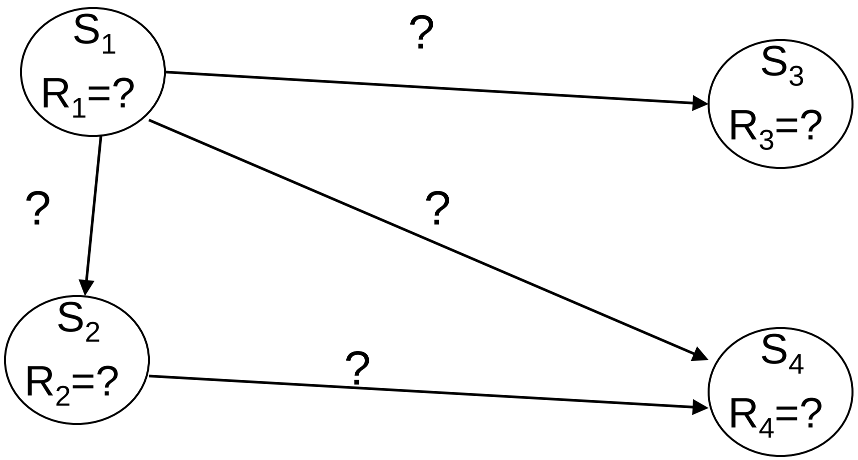
- Takes a long time to converge
- Does not use all available data

  - We can learn transition probabilities as well!

Certainty-Equivalent (CE) Learning

- Lets try to learn the underlying Markov system's parameters





CE learning

- We keep track of three vectors:

Counts(s): number of times we visited state s

J(s): sum of rewards from state s

Trans(i,j): number of time we transtiioned from state s_i to state s_j

- When we visit state s_i, receive reward r and move to state s_j we do the following:

$Counts(s_i) = Counts(s_i) + 1$

$J(s_i) = J(s_i) + r$

$Trans(i,j) = Trans(i,j) + 1$

CE learning

- When we visit state s_i, receive reward r and move to state s_j we do the following:

$Counts(s_i) = Counts(s_i) + 1$

$J(s_i) = J(s_i) + r$

$Trans(i,j) = Trans(i,j) + 1$

Using this we can estimate at any time the following parameters:

$R^{est}(s_i) = J(s_i)/Counts(s_i)$

$P^{est}(j|i) = Trans(i,j) / Counts(s_i)$

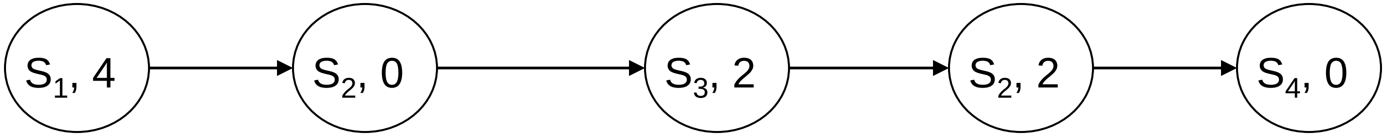
Example: CE learning

$P^{est}(j|i)$

	s1	s2	s3	s4
s1	0	1	0	0
s2	0	0	0.5	0.5
s3	0	1	0	0
s4	0	0	0	1

$R^{est}(s_i)$

State	Mean reward
S_1	4
S_2	1
S_3	2
S_4	0



CE learning

We can estimate at any time the following parameters:

$R^{est}(s_i) = J(s_i)/Counts(s_i)$

$P^{est}(j|i) = Trans(i,j) / Counts(s_i)$

We now basically have an estimated which we can solve for all states s_k:

$$J^{est}(s_k) = r^{est}(s_k) + \gamma \sum_j p^{est}(s_j \mid s_k) J^{est}(s_j)$$

CE: Run time and space

Run time

- Updates: O(1)
- Solving MDP:

  - $O(n^3)$ using matrix inversion

  - $O(n^2 * \#it)$ when using value iteration

Space

- $O(n^2)$ for transition probabilities

Improving CE: One backup

- We do the same updates and estimates as the original CE:

$Counts(s_i) = Counts(s_i) + 1$

$J(s_i) = J(s_i) + r$

$Trans(i,j) = Trans(i,j) + 1$

$R^{est}(s_i) = J(s_i)/Counts(s_i)$

$P^{est}(j|i) = Trans(i,j) / Counts(s_i)$

- But we do not carry out the full value iteration
- Instead, we **only** update $J^{est}(s_i)$ for the current state:

$$J^{est}(s_i) = r^{est}(s_i) + \gamma \sum_j p^{est}(s_j \mid s_i) J^{est}(s_j)$$

CE one backup: Run time and space

Run time

- Updates: O(1)
- Solving MDP:

  - O(1) just update current state

Space

- O(n^2) for transition probabilities

- Still a lot of memory, but much more efficient
- Can prove convergence to optimal solution (but slower than CE)

Summary so far

- Three methods

Method	Time	Space
Supervised learning	O(n)	O(n)
CE learning	$O(n^2 * \#it)$	$O(n^2)$
One backup CE	O(1)	$O(n^2)$

Temporal difference (TD) learning

- Goal: Same efficiency as one backup CE while much less space
- We only maintain the J^{est} array.
- Assume we have $J^{est}(s_1)$ … $J^{est}(s_n)$. If we observe a transition from state s_i to state s_j and a reward r, we update using the following rule:

$$J^{est}(s_i) = (1-\alpha) J^{est}(s_i) + \alpha(r + \gamma J^{est}(s_j))$$

Temporal difference (TD) learning

- Assume we have $J^{est}(s_1)$ … $J^{est}(s_n)$. If we observe a transition from state s_i to state s_j and a reward r, we update using the following rule:

$$J^{est}(s_i) = (1-\alpha)J^{est}(s_i) + \alpha(r + \gamma j^{est}(s_j))$$

parameter to determine how much weight we place on current observation

We have seen similar update rule before, as always, choosing α is an issue

Convergence

- TD learning is guaranteed to converge if:
- All states are visited often
- And:

$$\sum_t \alpha_t = \infty$$

$$\sum_t \alpha_t^2 < \infty$$

For example, α_t=C/t for some constant C would satisfy both requirements

TD: Complexity and space

- Time to update: O(1)
- Space: O(n)

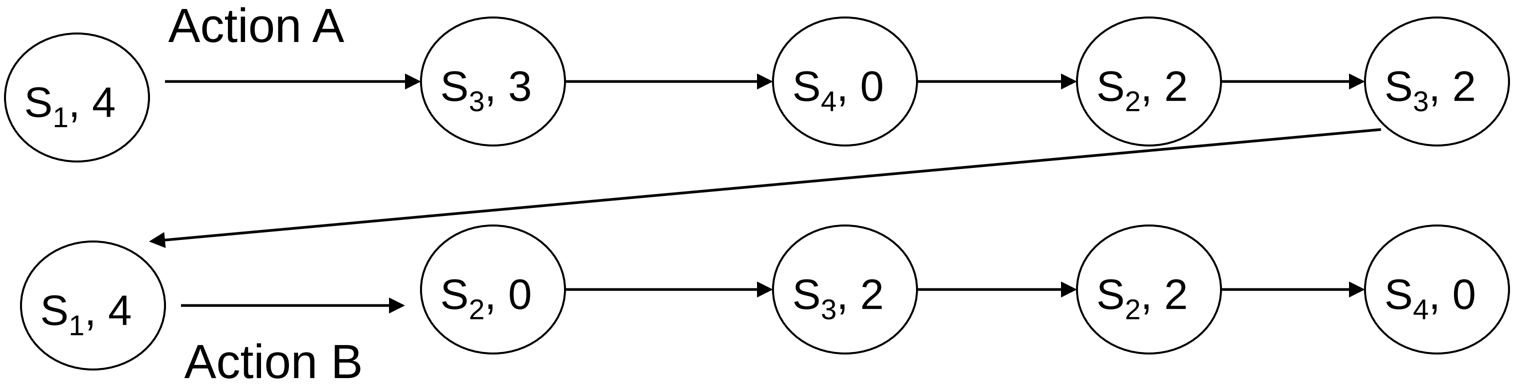
Method	Time	Space
Supervised learning	O(n)	O(n)
CE learning	$O(n^2 * \#it)$	$O(n^2)$
One backup CE	O(1)	$O(n^2)$

RL

- No actions √
- With actions

Policy learning

- So far we assumed that we cannot impact the outcome transition.
- In real world situations we often have a choice of actions we take (as we discussed for MDPs).
- How can we learn the best policy for such cases?



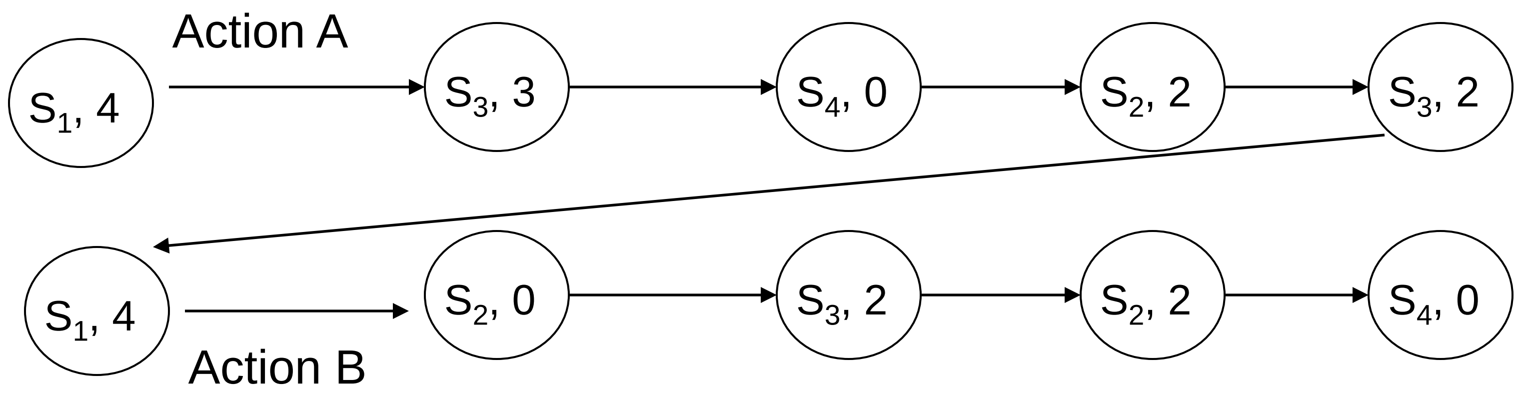
Policy learning using CE : Example

$R^{est}(s_i)$

State	Mean reward
S_1	4
S_2	4/3
S_3	2.5
S_4	0

$P^{est}(j|i,a)$

	s1	s2	s3	s4
s1,A	0	0	1	0
s1,B	0	1	0	0
s2	0	0	2/3	1/3
s3	1/3	1/3	0	1/3
s4	0	1	0	0



Policy learning using CE

We can easily update CE by setting:

$$J^{est}(s_k) = r^{est}(s_k) + \max_a \left[\gamma \sum_j p^{est}(s_j \mid s_k, a) J^{est}(s_j) \right]$$

We revise our transition model to include actions

Policy learning for TD

- TD is model free
- We can adjust TD to learn policies by defining the Q function:
- $Q^*(s_i,a)$ = expected sum of future (discounted) rewards if we start at state s_i and take action a
- When we take a specific action a in state s_i and then transition to state s_j we can update the Q function directly by setting:

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

Instead of the J^{est} vector we maintain the Q^{est} matrix, which is a rather sparse n by m matrix (n states and m actions)

Choosing the next action

- We can select the action that results in the highest expected sum of future rewards
- But that may not be the best action. Remember, we are only sampling from the distribution of possible outcomes. We do not want to avoid potentially beneficial actions.
- Instead, we can take a more probabilistic approach:

$$p(a) = \frac{1}{Z}\exp\left(-\frac{Q^{est}(s_i, a)}{f(t)}\right)$$

The probability we will use action a

Normalizing constant

Decreases as time goes by and we are more confident in the model we learned

Choosing the next action

- Instead, we can take a more probabilistic approach:

$$p(a) \propto \exp(-\frac{Q^{est}(s_i, a)}{f(t)})$$

- We can initialize Q values to be high to increase the likelihood that we will explore more options
- It can be shown that Q learning converges to optimal policy

What you should know

- Strategies for computing with expected rewards
- Strategies for computing rewards and actions
- Q learning