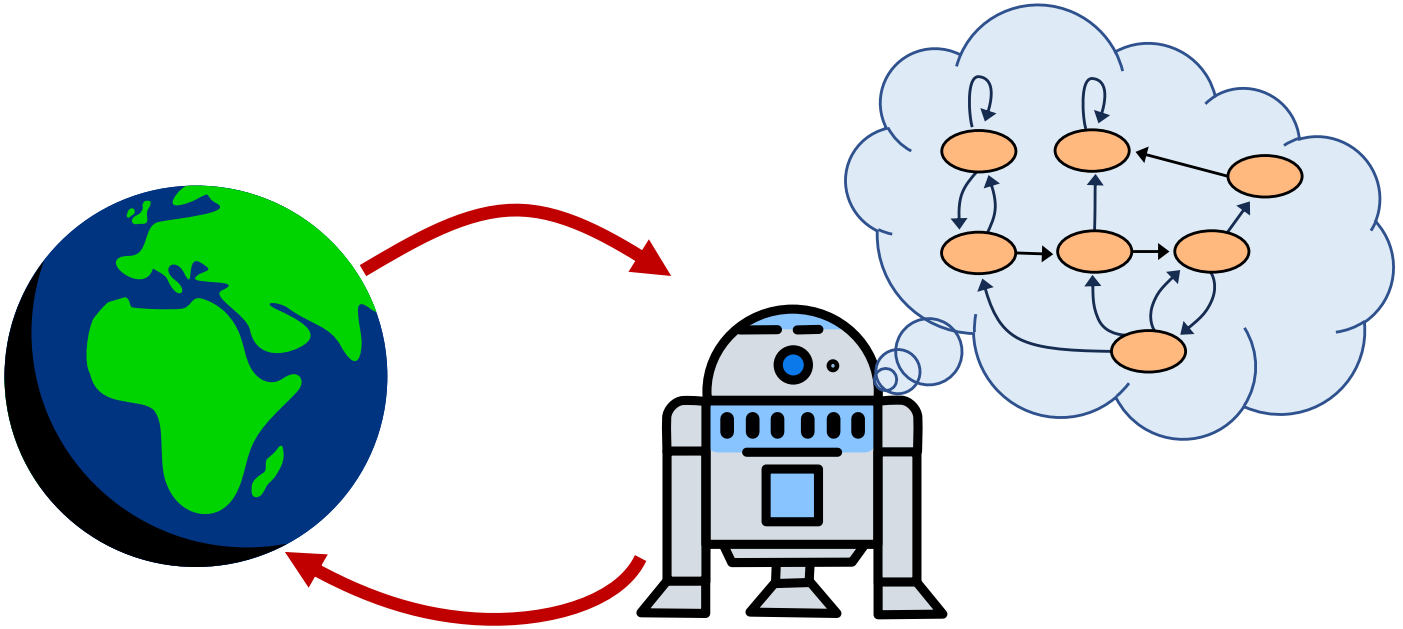
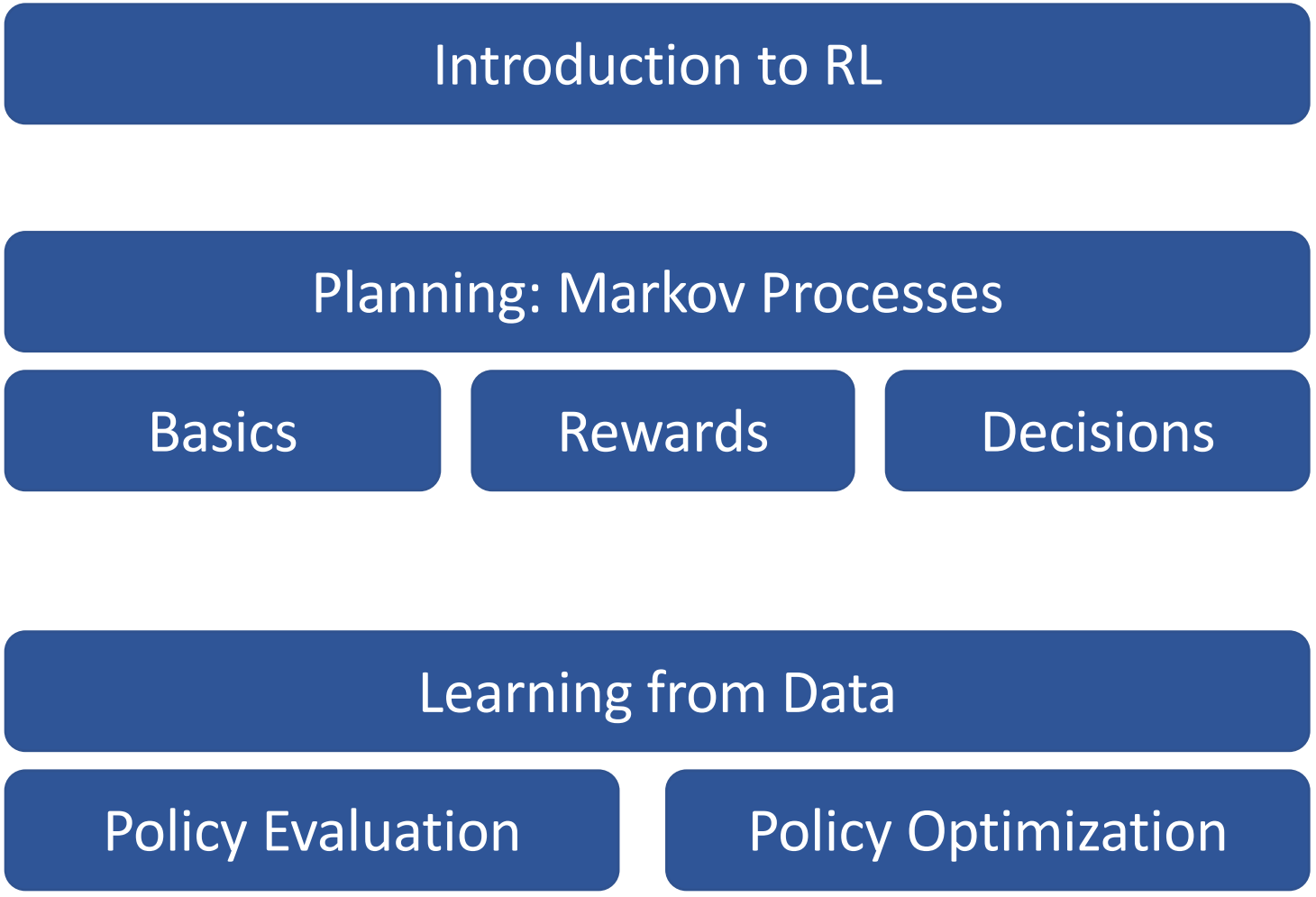
CS178:
Reinforcement Learning



Prof. Alexander Ihler

Fall 2023

Reinforcement Learning



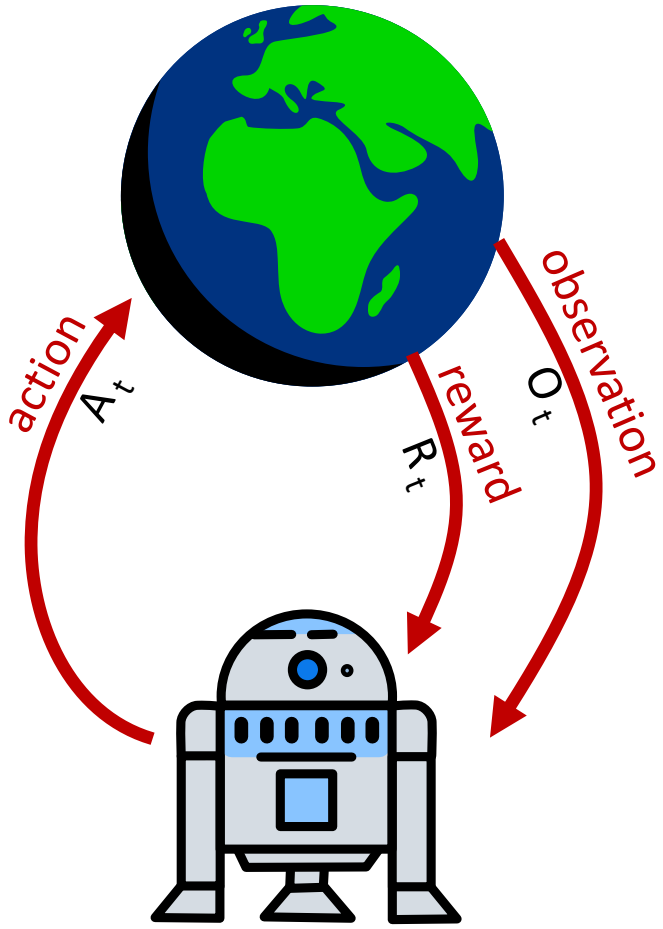
What makes RL different?

- Compared with supervised learning:
  - No direct supervision; just rewards
  - Feedback is delayed, not instantateous
  - Fundamentally temporal: data are sequential
  - Actions affect what data we receive after

Examples

- Fly stunt maneuvers in a helicopter
- Defeat the world champion at Backgammon or Go
- Manage an investment portfolio
- Control a power station
- Make a humanoid robot walk
- Play many different Atari games better than humans

Agent-Environment Interface



Agent

- decides on an action
- receives next observation
- receives next reward

Environment

- executes the action
- computes next observation
- computes next reward

Reward, R_t

- How well are we doing?
  - May be positive (good), or negative (bad)
  - Nothing about **why**: may have little to do with A_{t-1}!
- Agent wants to maximize cumulative reward

- Examples:

Autonomous stunt helicopter
+ Follow trajectory
– Crash

image: [Ng et al. 2004]

PvP Strategy Game
+ Win the game

Real-time video game
+ Earn points
– Lose lives

Manage a stock portfolio
+ Earn $$$

Sequential Decision Making

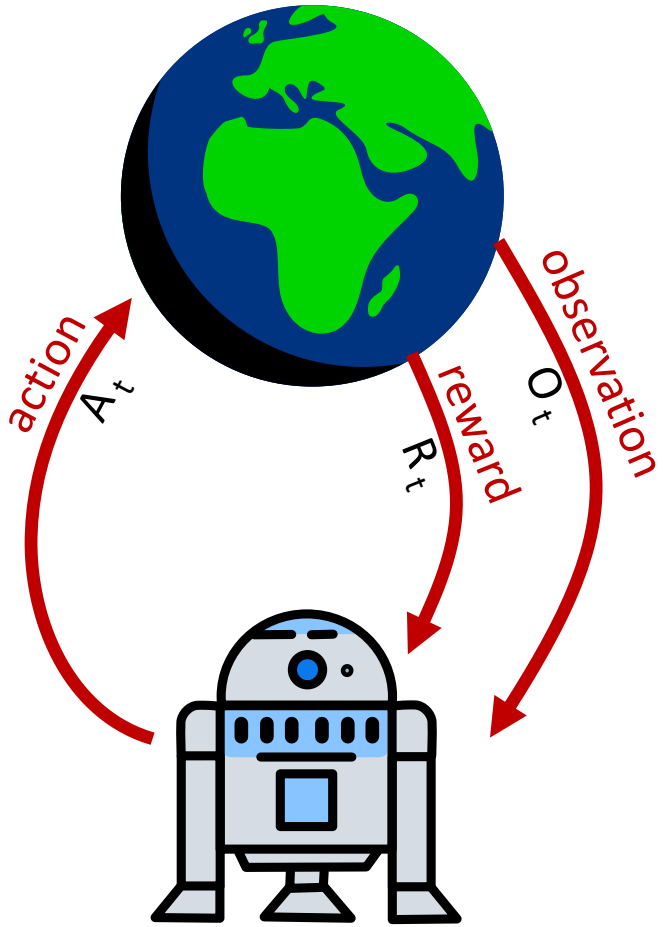
- RL tasks are typically sequential by nature
  - Actions taken have long-term consequences
  - Rewards may be delayed
  - May be better to sacrifice immediate reward for long-term benefits

Examples

- A financial investment (may take months to mature)
- Refueling a helicopter (might prevent a crash later)
- Blocking opponent moves (might eventually help win)
- Spend a lot of money and go to college (earn more later)
- Don't commit crimes (rewarded by not going to jail)
- Get started on final project early (avoid stress later)

A key aspect of intelligence: How far ahead are you able to plan?

Reinforcement Learning



Environment is given to us:

- Aerodynamics
- Game rules
- Market participants

… determines observations & rewards

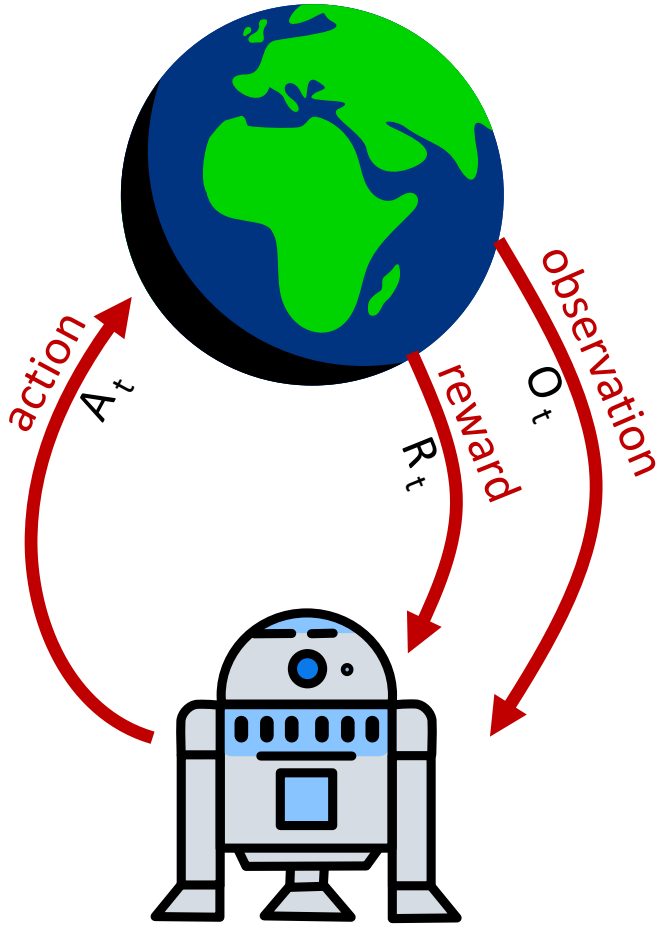
Agent is ours to design:
select actions to maximize total rewards

What can our choice of action depend on?

- Ignore observations $\{O_t\}$ completely?
- Most recent Ot enough? Or, $[O_t, A_t]$?
- Last few observations, $[O_t, O_{t-1}, O_{t-2}]$?
- Need all observations so far, $[O_t \ldots O_1]$?

Agent State, S_t

- What should our choice of action depend on?



History: everything that has happened so far

$$H_t = O_1R_1A_1O_2R_2A_2O_3R_3,\ldots,A_{t-1}O_tR_t$$

State, S_t can be

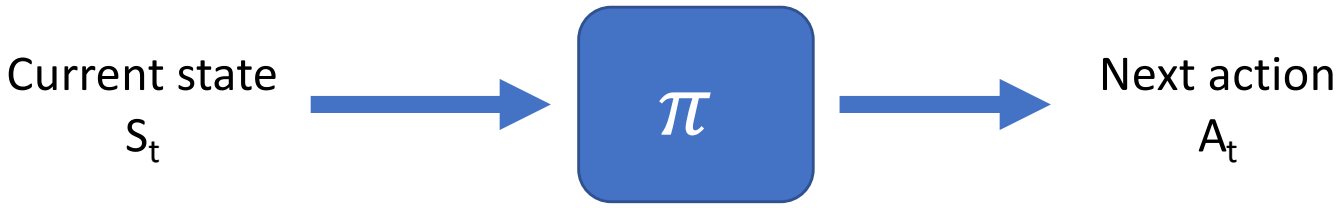
- O_t
- O_tR_t
- $A_{t-1}O_tR_t$
- $O_{t-3}O_{t-2}O_{t-1}O_t$

In general, $S_t = f(H_t)$

You, as AI designer, specify this function

Agent Policy, π

- Our policy maps the current state to an action:



Deterministic Policy: $A_t = \pi(S_t)$

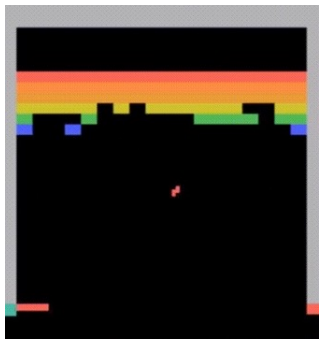
Stochastic Policy: $\pi(a|s) = P(A_t = a|S_t = s)$

Good policy: Leads to larger cumulative reward

Bad policy: Leads to worse cumulative reward

(we will explore this later)

Example: Atari Breakout!



Rules are unknown

- What makes the score increase?

Dynamics are unknown

- How do actions change pixels?

Example: Atari Breakout!

10 min training
"basics"

https://www.youtube.com/watch?v=V1eYniJ0Rnk

Example: Atari Breakout!

10 min training
"basics"

120 min training
"tactics"

https://www.youtube.com/watch?v=V1eYniJ0Rnk

Example: Atari Breakout!

10 min training
"basics"

120 min training
"tactics"

240 min training
"strategy"

https://www.youtube.com/watch?v=V1eYniJ0Rnk

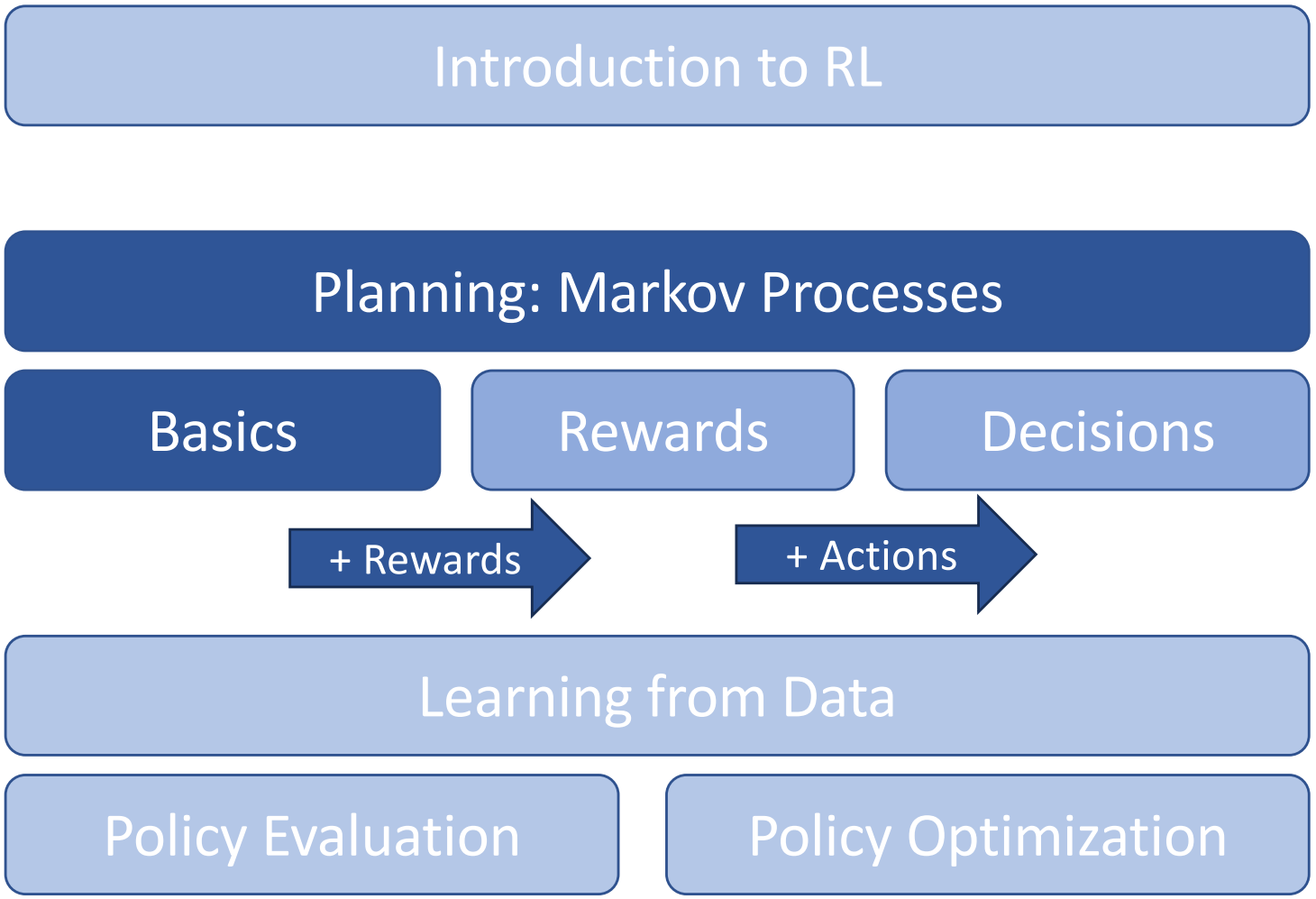
Example: Robotic Soccer

https://www.youtube.com/watch?v=CIF2SBVY-J0

AlphaGo

https://www.youtube.com/watch?v=I2WFvGl4y8c

Reinforcement Learning



Markov Property

"The future is independent of the past, given the present"

- A state S_t is **Markov** if and only if,

$$p(S_{t+1}|S_t, S_{t-1}, \ldots, S_1) = p(S_{t+1}|S_t)$$

  - The state S_t captures all relevant information from the history
  - So, if we know S_t, we can discard the rest of the history
  - S_t is a *sufficient statistic* of the system's future behavior

State Transition Matrix

- For a Markov state s, and successor state s', the state transition probability is defined by

$$\mathbf{P}_{ss'} = p(S_{t+1} = s' \mid S_t = s)$$

- For *n* discrete states, the state transition matrix P defines probabilities from each state s to all successor states s':

$$\mathbf{P} = \begin{matrix} \\ \text{(from)} \end{matrix} \begin{bmatrix} \mathbf{P}_{11} & \cdots & \mathbf{P}_{1n} \\ \vdots & \ddots & \vdots \\ \mathbf{P}_{n1} & \cdots & \mathbf{P}_{nn} \end{bmatrix}$$

(to)

  - So, each row of the matrix sums to one

Markov Processes

- A memoryless random process:
  - A sequence of random states S_1, S_2, … with the Markov property
  - Discrete states: basically, a finite state machine with randomness

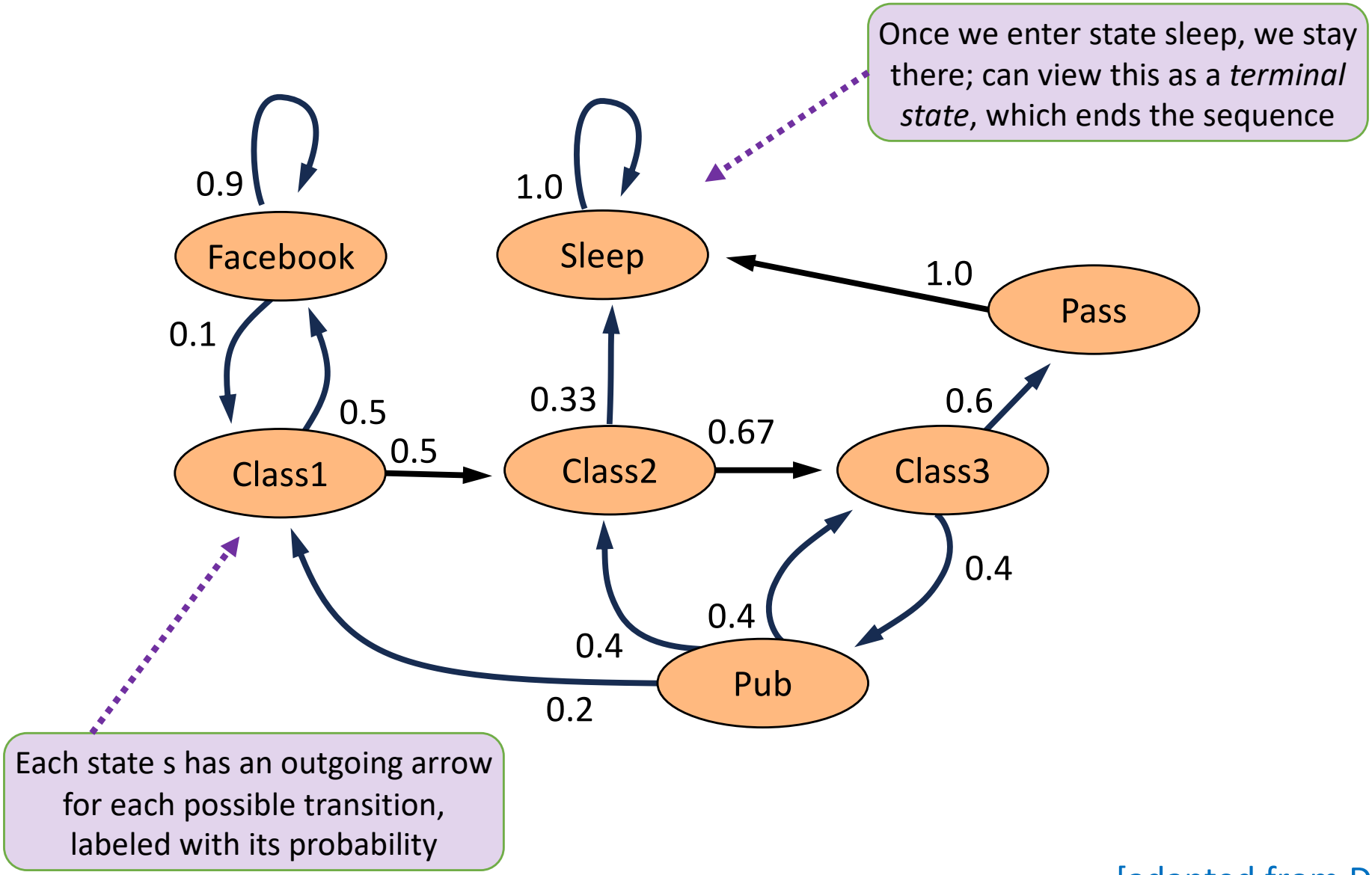
Definition

A Markov Process (or Markov Chain) is a tuple [**S**, **P**]:

- **S** is a (finite) set of states
- **P** is a state transition probability matrix,

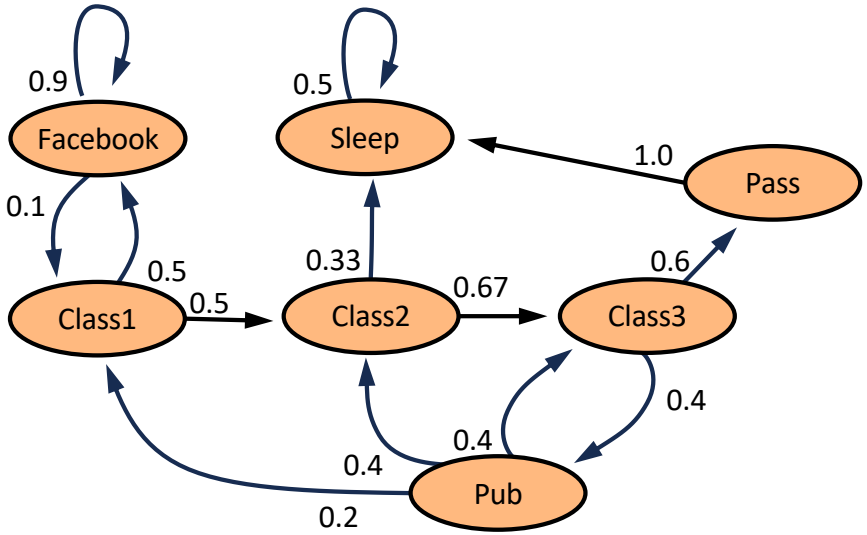
$$\mathbf{P}_{ss'} = p(S_{t+1} = s' \mid S_t = s)$$

Ex: University Life Markov Chain



[adapted from D. Silver]

Ex: University MC: Episodes

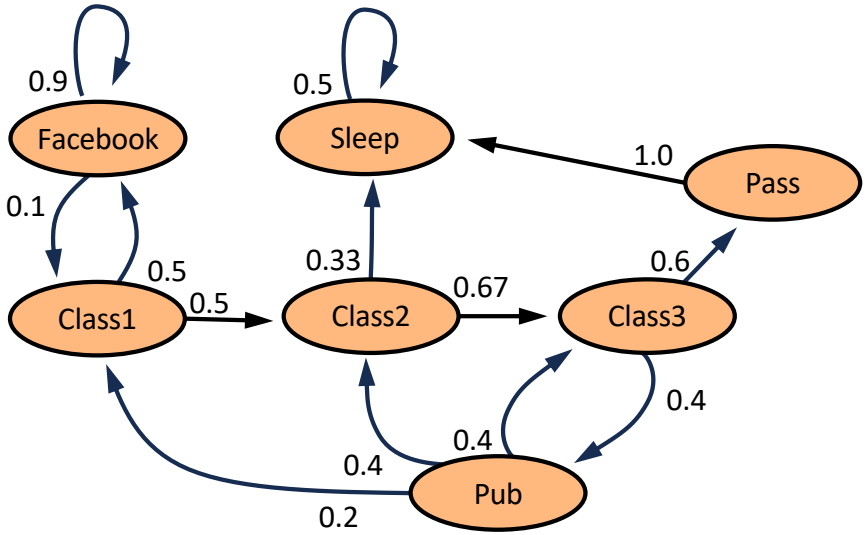


We can sample episodes for our Markov chain, starting from initial state S_1=C1

Ex:

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

Ex: University MC: Transition Matrix



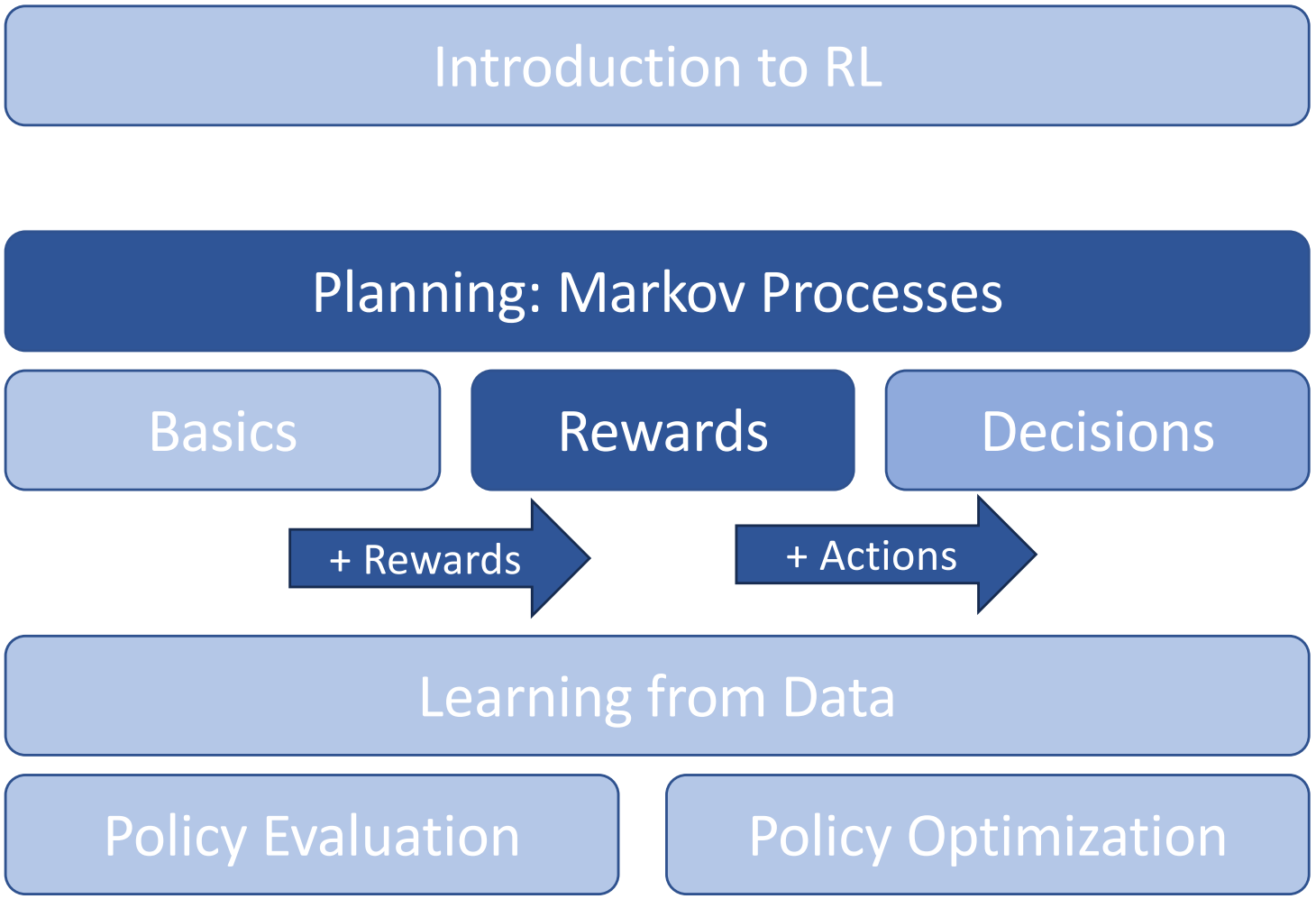
$$\mathbf{P}_{ss'} = p(S_{t+1} = s' | S_t = s) :$$

s \ s'	Fa	C1	C2	C3	Pu	Pa	Sl
Fa	0.9	0.1					
C1	0.5		0.5				
C2				0.67			0.33
C3					0.4	0.6	
Pu		0.2	0.4	0.4			
Pa							1.0
Sl							1.0

Ex: [sequence] : [probability of that sequence]

- C1 C2 C3 Pass Sleep : (0.5) (0.67) (0.6) (1.0)
- C1 FB FB C1 C2 Sleep : (0.5) (0.9) (0.1) (0.5) (0.3))
- C1 C2 C3 Pub C2 C3 Pass Sleep : (0.5) (0.67) (0.4) (0.4) (0.67) (0.6) (1.0)
- …

Reinforcement Learning



Markov Reward Process

- A Markov Process with reward values:

Definition

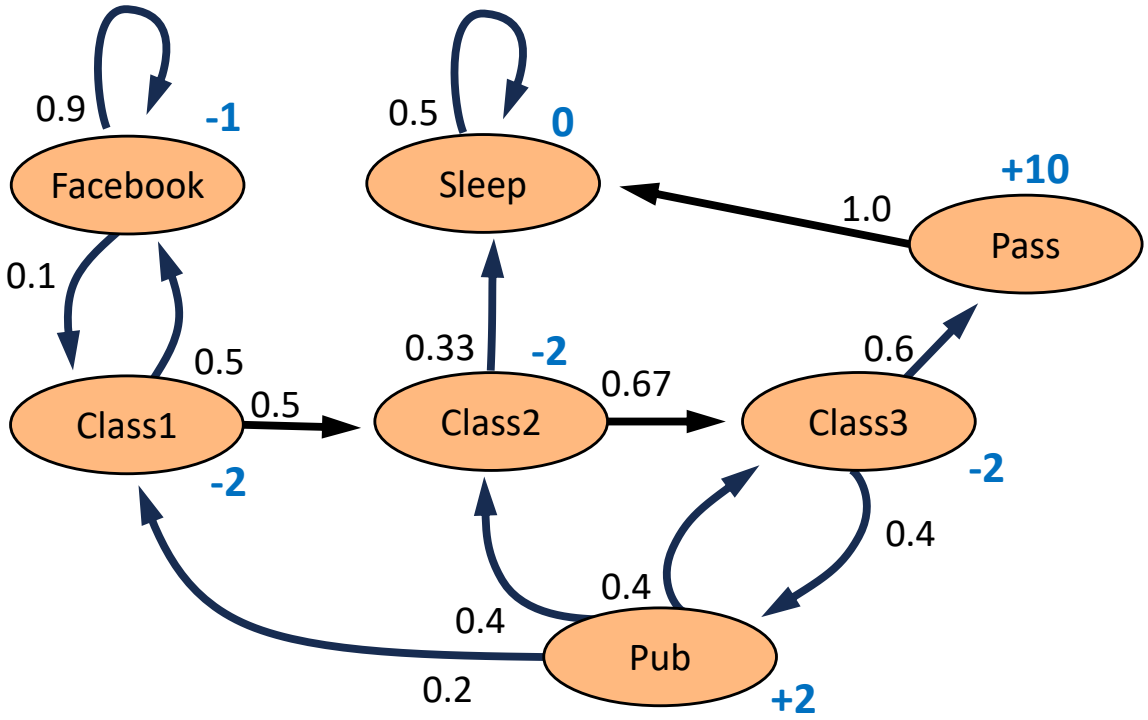
A Markov Reward Process is a tuple [**S**, **P, R,** γ]:

- **S** is a (finite) set of states
- **P** is a state transition probability matrix,

$$\mathbf{P}_{ss'} = p(S_{t+1} = s' \,|\, S_t = s)$$

- **R** is a reward function, $\mathbf{r}_s = \mathbb{E}\big[R_{t+1} \,|\, S_t = s\big]$
- γ is a discount factor, $\gamma \in [0, 1]$

Ex: University Life MRP



$\mathbf{r}_s = \mathbb{E}[R_{t+1}|S_t = s]$:

s	Fa	C1	C2	C3	Pu	Pa	Sl
	-1	-2	-2	-2	+2	+10	0

$\mathbf{P}_{ss'} = p(S_{t+1} = s'|S_t = s)$:

s \ s'	Fa	C1	C2	C3	Pu	Pa	Sl
Fa	0.9	0.1					
C1	0.5		0.5				
C2				0.67			0.33
C3					0.4	0.6	
Pu		0.2	0.4	0.4			
Pa							1.0
Sl							1.0

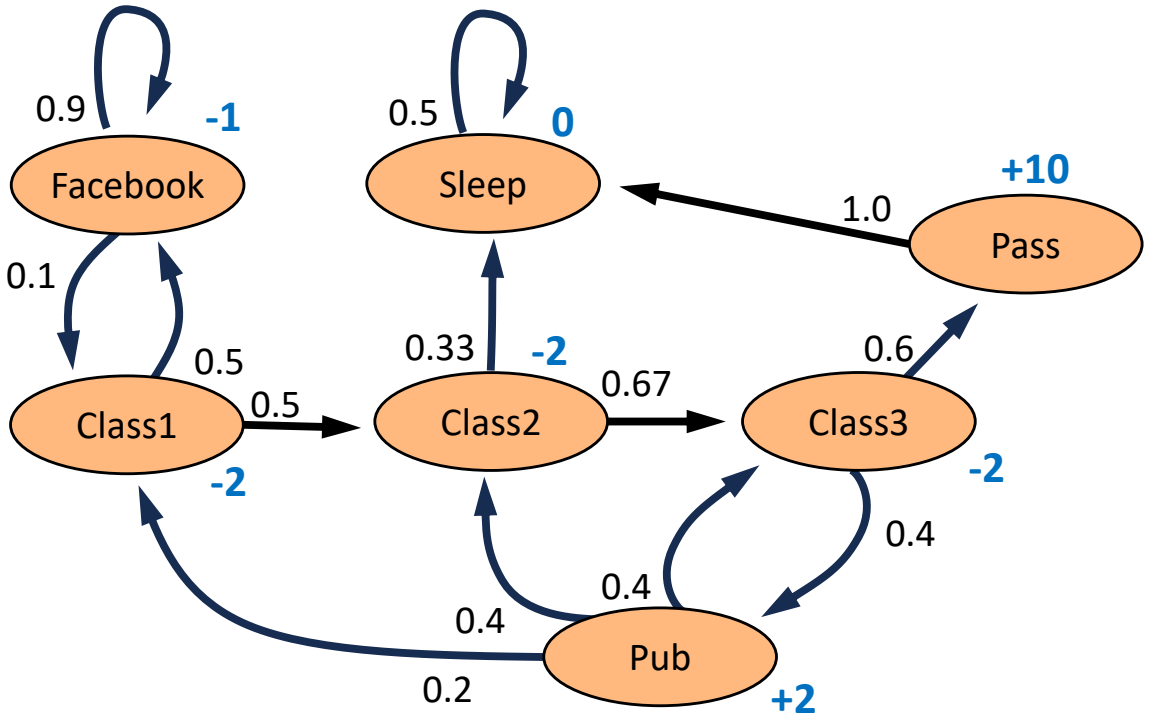
Return

- The return, G_t, is the total discounted reward from time t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

  - Depends (implicitly) on the states we traverse
  - Discount factor γ determines the present value of future rewards
  - Value of getting reward R after (k+1) time steps is γ^k R
  - Smaller γ values immediate rewards over delayed rewards:
    - If γ is close to zero, "myopic" evaluation
    - If γ is close to one, "far-sighted" evaluation

Ex: University Life MRP: Returns



$\mathbf{r}_s = \mathbb{E}[R_{t+1}|S_t = s]$:

s	Fa	C1	C2	C3	Pu	Pa	Sl
	-1	-2	-2	-2	+2	+10	0

Ex: [sequence] : [return of that sequence]

- C1 C2 C3 Pass Sleep : (-2) + (.5)(-2) + (.25)(-2) + (.125)(+10) = -2.25
- C1 FB FB C1 C2 Sleep : (-2) + (.5)(-1) + (.25)(-1) + … = -3.125
- C1 C2 C3 Pub C2 C3 Pass Sleep : (-2) + (.5)(-2) + (.25)(-2) + … = -3.41
- …

Why discount?

Typically, MRPs / MDPs are discounted!

- Geometric discounting is mathematically convenient
- Guarantees finite return values, even for infinite sequences
- Real, pyschological preference for immediate reward
  - Prefer $20 today, or in 20 years?
  - Future rewards "worth less" than same reward right now?

[Mischel & Ebbesen, 1970]

Why discount?

Typically, MRPs / MDPs are discounted!

- Geometric discounting is mathematically convenient
- Guarantees finite return values, even for infinite sequences
- Real, pyschological preference for immediate reward
  - Prefer $20 today, or in 20 years?
  - Future rewards "worth less" than same reward right now?
- May have (unmodeled) uncertainty about the future
  - What if I die of boredom in class?
- In some settings, undiscounted ($\gamma = 1$) is OK
  - E.g., if all sequences eventually terminate: blackjack, PacMan, …

Value Function

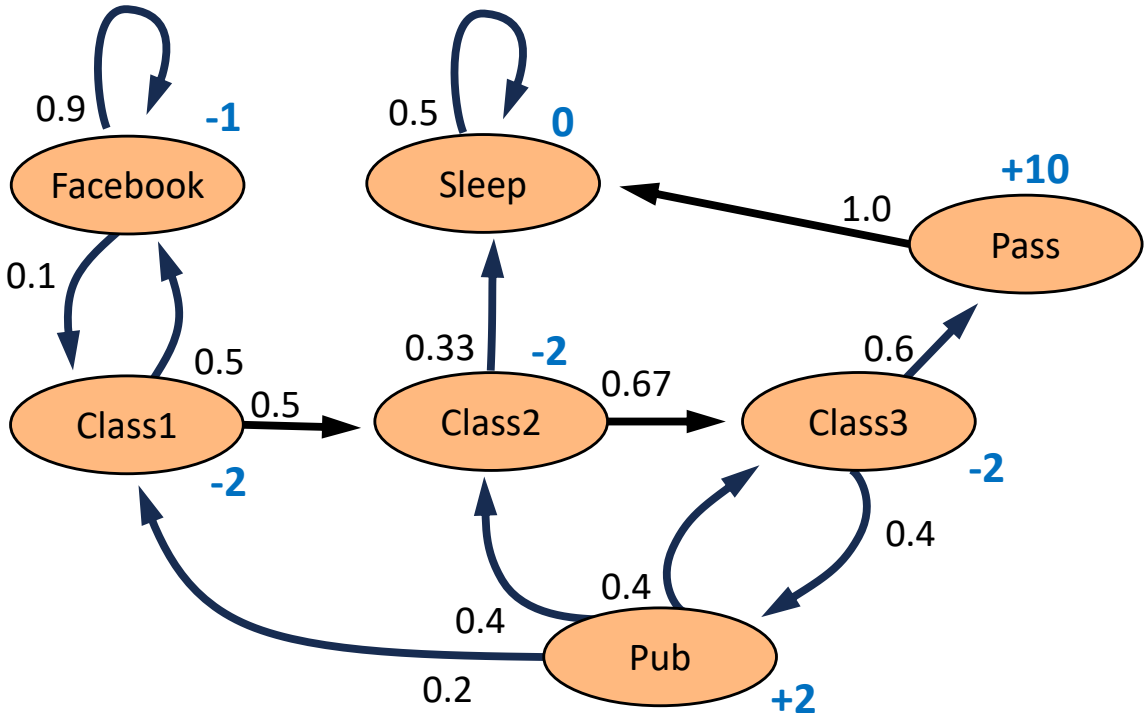
- What is the (long-term) value of getting to state s?

- The **state value function** v(s) of a MRP is the expected return, starting from state s:

$$v(s) = \mathbb{E}\big[G_t \mid S_t = s\big]$$

  - Expectation over state trajectory S_{t+1}, S_{t+2}, … and rewards R_{t+1}, R_{t+2}, …

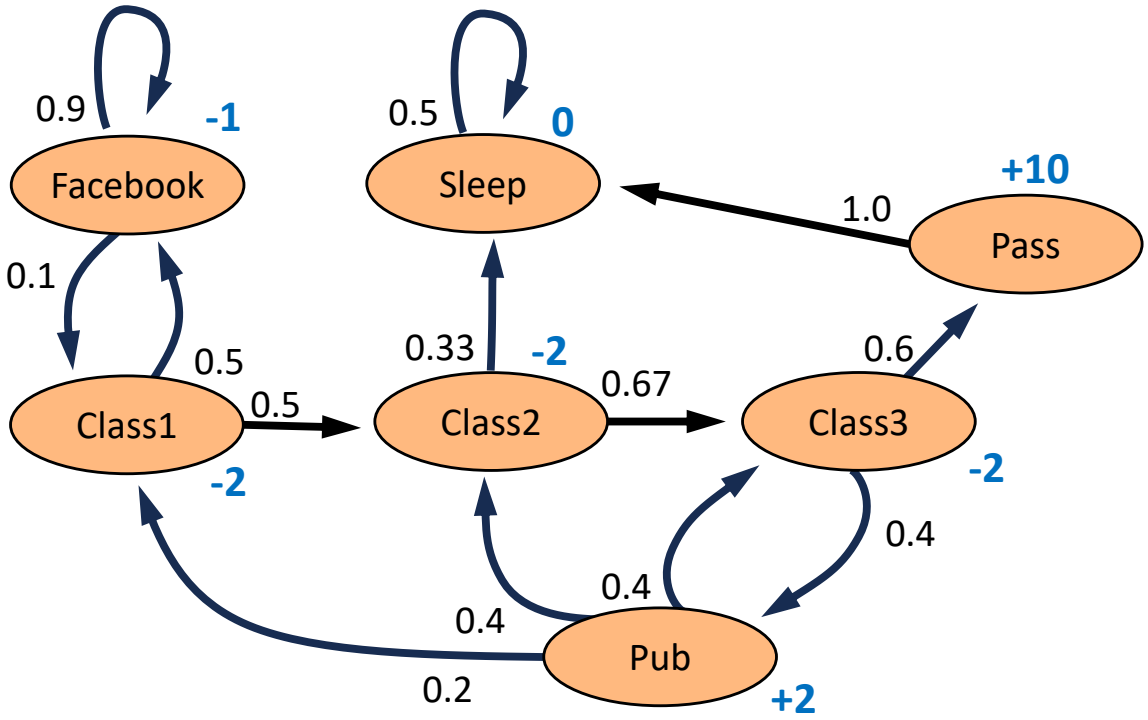
Ex: University MRP: Value Function



	Fa	C1	C2	C3	Pub	Pa	Slp
$\gamma = 0$	-1	-2	-2	-2	1	10	0

Don't care about future rewards?
Trivial value function: YOLO baby!

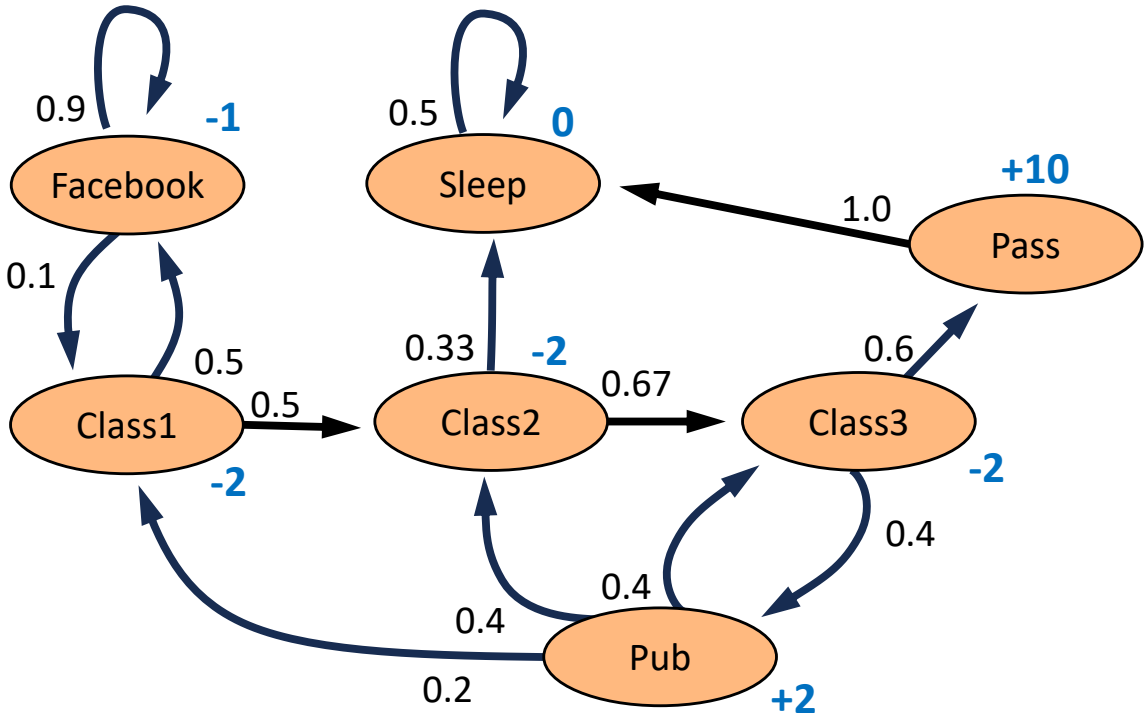
Ex: University MRP: Value Function



	Fa	C1	C2	C3	Pub	Pa	Slp
γ = .5	-2.08	-2.91	-1.55	-1.33	1.67	10	0

Future is more important?
Value of states that can take us to Pass are higher

Ex: University MRP: Value Function



	Fa	C1	C2	C3	Pub	Pa	Slp
γ=.99	-19	-10.8	1.2	4.83	2.25	10	0

Future is more important?
Value of states that can take us to Pass are higher

Bellman Equations for MRP

- Value function can be written recursively:

$$
\begin{aligned}
v(s) &= \mathbb{E}[G_t \,|\, S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \,|\, S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \ldots\right) | \, S_t = s] \\
&= \mathbb{E}[R_{t+1} \quad + \quad \gamma \; G_{t+1} \,|\, S_t = s]
\end{aligned}
$$

(immediate reward) + (discounted future return)

$$
\begin{aligned}
&= \mathbb{E}[R_{t+1} \quad + \quad \gamma \; v(S_{t+1}) \,|\, S_t = s] \\
&= \mathbb{E}[R_{t+1} \,|\, S_t = s] + \gamma \, \mathbb{E}[v(S_{t+1}) \,|\, S_t = s]
\end{aligned}
$$

discounted value of (random) next state

- Giving the recursive relation:

$$v(s) = \mathbf{r}_s + \gamma \sum_{s'} \mathbf{P}_{ss'} \; v(s')$$

Backup Diagrams for MRP

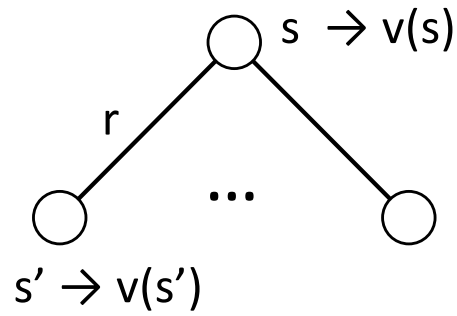
$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma\, v(S_{t+1}) \mid S_t = s\right]$$

$$= \mathbf{r}_s + \gamma \sum_{s'} \mathbf{P}_{ss'}\, v(s')$$

(expected value of immediate reward)

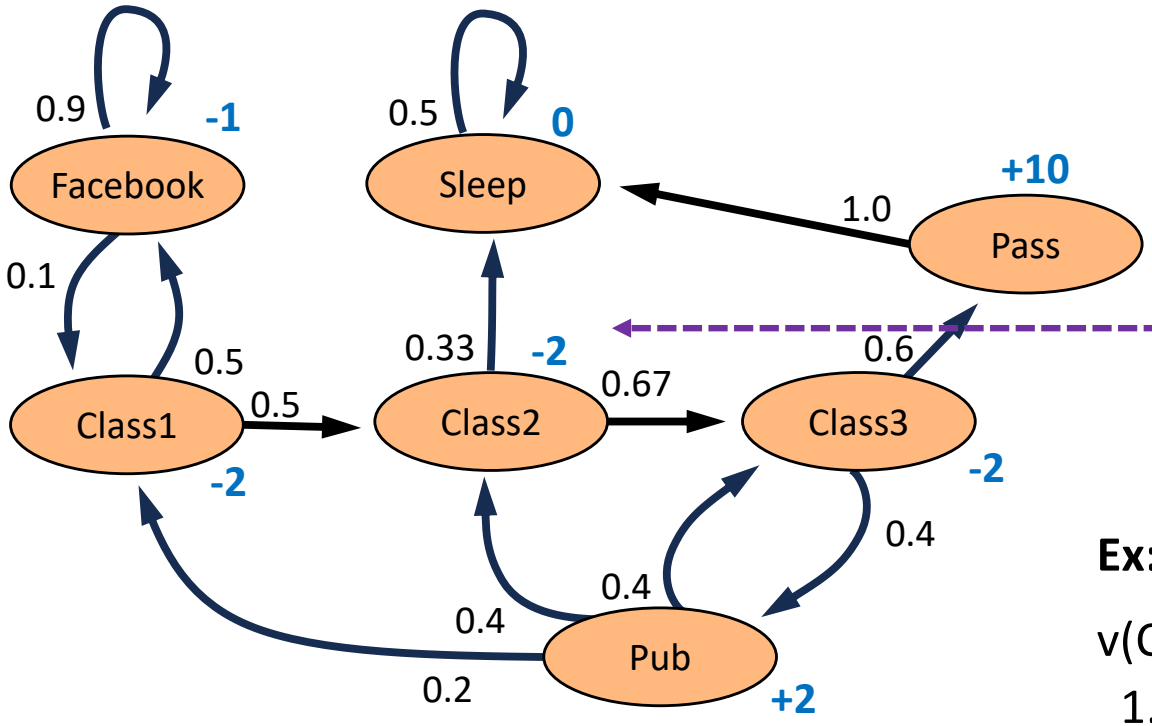
(probability of moving to s')

(value of being in s')



(dynamics: average over transitions & rewards)

Ex: UniLife MRP: Value Function



Ex: Bellman Recursion (γ=.99)

v(C2) = -2 + .99 ((.67) v(C3) + (.33) v(Sl))

1.2 = -2 + .99 ((.67) 4.83 + (.33) 0)

	Fa	C1	C2	C3	Pub	Pa	Slp
γ=.99	-19	-10.8	1.2	4.83	2.25	10	0

Matrix Form of Bellman Eq

- Write Bellman equation in matrix form:

$$\mathbf{v} = \mathbf{r} + \gamma \, \mathbf{P} \odot \mathbf{v}$$

where v is a vector with one entry per state:

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathbf{P}_{11} & \cdots & \mathbf{P}_{1n} \\ \vdots & \ddots & \vdots \\ \mathbf{P}_{n1} & \cdots & \mathbf{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

- We can then solve for v directly:

$$\mathbf{v} = \mathbf{r} + \gamma \, \mathbf{P} \odot \mathbf{v}$$

$$(I - \gamma \, \mathbf{P}) \odot \mathbf{v} = \mathbf{r}$$

$$\mathbf{v} = (I - \gamma \, \mathbf{P})^{-1} \odot \mathbf{r}$$

Direct solution with n states:

- Matrix inverse is $O(n^3)$
- Only feasible for small MRPs

For larger MRPs, use iterative methods!

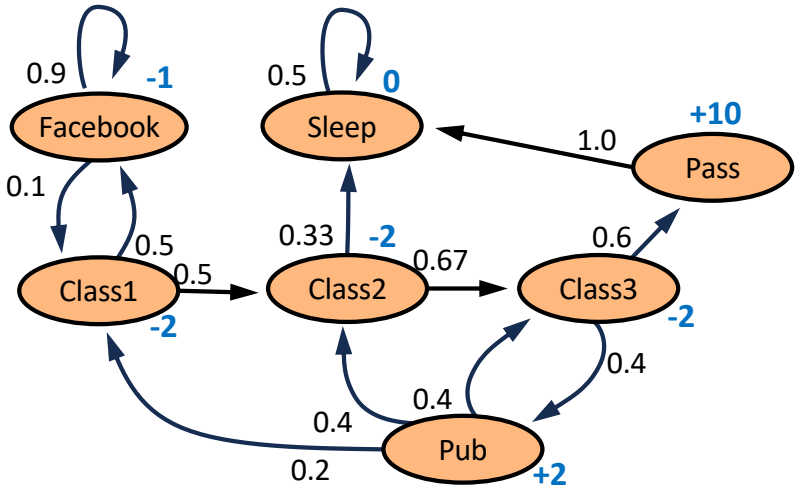
Dynamic Programming

Define $v^L(s)$ to be the value of length-L state sequences:

- $v^1(s) = r_s$ [no future!]
- $v^2(FB) = -1 + \gamma\,(\,.9\,v^1(FB) + .1\,v^1(C1)\,)$
- $v^2(C1) = -2 + \gamma\,(\,.5\,v^1(FB) + .5\,v^1(C2)\,)$
- …

Then,

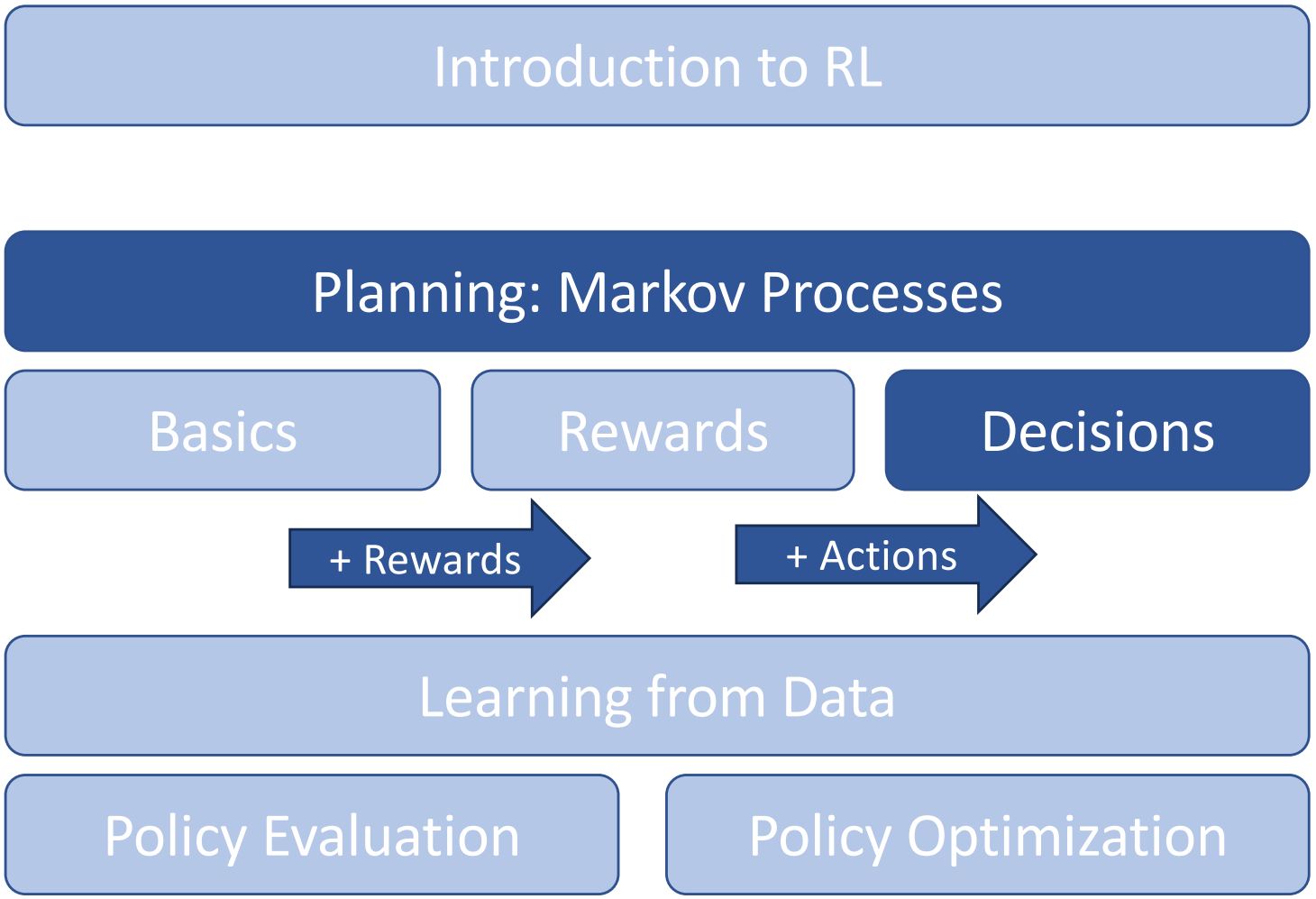
- $v^3(FB) = -1 + \gamma\,(\,.9\,v^2(FB) + .1\,v^2(C1)\,)$
- …



$\gamma = 0.5$

	FB	C1	C2	C3	Pub	Pa	Slp
L=1	-1	-2	-2	-2	2	10	0
t=2	-1.55	-2.75	-2.67	1.4	1	10	0
t=3	-1.84	-3.06	-1.53	1.2	1.47	10	0
t=4	-2.08	-2.91	-1.55	1.33	1.67	10	0

Reinforcement Learning



Markov Decision Process

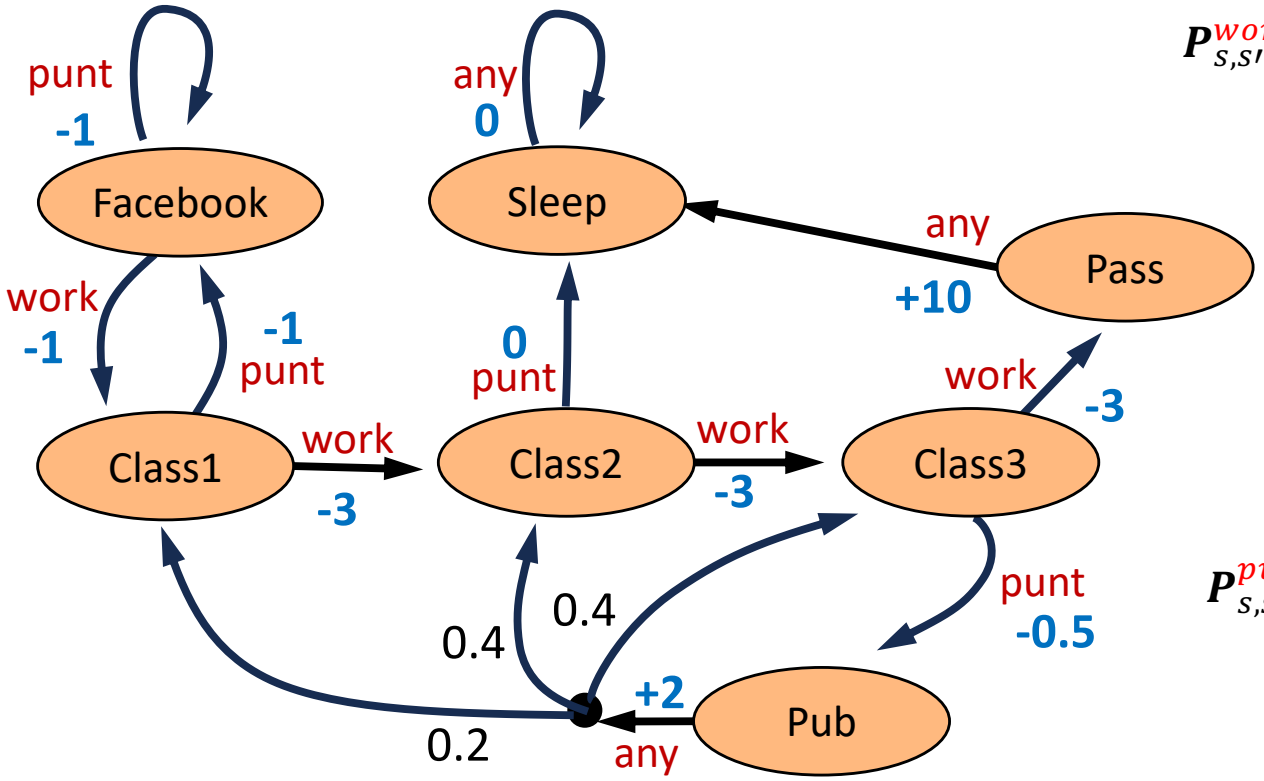
- Markov reward process, plus decisions (actions)
  - Specifies an environment of Markov states

Definition

A Markov Decision Process is a tuple [**S**, **A**, **P, R,** γ]:

- **S** is a (finite) set of states
- **A** is a (finite) set of actions
- **P** is a state transition probability matrix,
  $$\mathbf{P}^{a}_{ss'} = p(S_{t+1} = s' | S_t = s, A_t = a) :$$
- **R** is a reward function, $\mathbf{r}^{a}_{s} = \mathbb{E}\left[R_{t+1} \,|\, S_t = s, A_t = a \right]$
- γ is a discount factor, $\gamma \in [0, 1]$

Ex: University Life MDP



$$\mathbf{P}_{s,s'}^{work} = p(S_{t+1} = s' | S_t = s, A_t = work)$$

s \ s'	Fa	C1	C2	C3	Pu	Pa	Sl
Fa		1.0					
C1	0.5		1.0				
C2				1.0			
C3						1.0	
Pu		0.2	0.4	0.4			
Pa							1.0
Sl							1.0

$$\mathbf{P}_{s,s'}^{punt} = p(S_{t+1} = s' | S_t = s, A_t = punt)$$

s \ s'	Fa	C1	C2	C3	Pu	Pa	Sl
Fa	1.0						
C1	1.0						
C2							1.0
C3					1.0		
Pu		0.2	0.4	0.4			
Pa							1.0
Sl							1.0

$$\mathbf{r}_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a] :$$

	Fa	C1	C2	C3	Pu	Pa	Sl
work	-1	-3	-3	-3	+2	+10	0
punt	-1	-1	0	-0.5	+2	+10	0

Policies

- A policy π is a distribution over actions given states:

$$\pi(a|s) = p(A_t = a \,|\, S_t = s)$$

  - Fully defines the behavior of our agent

  - In a Markov Decision Process,
    - Policies depend only on the current state s,
      not on the (full) history H_t

    - Policies are stationary, $A_t \sim p(A \mid S_t)$ for all times t

MPs → MRPs → MDPs

- Selecting a policy reduces an MDP to an MRP / MP:

- Given an MDP [S,A,P,R,γ] and a policy π,
  - The state sequence S1, S2,… is a Markov process [S, P^π]
  - The state and reward sequences S1, R1, S2, R2,… is a Markov reward process [S, P^π, R^π, γ]

  where:

$$\mathbf{P}^{\pi}_{s,s'} = \sum_a \pi(a|s)\,\mathbf{P}^{a}_{s,s'} \qquad \mathbf{r}^{\pi}_{s} = \sum_a \pi(a|s)\,\mathbf{r}^{a}_{s}$$

(i.e., weighted averages over the actions)

Value Function

- State-value function $v_\pi(s)$
  - Expected return, starting from s, following policy π

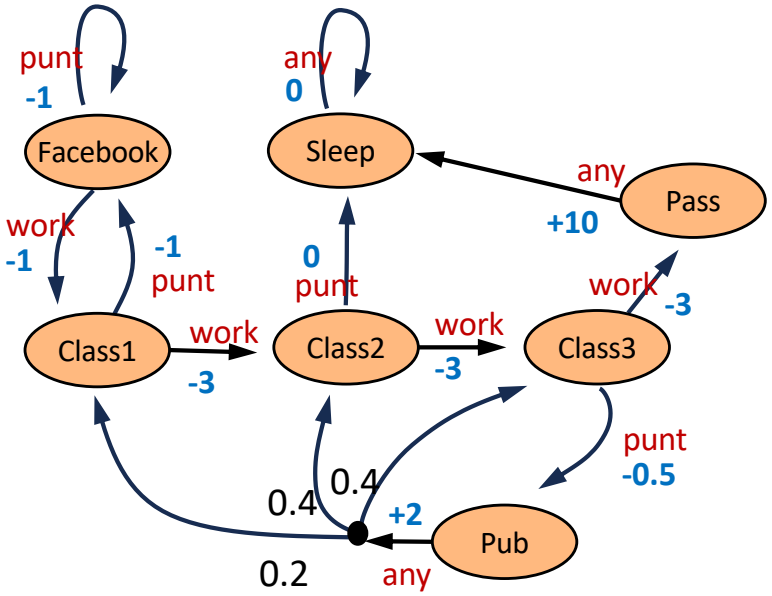
$$v_\pi(s) = \mathbb{E}_\pi\left[G_t \mid S_t = s\right]$$

- State-action-value function $q_\pi(s,a)$
  - Expected return, starting from s, taking action a, then following policy π.

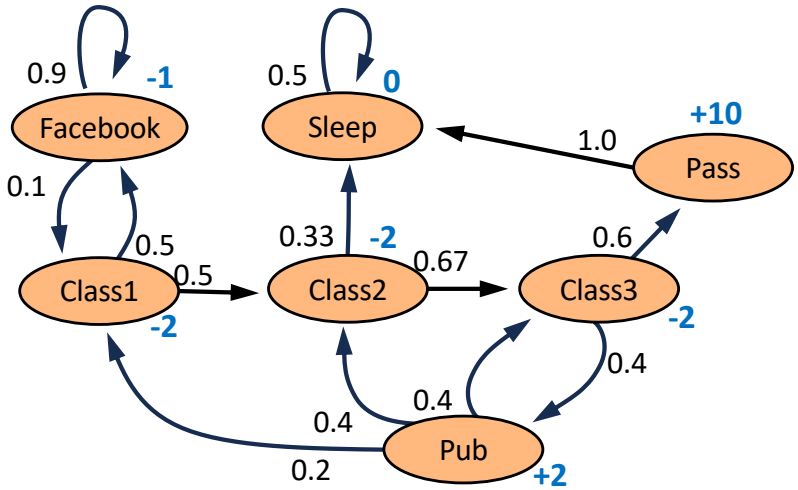
$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right]$$

  - Will be useful later…

Ex: UniLife MDP: Value Function



$$p_\pi(\text{work}|\text{Fa}) = 0.1$$
$$p_\pi(\text{work}|\text{C1}) = 0.5$$
$$p_\pi(\text{work}|\text{C2}) = 0.67$$
$$p_\pi(\text{work}|\text{C3}) = 0.6$$



	Fa	C1	C2	C3	Pub	Pa	Slp
γ = .5	-2.08	-2.91	-1.55	-1.33	1.67	10	0

Bellman Expected Equation

- Can express state-value function recursively:
  - Immediate reward, plus discounted value of next state

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma \, v_\pi(S_{t+1}) \,|\, S_t = s \right]$$

- Similarly for the state-action-value function:

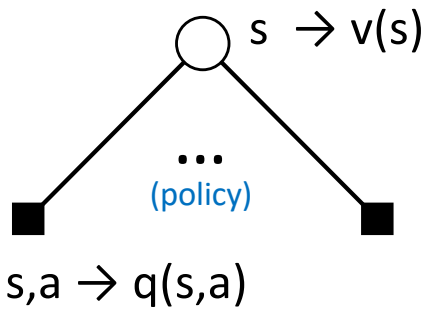
$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma_\pi(S_{t+1}, A_{t+1}) \,|\, S_t = s, A_t = a \right]$$

Bellman Expected Equation: V from Q

$$v_\pi(s) = \sum_a \pi(a|s)\ q_\pi(s,a)$$

(prob. take action a)

(follow pi after s,a)



Bellman Expected Equation: Q from V

$$v_\pi(s) = \sum_a \pi(a|s)\ q_\pi(s,a)$$

(prob. take action a)

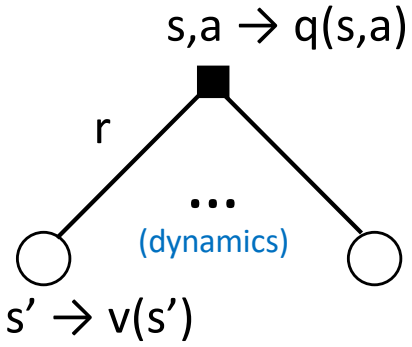
(follow pi after s,a)

$$q_\pi(s,a) = \mathbf{r}_s^a + \gamma \sum_{s'} \mathbf{P}_{s,s'}^a\ v_\pi(s')$$

(reward s,a)

(prob. s,a goes to s')

(value of s')



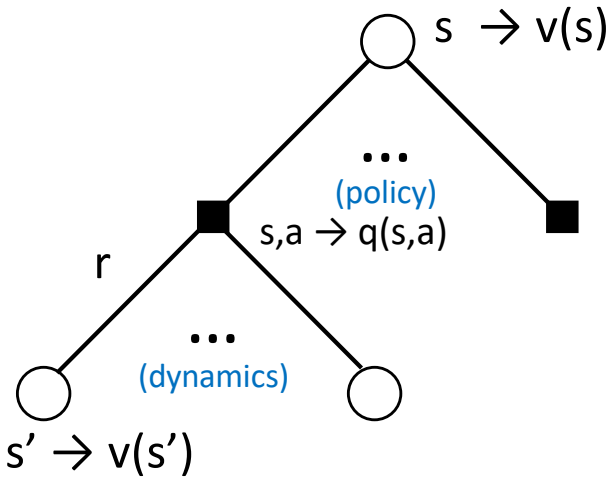
Bellman Expected Equation: V or Q

$$v_\pi(s) = \sum_a \pi(a|s)\ q_\pi(s,a)$$

(prob. take action a) (follow pi after s,a)

$$q_\pi(s,a) = \mathbf{r}_s^a + \gamma \sum_{s'} \mathbf{P}_{s,s'}^a\ v_\pi(s')$$

(reward s,a) (prob. s,a goes to s') (value of s')



$$v_\pi(s) = \sum_a \pi(a|s) \left(\mathbf{r}_s^a + \gamma \sum_{s'} \mathbf{P}_{s,s'}^a\ v_\pi(s') \right)$$

$$q_\pi(s,a) = \mathbf{r}_s^a + \gamma \sum_{s'} \mathbf{P}_{s,s'}^a \sum_{a'} \pi(a'|s')\ q_\pi(s',a')$$

Optimal Value Function

- How can we select a policy? How well can we possibly do?

Definition

The optimal state-value function v*(s) is the maximum of the value function over all possible policies:

$$v^*(s) = \max_\pi v_\pi(s)$$

The optimal state-action-value function q*(s,a) is the maximum of the s-a-value function over all possible policies:

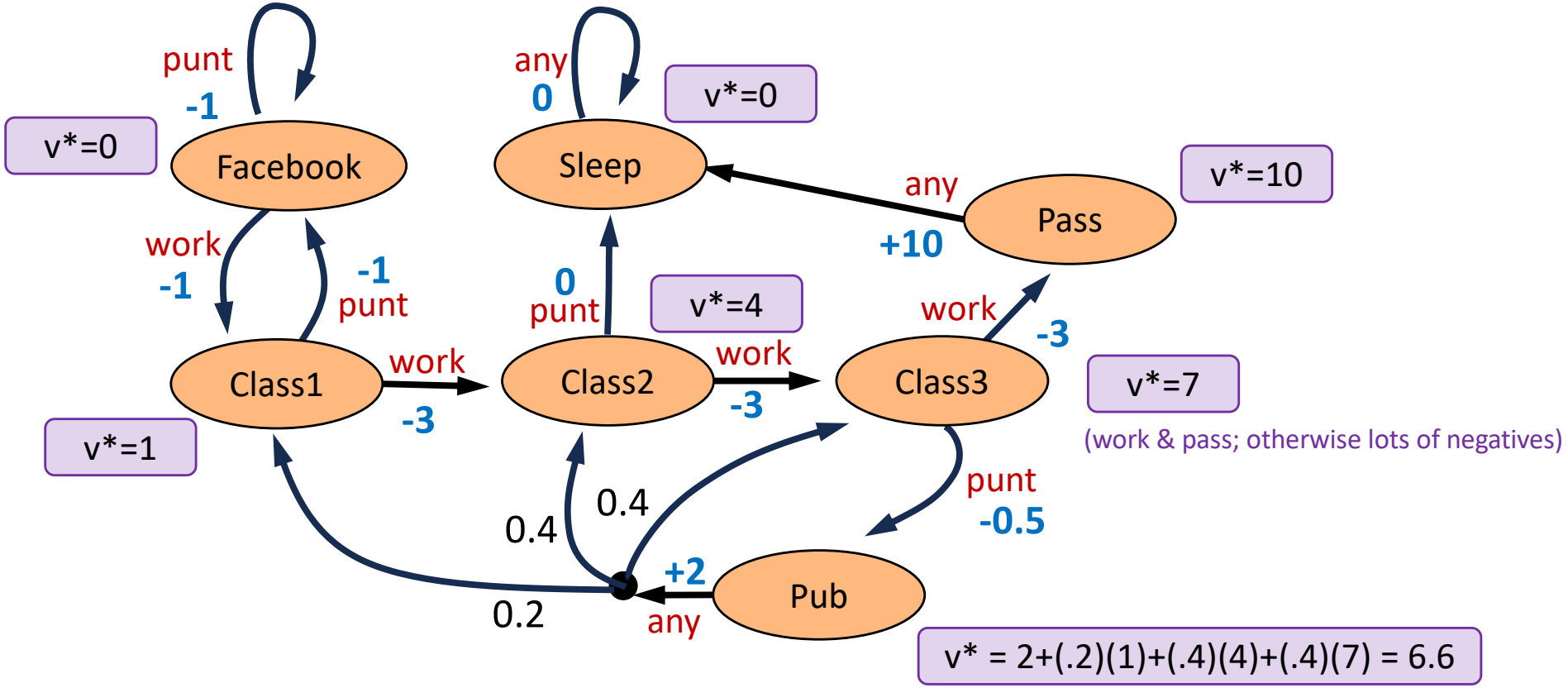
$$q^*(s,a) = \max_\pi q_\pi(s,a)$$

- The optimal value function tells us the best possible performance in the MDP
- An MDP is "solved" once we know the optimal value function

Ex: University Life MDP

- Optimal values?

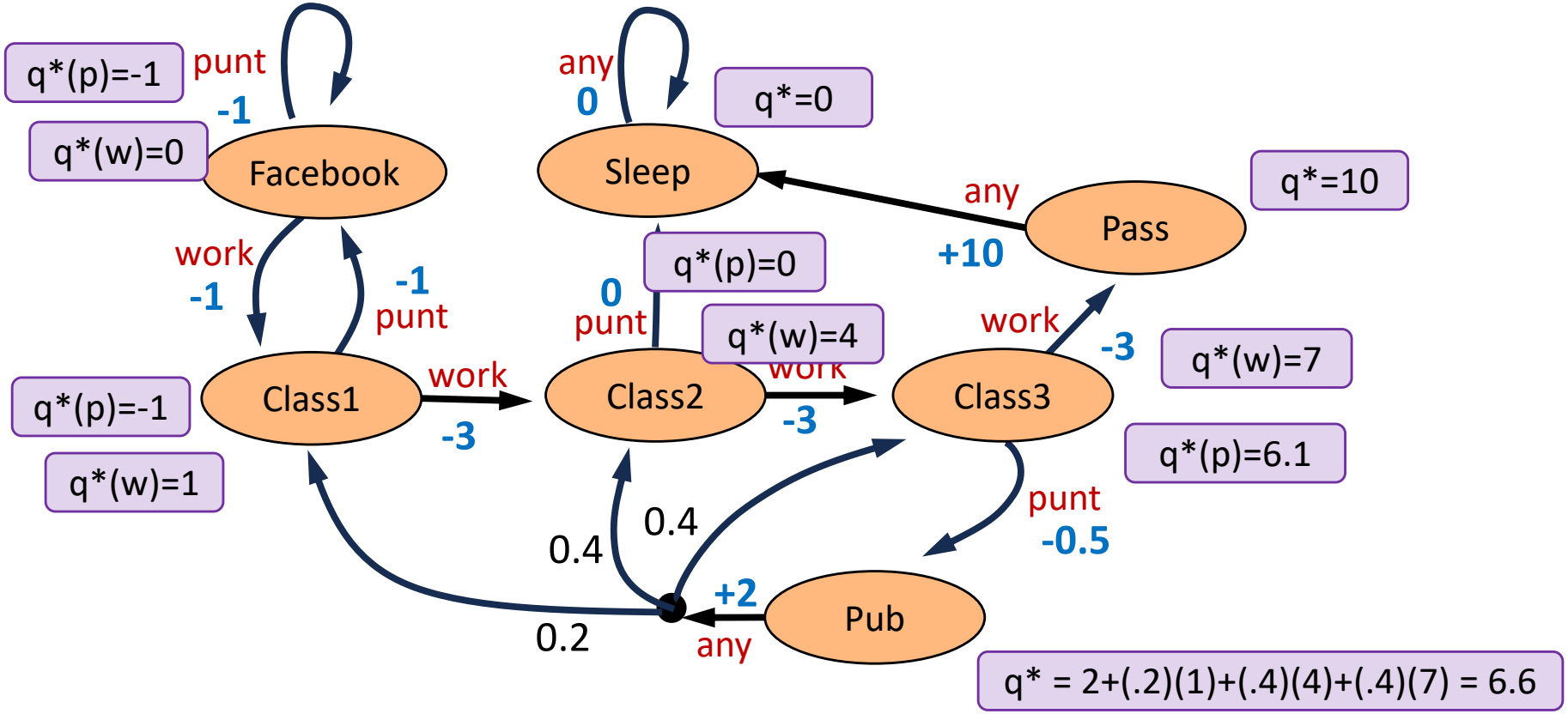
- Take $\gamma = 1$ (no discounting); easy to "read" solution:
  - Optimal state-values:



Ex: University Life MDP

- Optimal values?

- Take $\gamma = 1$ (no discounting); easy to "read" solution:
  - Optimal state-action values:



Optimal Policy

- Is there an optimal policy?

- Define a partial ordering over policies

$$\pi \geq \pi' \quad \text{if} \quad v_\pi(s) \geq v_{\pi'}(s) \;\; \forall s$$

(π dominates π': as good or better from every state s)

Theorem: For any Markov Decision Process,

- There exists an optimal policy p that is better or equal to all other policies, $\pi \geq \pi' \;\; \forall \pi'$
- All optimal policies achieve the optimal state-value function:

  $v_\pi(s) = v^*(s)$
- All optimal policies achieve the optimal state-action-value f'n:

  $q_\pi(s,a) = q^*(s,a)$

Finding Optimal Policy

- If we know v*(s) or q*(s,a), the MDP is solved: why?

- To follow the optimal policy given q*(s,a), just find the max:

$$\pi^*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a'} q(s, a') \\ 0 & \text{otherwise} \end{cases}$$

(i.e. choose the best action a' with probability 1)

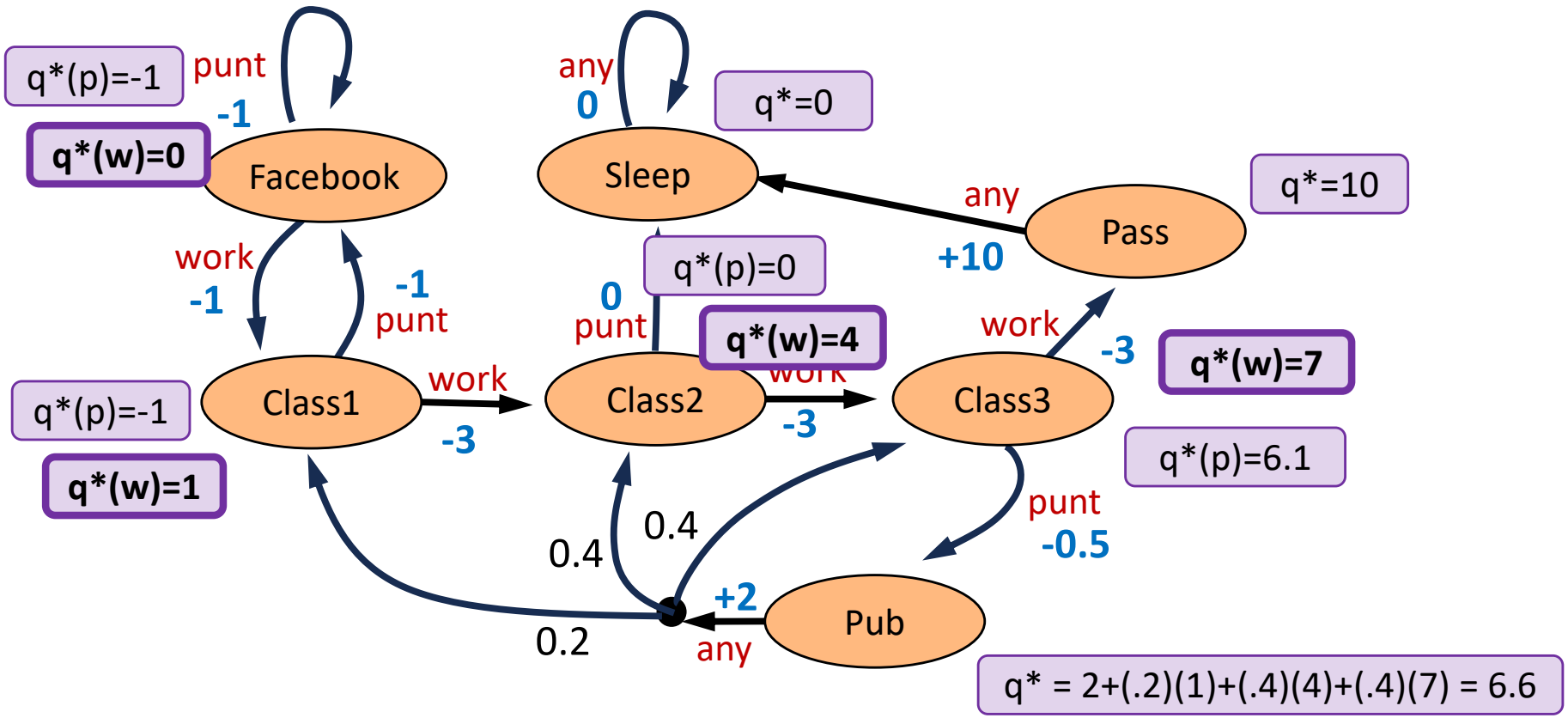
  - There is always a deterministic optimal policy!

- To follow the optimal policy given v*(s), convert to q*(s,a):

$$a^*(s) = \arg\max_{a'} \; \mathbf{r}_s^{a'} \; + \; \gamma \sum_{s'} \mathbf{P}_{s,s'}^{a'} \; v^*(s')$$

Ex: University Life MDP

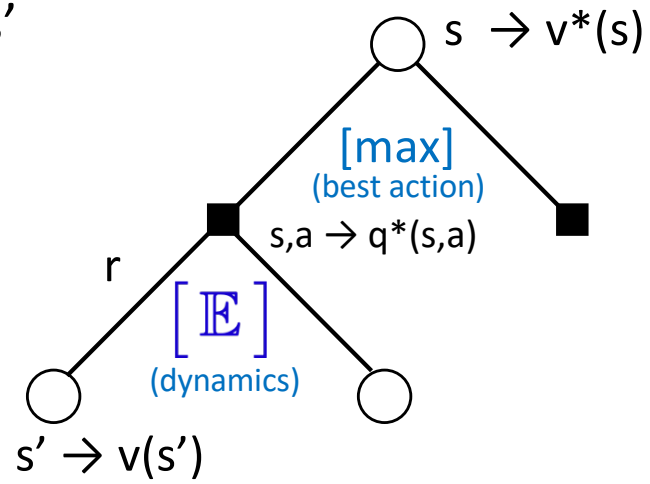
- Reading out the optimal policy
  - For $\gamma = 1$ (no discounting)
  - Choose the action that maximizes q*(s,a) at each state s:



Bellman Optimality

- Apply recursive definition to v*, q*:
  - Choose action a with best value (max)
  - Take expectation (average) over outcomes s'

$$v^*(s) = \max_a \left(\mathbf{r}_s^a + \gamma \sum_{s'} \mathbf{P}_{s,s'}^a \, v^*(s') \right)$$



or, equivalently:

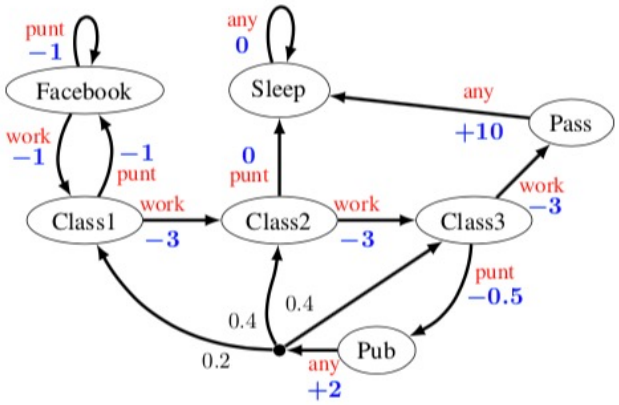
$$q^*(s,a) = \mathbf{r}_s^a + \gamma \sum_{s'} \mathbf{P}_{s,s'}^a \max_{a'} q^*(s',a')$$

- Analogous to expecti-mini-max in game trees (but no adversary / min)

From MDPs to RL

	Evaluate Policy, π	Find Best Policy, π^*
MDP Known	Policy Evaluation	Value/Policy Iteration
MDP Unknown	MC and TD Learning	Sarsa + Q-Learning

Value Iteration



- Compute value of best policy over L steps:
  - v_1 is easy (one action, reward, then stop)

  - v_2: take one action, then get value v_1, e.g.:
  - $q_2^*(C1, \text{work}) = -3 + \gamma\, v_1^*(C2) \quad = -3$
  - $q_2^*(C1, \text{punt}) = -1 + \gamma\, v_1^*(FB) \quad = -1.5$ so, $v_2^*(C1) = -1.5$

  - …

$\gamma = 0.5$

L	Fa	C1	C2	C3	Pu	Pa	Sl
1	-1	-1	0	-0.5	2	10	0
2	-1.5	-1.5	0	2	1.8	10	0
3	-1.75	-1.75	0	2	2.25	10	0
⋮							
10	-2.0	-2.0	0	2	2.2	10	0

As L increases, approaches the optimal, infinite horizon value.

Improving a Policy!

- Suppose we already have a policy
- How can we find a better one?

- Given a policy π:
  - Evaluate the policy:

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s]$$

  - Improve the policy:

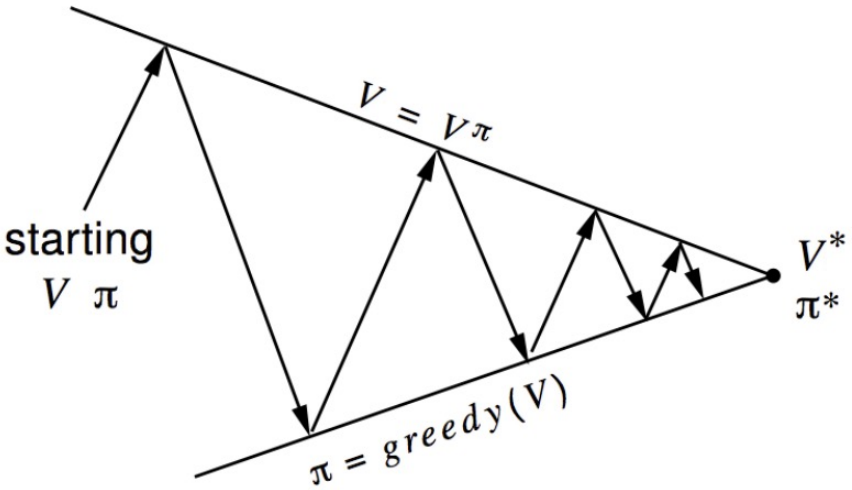
$$\pi' = \text{greedy}(v_\pi)$$

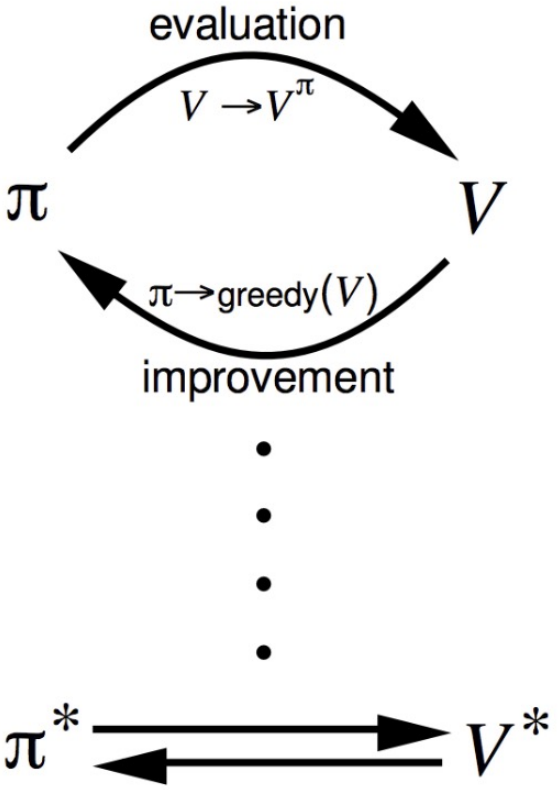
where

$$\text{greedy}_\pi(s) = \arg\max_{a'} \mathbf{r}_s^{a'} + \gamma \sum_{s'} \mathbf{P}_{s,s'}^{a'} v_\pi(s')$$

  - New policy π' at least as good as π (could pick same action)
  - Iterating eventually converges to the optimal policy, π^*!

Policy Iteration

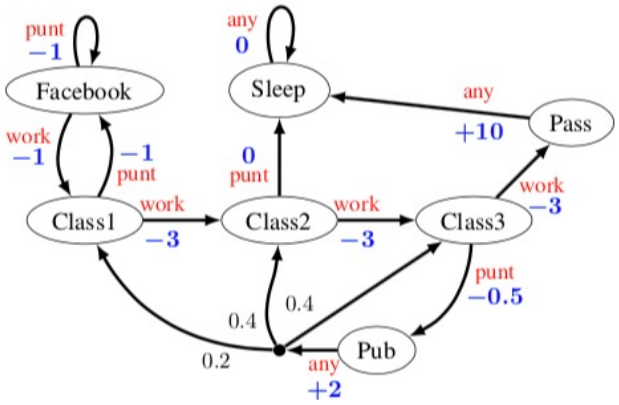




Policy evaluation Estimate v_π
Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement

Policy Iteration



- Iteratively improve on a fixed policy:
  - Initialize our policy, e.g.,

$$p_\pi(\text{work}|\text{Fa}) = 0.1 \qquad p_\pi(\text{work}|\text{C1}) = 0.5 \qquad p_\pi(\text{work}|\text{C2}) = 0.67 \qquad p_\pi(\text{work}|\text{C3}) = 0.6.$$

  - Find the current policy's value function:

$s =$	Fa	C1	C2	C3	Pu	Pa	Sl
$v(s) =$	-2.08	-2.91	-1.55	1.33	1.67	10	0

  - For each state, find the greedy best action:

$$\pi'(\text{Fa}) = \arg\max_A \begin{cases} -1 + \gamma(-2.08) & (A = \text{punt}) \\ -1 + \gamma(-2.91) & (A = \text{work}) \end{cases} = \begin{cases} -2.04 \\ -2.45 \end{cases}$$

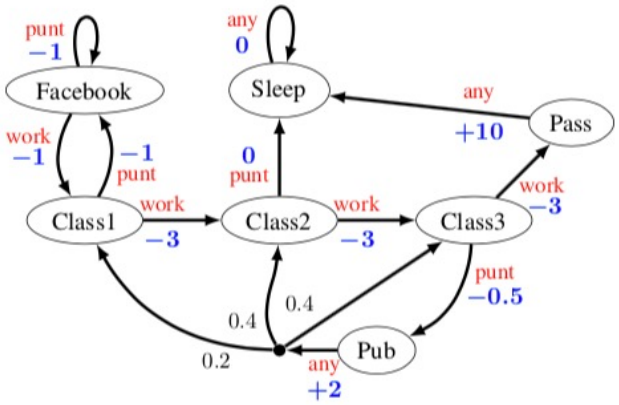
$\mathbf{r}_s^a + \gamma\,\mathbb{E}[v_\pi(S_{t+1})|S_t = s]$:

	Fa	C1	C2	C3	Pu	Pa	Sl
work	−2.45	−3.78	−2.33	2	1.66	10	0
punt	−2.04	−2.04	0	0.33			

$\Rightarrow$ $\pi'(s)$:

Fa	C1	C2	C3
punt	punt	punt	work

Policy Iteration (2)



- Iteratively improve on a fixed policy:
  - Using our new policy,

$\pi'(s)$:

	Fa	C1	C2	C3
	punt	punt	punt	work

  - Find the policy's value function:

$s =$	Fa	C1	C2	C3	Pu	Pa	Sl
$v_{\pi'}(s) =$	-2	-2	0	0.6	2.2	10	0

  - For each state, find the greedy best action:

$\mathbf{r}_s^a + \gamma \mathbb{E}[v_{\pi'}(S_{t+1})|S_t = s]$:

	Fa	C1	C2	C3	Pu	Pa	Sl
work	−2	−3	−2	2	2.2	10	0
punt	−2	−2	0	0.6			

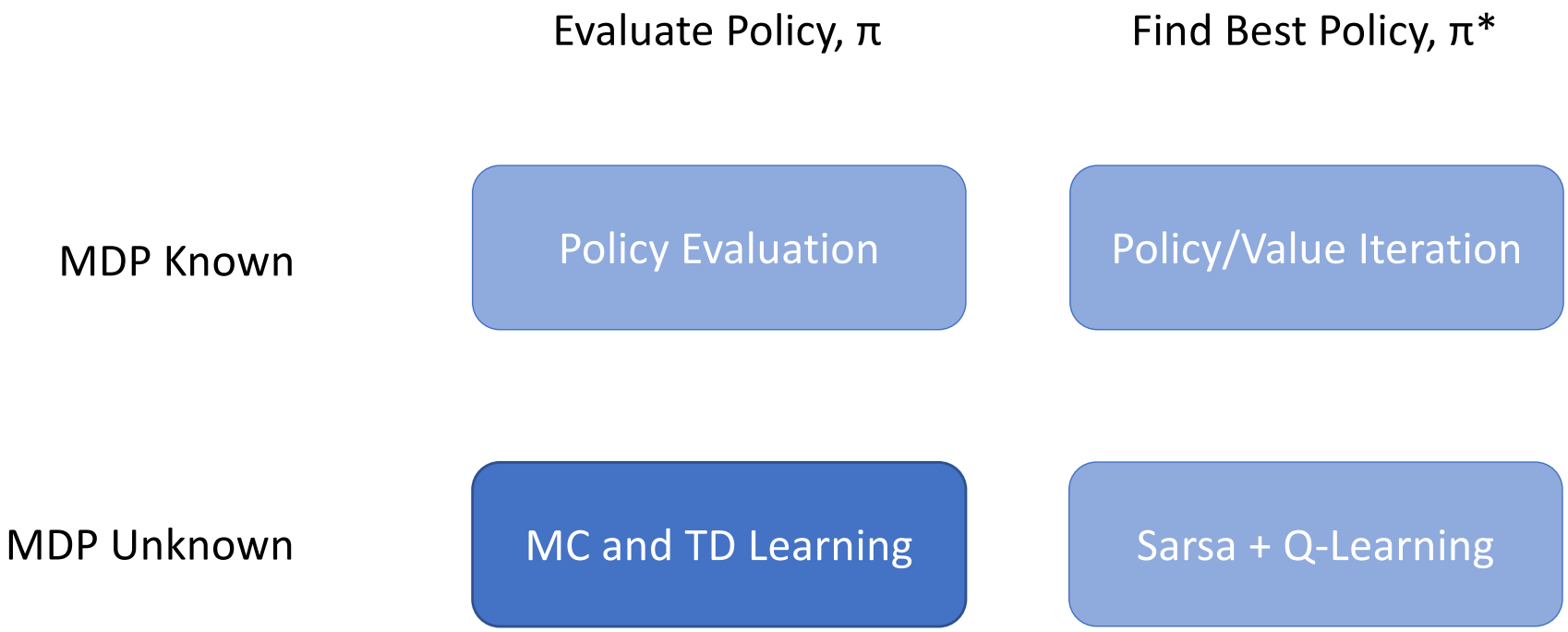
$\Rightarrow$

$\pi''(s)$:

	Fa	C1	C2	C3
	punt	punt	punt	work

  - Our new greedy policy is the same as our current policy! → **optimal**
  - In general, we may have to iterate more than once…

From MDPs to RL



Monte Carlo RL

- A general technique from statistics

- MC estimators use a simple idea:

  Estimate expected return with empirical mean return

- MC methods learn directly from experience episodes
- MC estimators are model-free
  - No knowledge of MDP transitions or rewards

- MC learns from complete episodes (no bootstrapping)
  - So, can only apply MC to episodic MDPs!
  - All episodes must terminate

Monte Carlo Policy Evaluation

- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_2, ..., S_k \sim \pi$$

- Recall that the *return* is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

- Recall that the value function is the expected return:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

- Monte-Carlo policy evaluation uses *empirical mean* return instead of *expected* return

Every-Visit MC Policy Evaluation

- To evaluate state s
- Every time-step t that state s is visited in an episode,
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- Again, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

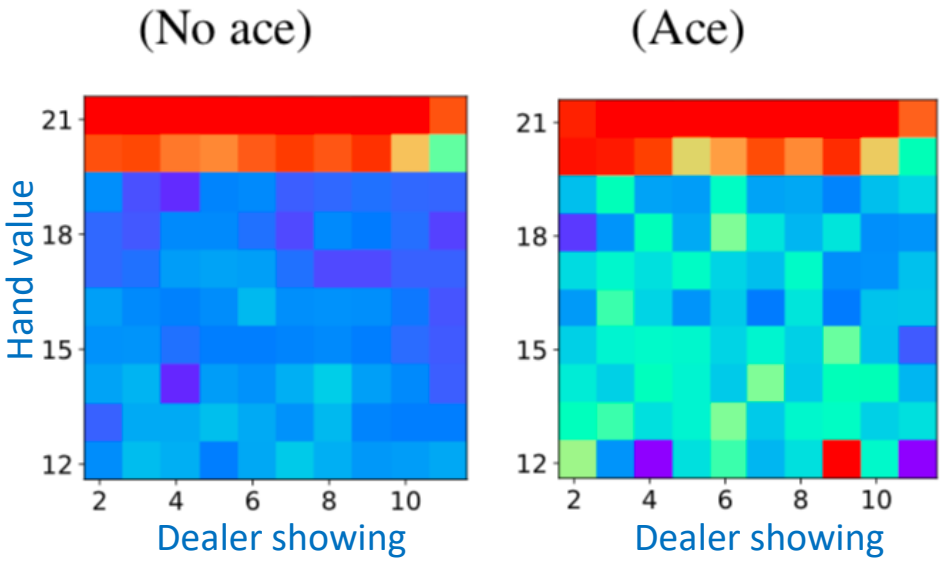
Ex: Blackjack

- States (200 of them):
  - Current sum (12-21)
  - Dealer's showing card (ace-10)
  - Do I have a "useable" ace? (yes-no)
- Action stand Stop receiving cards (and terminate)
- Action hit : Take another card (no replacement)
- Reward for stand
  - +1 if sum of cards > sum of dealer cards
  - 0 if sum of cards = sum of dealer cards
  - -1 if sum of cards < sum of dealer cards
- Reward for hit :
  - -1 if sum of cards > 21 (and terminate)
  - 0 otherwise
- Transitions: automatically hit if sum of cards < 12

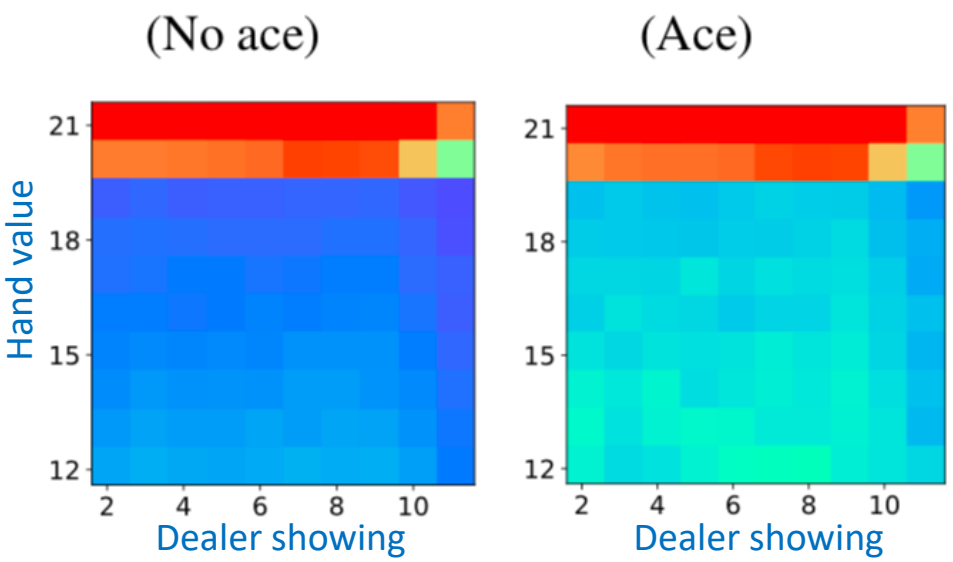
Ex: Blackjack Value Function, MC

(Policy π being evaluated: stand if sum of cards ≥ 20, otherwise hit)

m = 10,000 episodes:



m = 500,000 episodes:



Temporal Difference Learning

- TD methods also learn directly from experience episodes
- Also model-free
  - No knowledge of MDP transitions or rewards

- TD learns from incomplete episodes, by "bootstrapping"
  - "Updates a guess towards a guess"

MC and TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward *actual* return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

MC and TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward *actual* return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

MC and TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward *actual* return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha\,(G_t - V(S_t))$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha\,(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

MC and TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward *actual* return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$

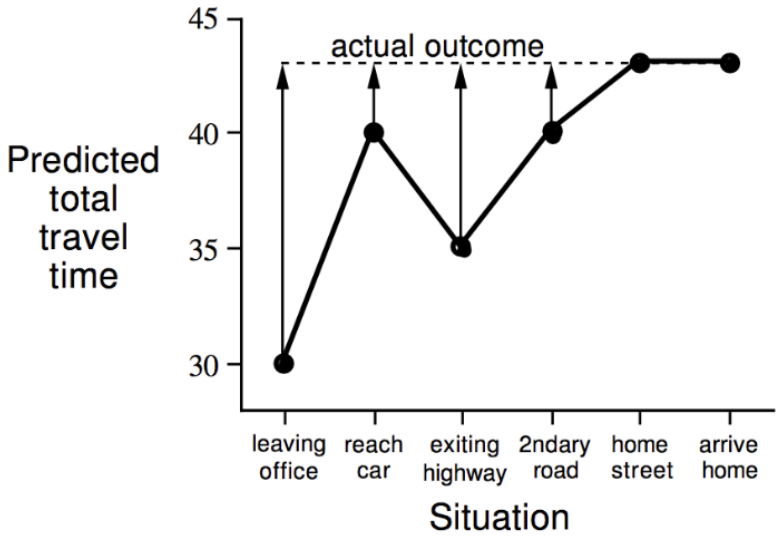
  - $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

Ex: Driving Home

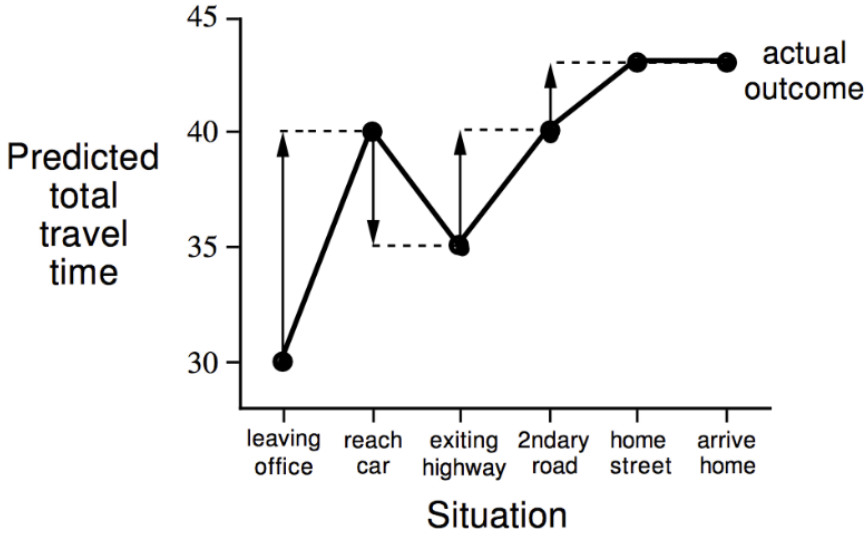
State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

Ex: Driving Home, MC vs TD

Changes recommended by Monte Carlo methods (α=1)



Changes recommended by TD methods (α=1)



Finite Episodes: AB Example

- Trivial MRP with two states, {A,B}
  - No discounting
  - Observe 8 epsiodes of experience:

A, 0, B, 0

B, 1

B, 1

B, 1

B, 1

B, 1

B, 1

B, 0

  - What are v(A) & v(B)?

MC & TD can give different answers on fixed data:

v(B) = 6 / 8

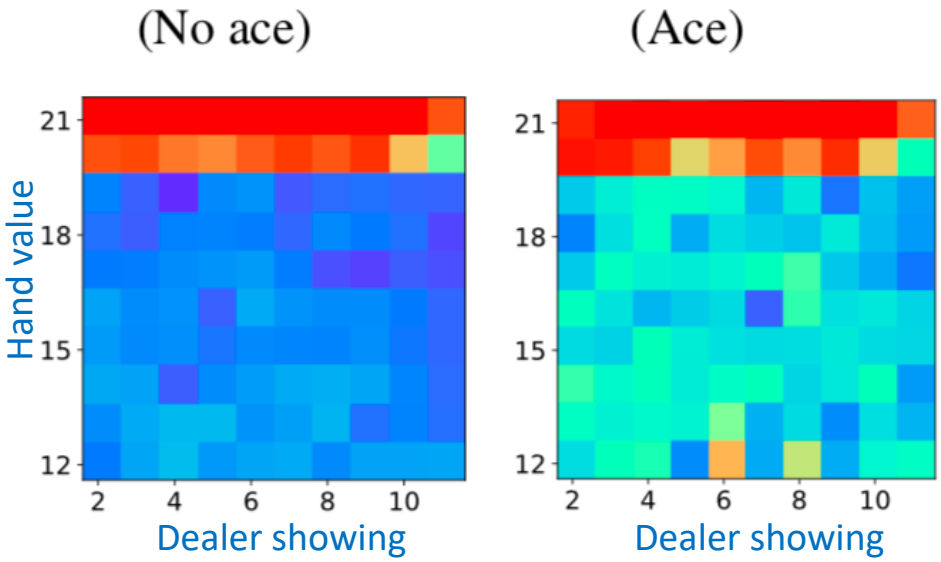
v(A) = 0 ? (Direct MC estimate)

v(A) = 6 / 8? (TD estimate)

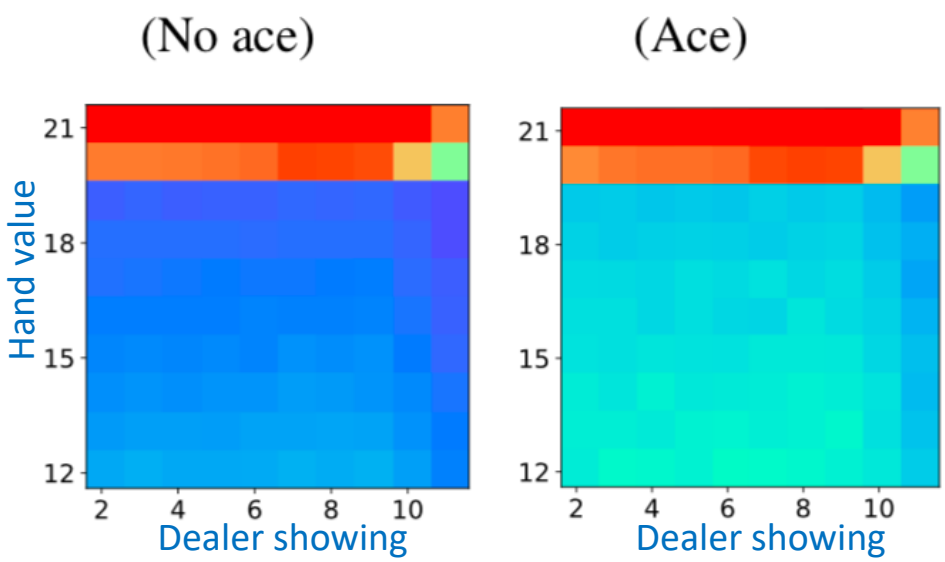
Blackjack Value Function: TD

(Policy being evaluated: stand if sum of cards ≥ 20, otherwise hit)

m = 10,000 episodes:



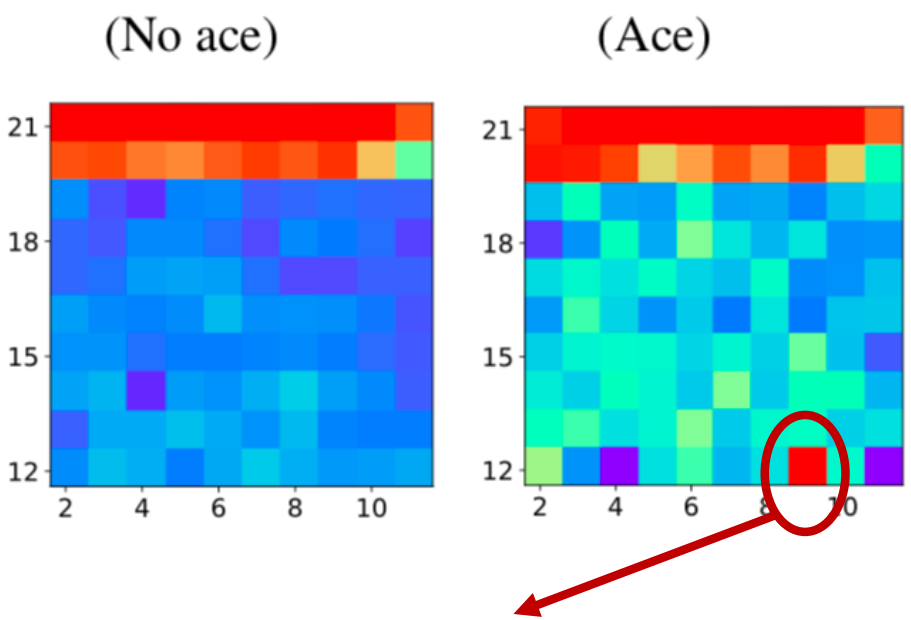
m = 500,000 episodes:



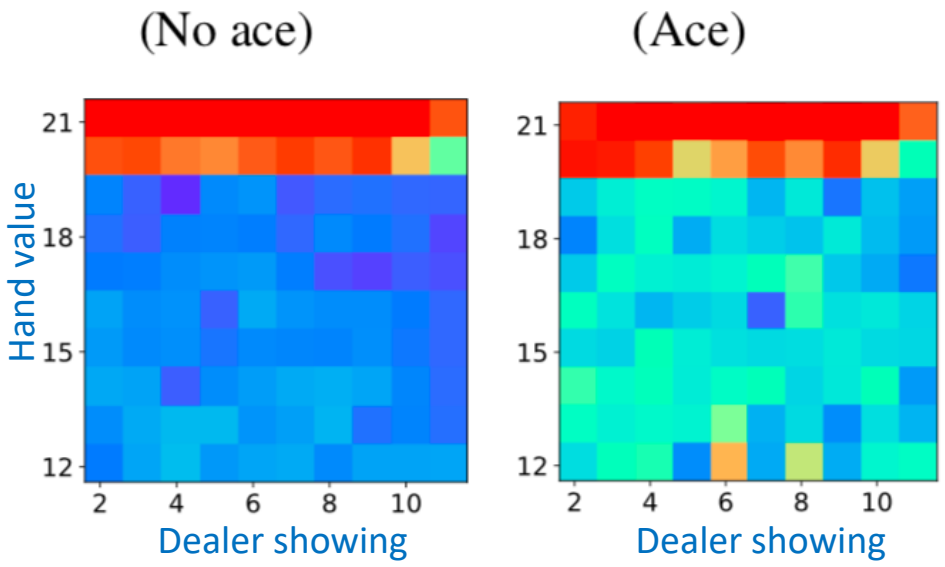
Blackjack Value Function: MC vs TD

(Policy being evaluated: stand if sum of cards ≥ 20, otherwise hit)

m = 10k, MC estimates:



m = 10k, TD(0) estimates:



Only one episode:

[12 (ace), 9] $\Rightarrow +10 \Rightarrow$ [12 (no ace), 9] $\Rightarrow +8 \Rightarrow$ [20 (no ace), 9] $\Rightarrow$ win

(but we've seen lots of other episodes from here, & often lose…)

MC vs TD

Monte Carlo

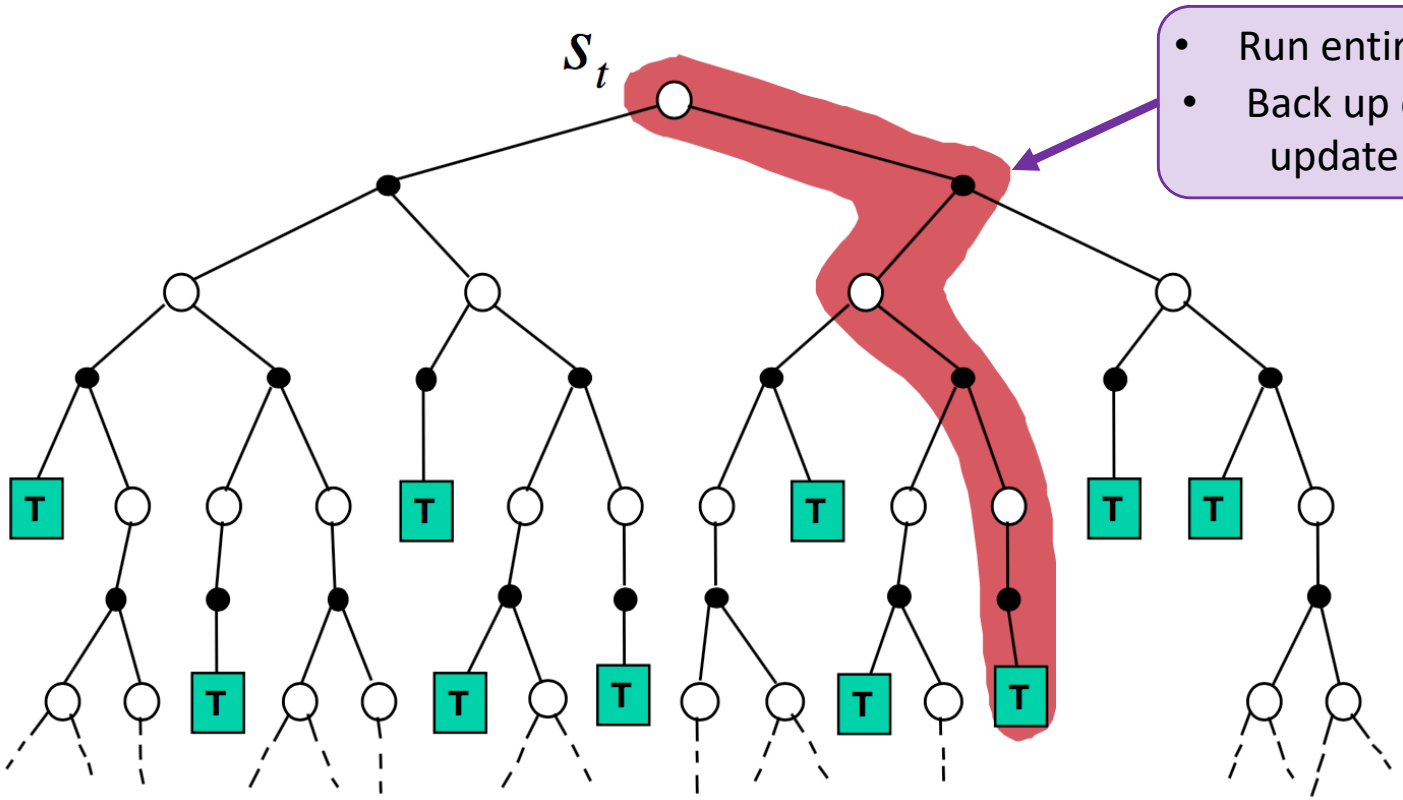
- Wait until the end of episode to learn
  - Only for terminating environments
- High variance; low bias
  - Not sensitive to initial value
  - Good convergence properties
- Does not use Markov property

Temporal difference

- Learn only after every step
  - Non-terminating environments OK
- Low variance, high bias
  - Sensitive to initialization of v(s)
  - Much more data-efficient
- Exploits Markov property

Unified View: Monte Carlo

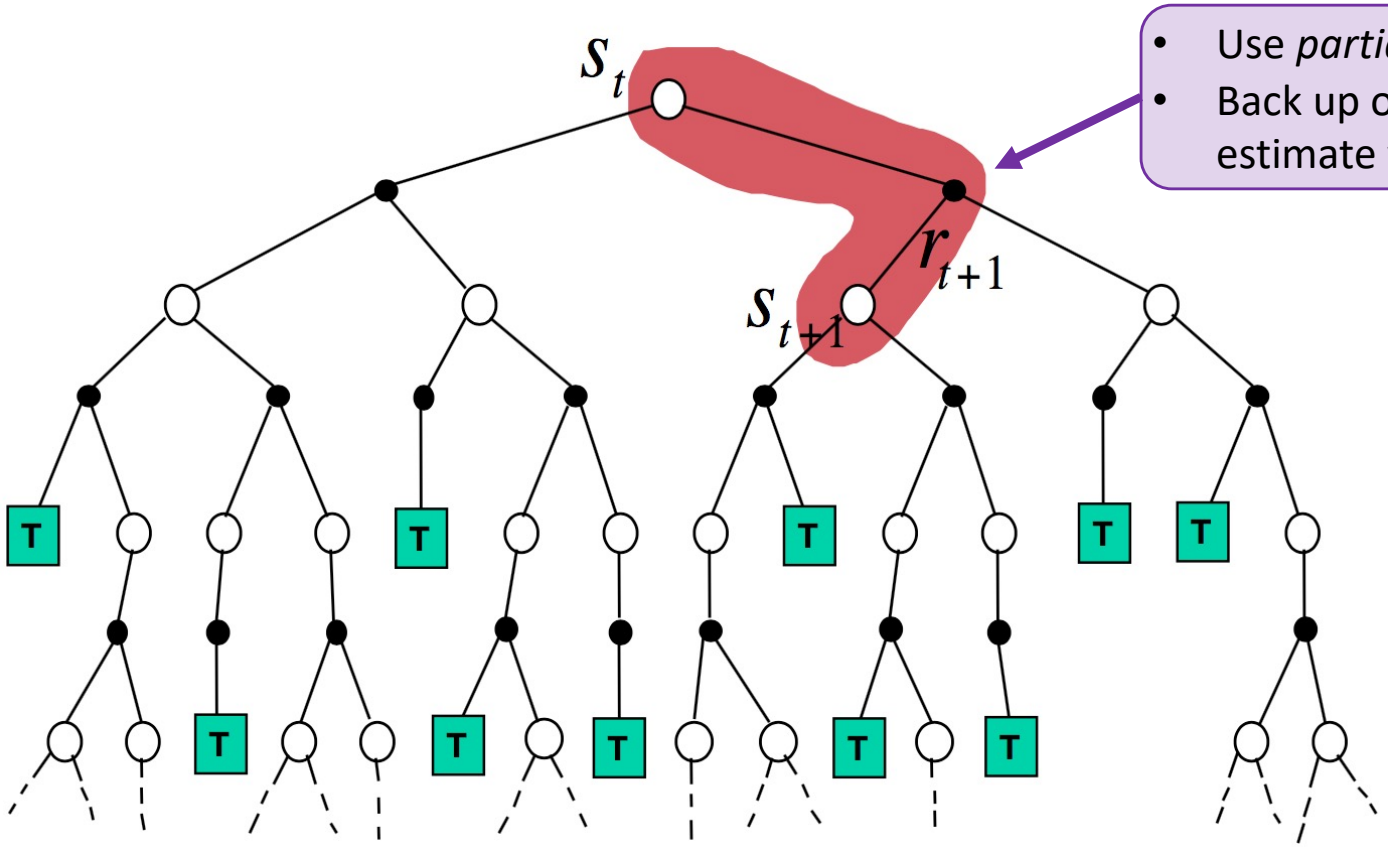
$$V(S_t) \leftarrow V(S_t) + \alpha\,(G_t - V(S_t))$$



- Run entire episode to the end
- Back up observed rewards to update values $v(S_t)$ at all S_t

Unified View: TD Learning

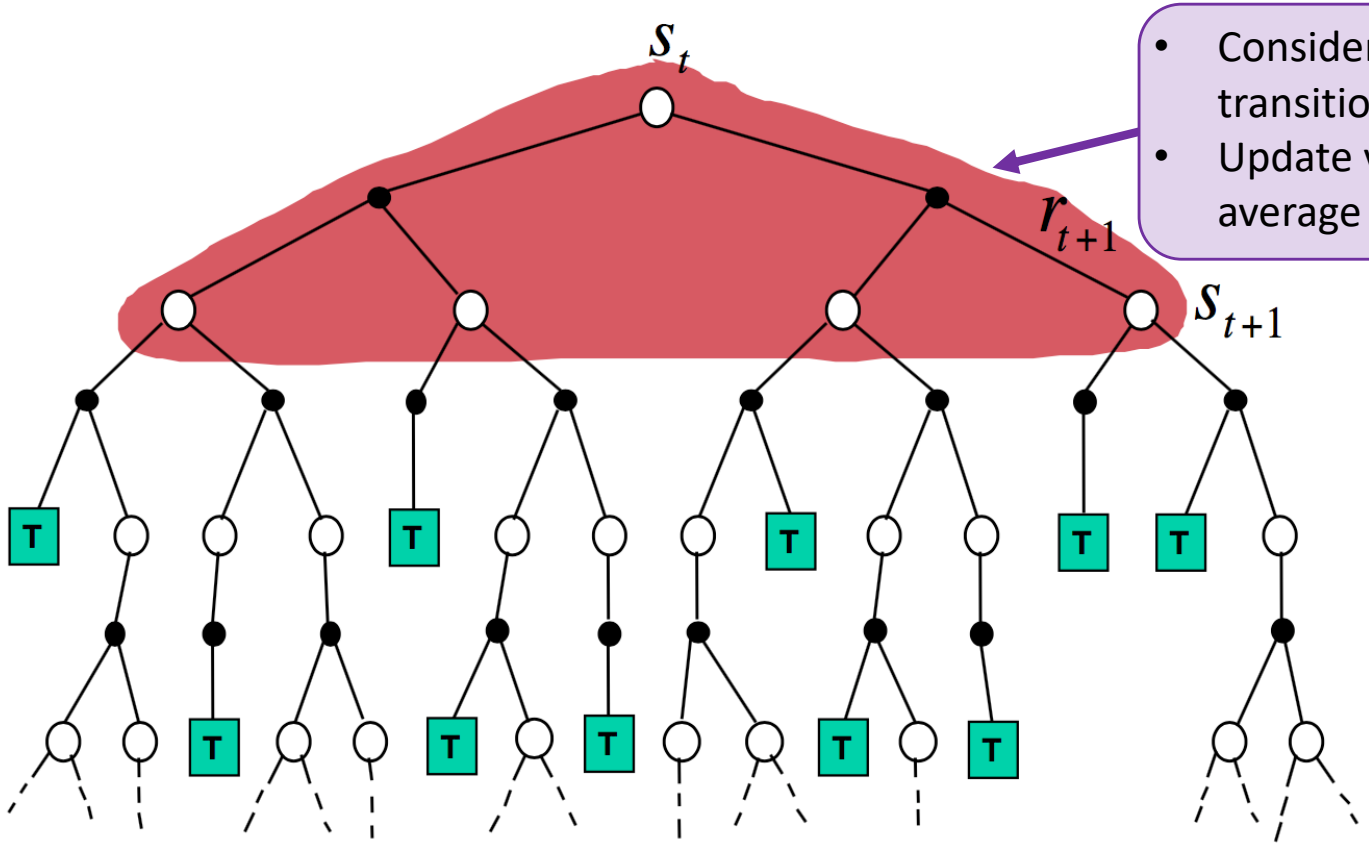
$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



- Use *partial* episode
- Back up observed rewards and old estimate $v(S_{t+1})$ to update $v(S_t)$

Unified View: Dynamic Programming

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1}) \right]$$

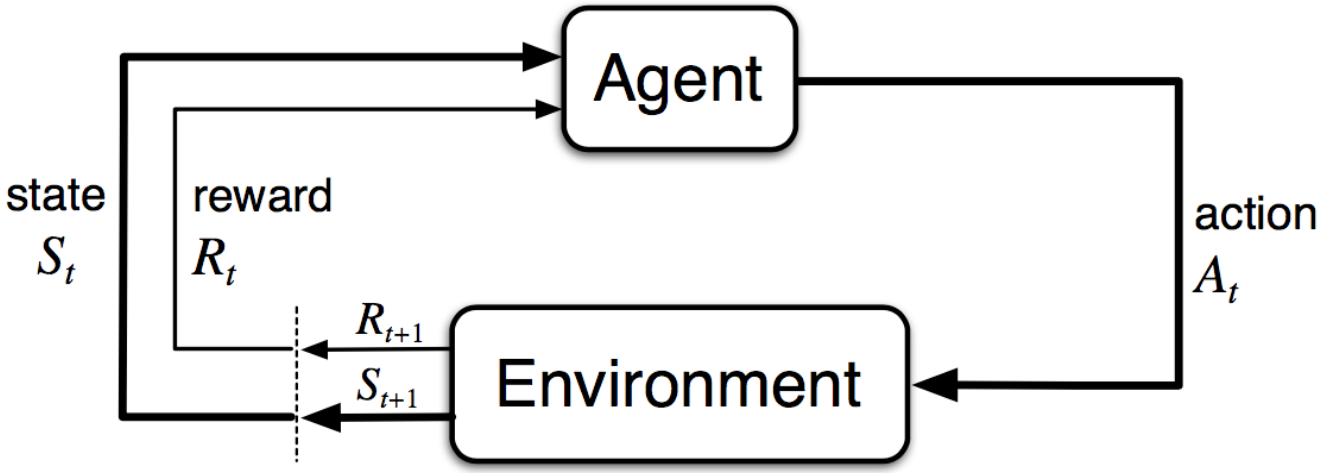


- Consider **all possible** actions & transitions (using model)
- Update $v(S_t)$ using weighted average of old estimate $v(S_{t+1})$

Overview

	Evaluate Policy, π	Find Best Policy, π^*
MDP Known	Policy Evaluation	Policy/Value Iteration
MDP Unknown	MC and TD Learning	Sarsa + Q-Learning

Model-free Control

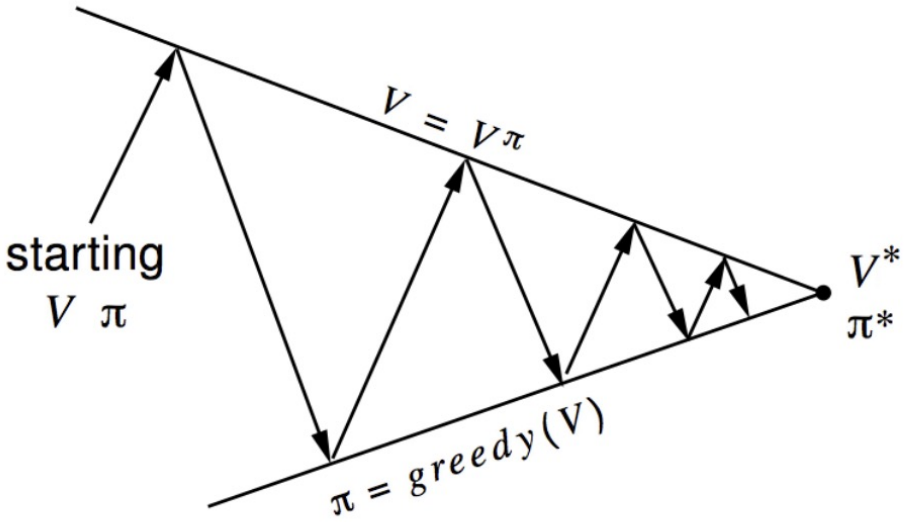


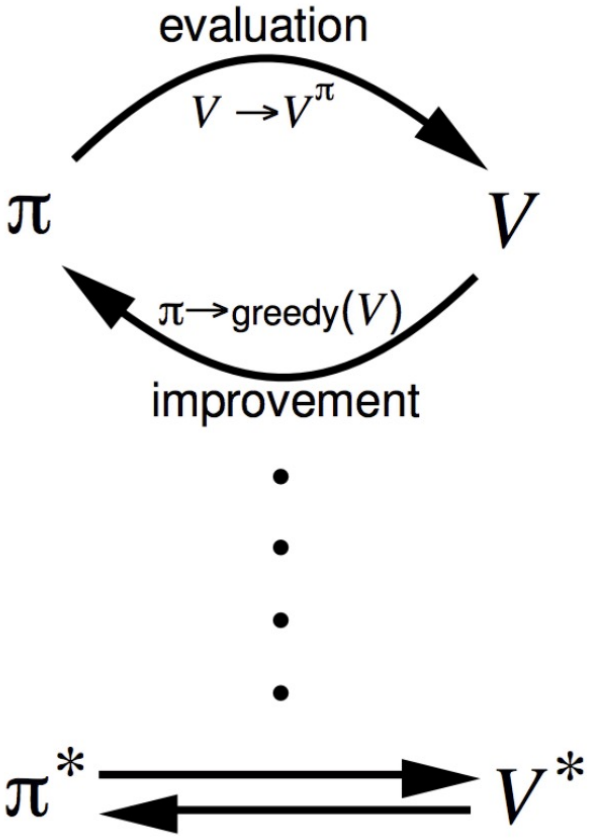
Learn a policy π to maximize rewards in the environment

On and Off Policy Learning

- On-policy learning
  - "Learn on the job"
  - Learn about policy π from experience sampled from π
- Off-policy learning
  - "Look over someone's shoulder"
  - Learn about policy π from experience sampled from μ

Generalized Policy Iteration





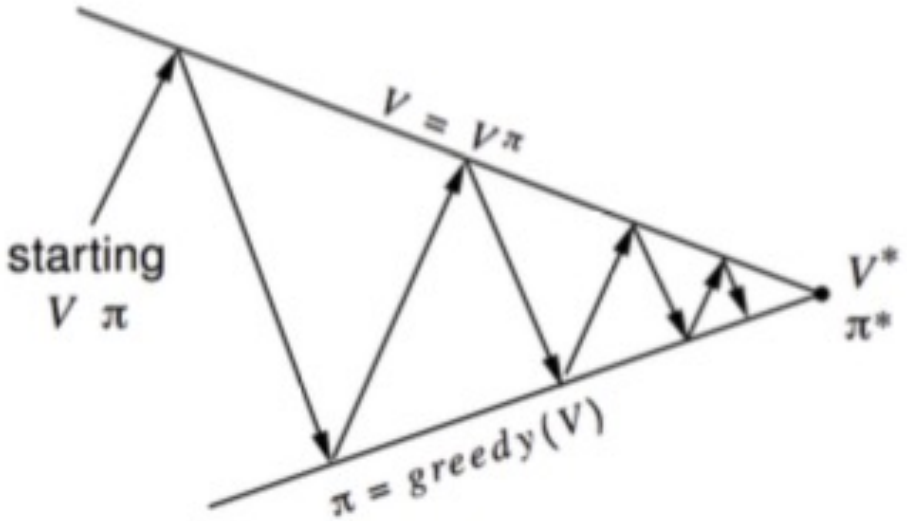
Policy evaluation Estimate v_π

e.g. Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

e.g. Greedy policy improvement

Gen Policy Improvement?



Policy evaluation Monte-Carlo policy evaluation, $V = v_\pi$?

Policy improvement Greedy policy improvement?

Not quite!

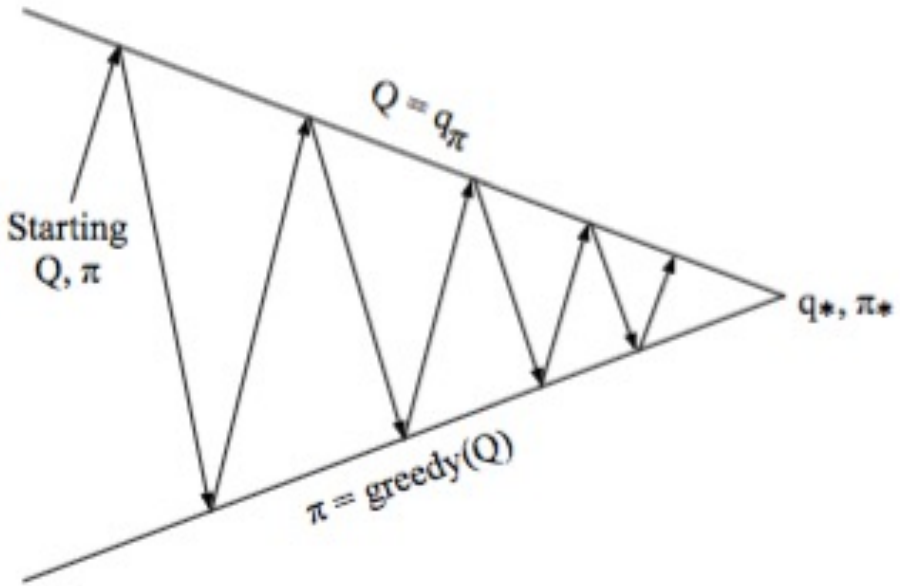
- Greedy policy improvement over $V(s)$ requires model of MDP

$$\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\, \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

- Greedy policy improvement over $Q(s, a)$ is model-free

$$\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\, Q(s, a)$$

Learn Q function directly…



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement Greedy policy improvement?

Greedy Action Selection?

- There are two doors in front of you.
- You open the left door and get reward 0
  $V(left) = 0$
- You open the right door and get reward $+1$
  $V(right) = +1$
- You open the right door and get reward $+3$
  $V(right) = +2$
- You open the right door and get reward $+2$
  $V(right) = +2$

- Are you sure you've chosen the best door?

∈-Greedy Exploration

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability

- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

Multi-armed Bandits

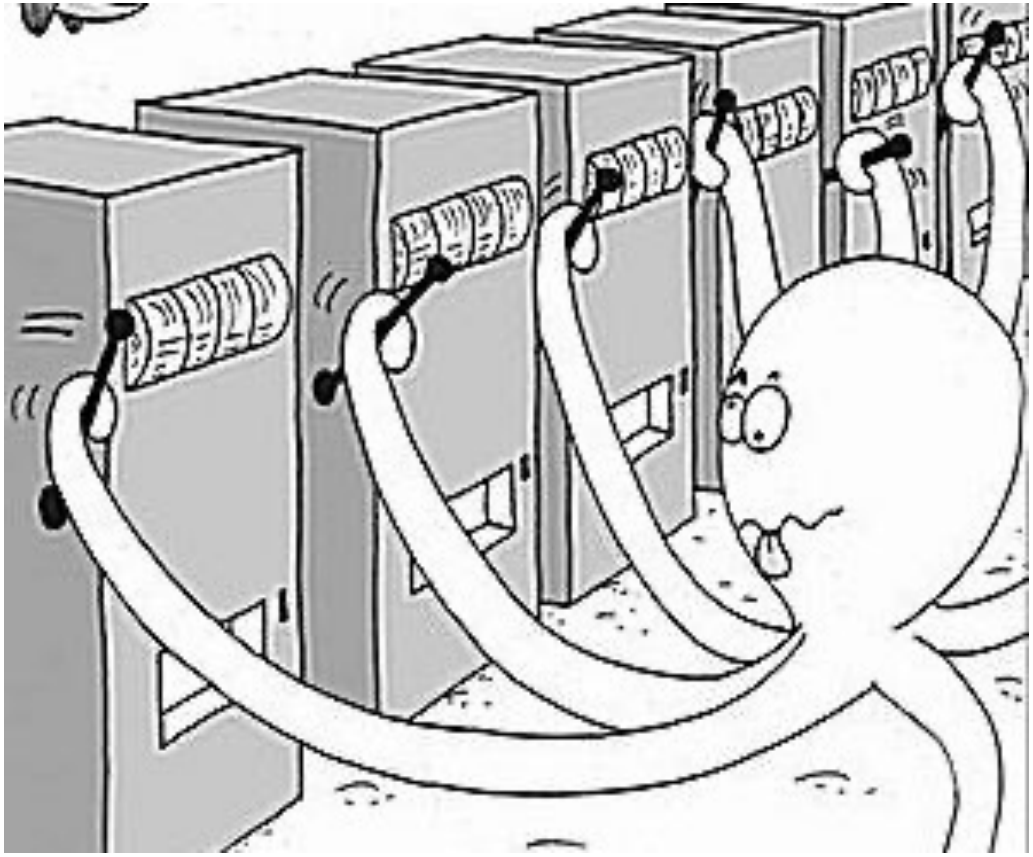


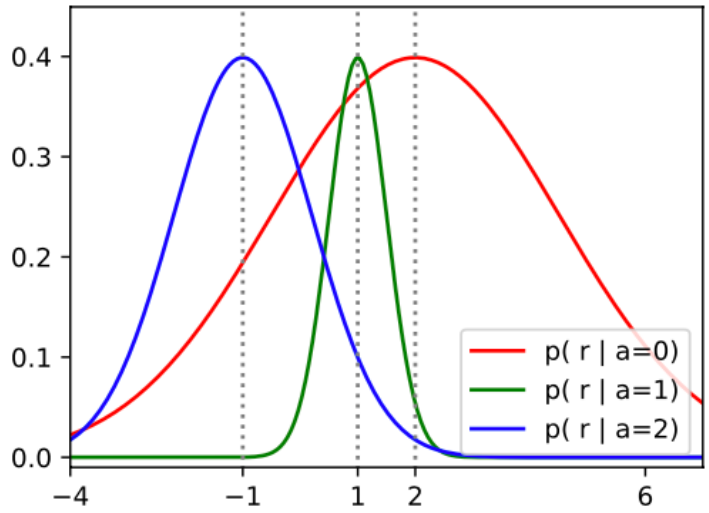
Image from Microsoft Research

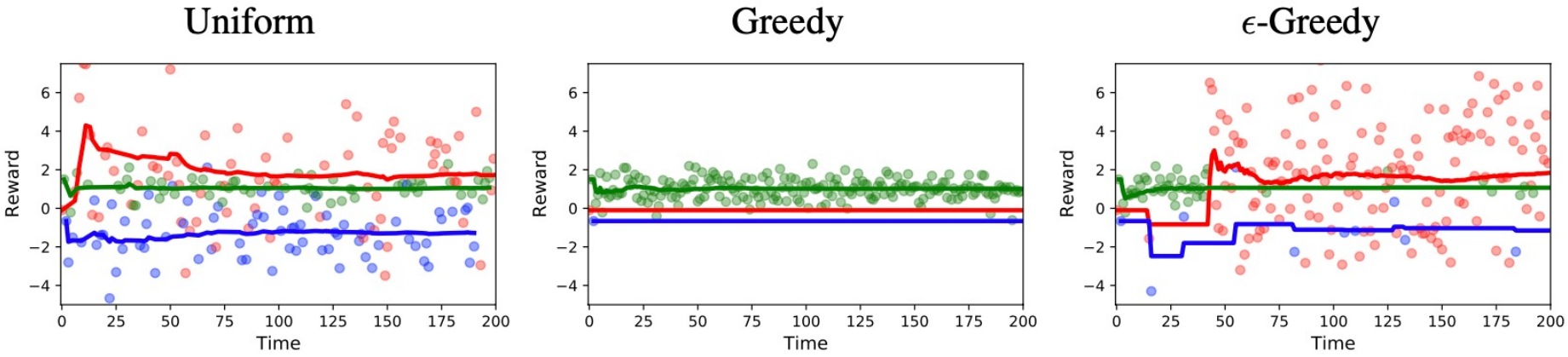
Take one of several possible actions
No temporal consequences:
- all sequences are length-1
- next episode restarts from scratch

Ex: pick the best slot machine to play in the casino
(slot machine = “one-armed bandit”)

Ex: Multi-armed Bandit

- Three actions (blue, green, red)
  - Red is best on average, but highly variable
- Uniform actions
  - Estimates all converge, but we often take suboptimal actions
- Greedy actions
  - Get stuck on good but suboptimal action
- Epsilon-Greedy
  - Eventually discover best action & exploit it





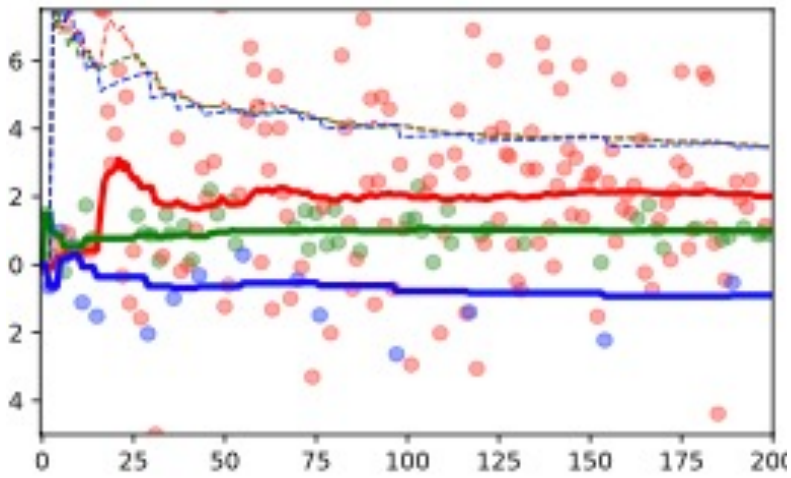
Upper-confidence bounds (UCB)

- Suppose a machine pays out either $0 or $1
  - We play 100 times, and get $0 every time
  - What's the best plausible expected payout of that machine?
  - Can't be close to $1, or we have been *extraordinarily* unlucky!

- Use probabilistic upper bounds on the possible payout
  - e.g., "with 95% probability, payout is less than $0.75"
  - Correspond to "optimistic but plausible" expected return
  - More observations gives tighter confidence bounds

- UCB method
  - Choose the action with the highest bound
  - May choose b/c action seems to do well,

  or b/c we are not yet sure how well it does:



Thompson Sampling

- Bayesian technique similar in spirit to UCB
  - Model a prior distribution over reward values
  - Condition on observations to get the posterior

- Thompson sampling:
  - Sample reward values from your posterior model
  - Choose the action with the highest sampled reward
  - Picks action proportionally to (the model's) probability it is the best

  - Like UCB, we may pick an action if that action has provided high rewards, or if we have a lot of uncertainty about its rewards.

Which Policy Evaluation for Learning?

- Temporal difference (TD) has some advantages
  - Lower variance than Monte Carlo
  - Can use online, on incomplete sequences

- A natural idea: use TD inside our control loop
  - Apply TD to update q(s,a)
  - Improve policy by updating to ε-greedy on q(.)
  - Update q & π each time step

Sarsa: TD for Policy Evaluation

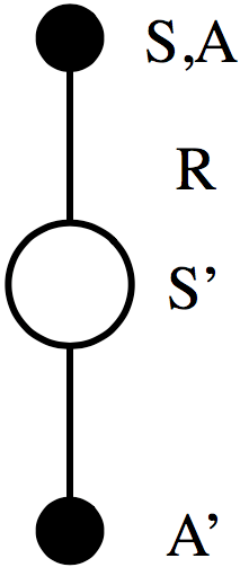
- At each time step, we observe a snippet of experience:
  - State s; took action a; got reward r; moved to state s', took action a'
  - Can update q(s,a) based on this experience:

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$

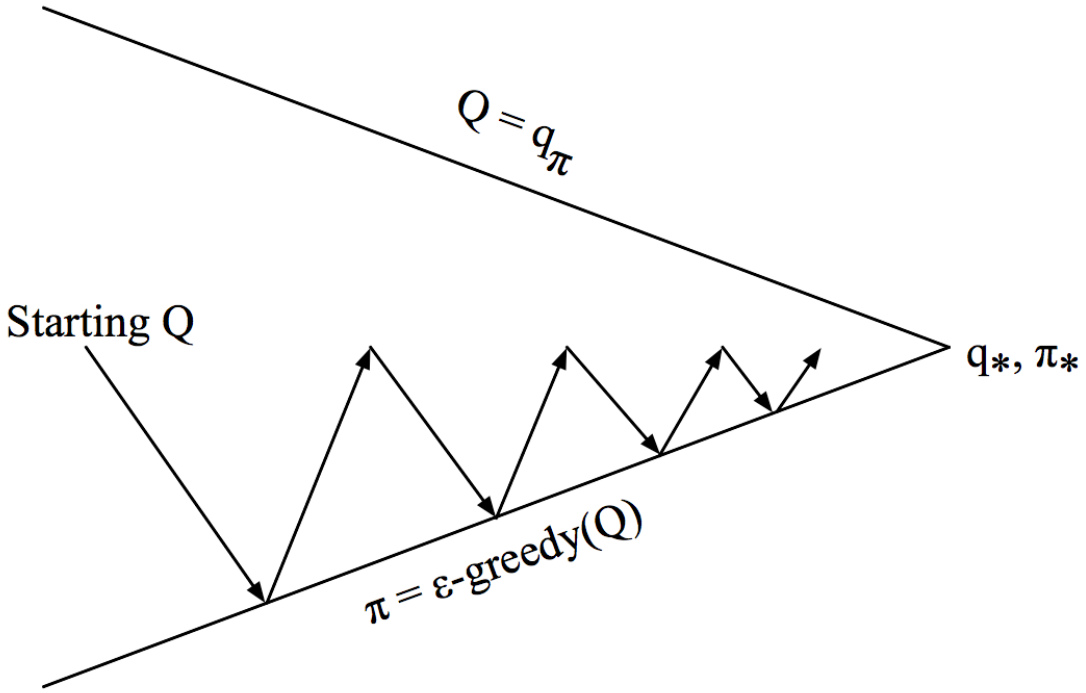
step size

predicted return for this snippet

old predicted return



On-Policy Control w/ Sarsa



Every time-step:

Policy evaluation Sarsa, $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$
- While following behaviour policy $\mu(a|s)$

$$\{S_1, A_1, R_2, ..., S_T\} \sim \mu$$

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$
- While following behaviour policy $\mu(a|s)$

$$\{S_1, A_1, R_2, ..., S_T\} \sim \mu$$

- Why is this important?
- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, ..., \pi_{t-1}$
- Learn about *optimal* policy while following *exploratory* policy
- Learn about *multiple* policies while following *one* policy

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$

- Next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot|S_t)$

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$

- Next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot|S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t) \right)$$

Off-Policy w/ Q-Learning

- We now allow both behaviour and target policies to improve

Off-Policy w/ Q-Learning

- We now allow both behaviour and target policies to improve
- The target policy π is greedy w.r.t. $Q(s, a)$

$$\pi(S_{t+1}) = \underset{a'}{\text{argmax}}\ Q(S_{t+1}, a')$$

- The behaviour policy μ is e.g. ϵ-greedy w.r.t. $Q(s, a)$

Off-Policy w/ Q-Learning

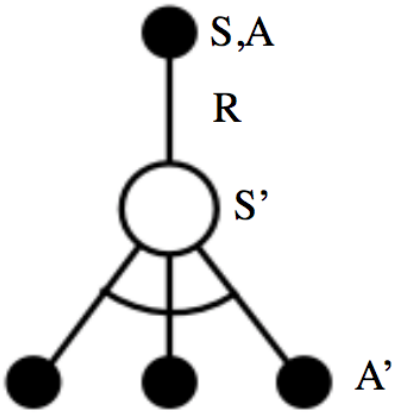
- We now allow both behaviour and target policies to improve
- The target policy π is greedy w.r.t. $Q(s, a)$

$$\pi(S_{t+1}) = \underset{a'}{\text{argmax}}\ Q(S_{t+1}, a')$$

- The behaviour policy μ is e.g. ϵ-greedy w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ =& R_{t+1} + \gamma Q(S_{t+1}, \underset{a'}{\text{argmax}}\ Q(S_{t+1}, a')) \\ =& R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

Q-Learning Control Algorithm

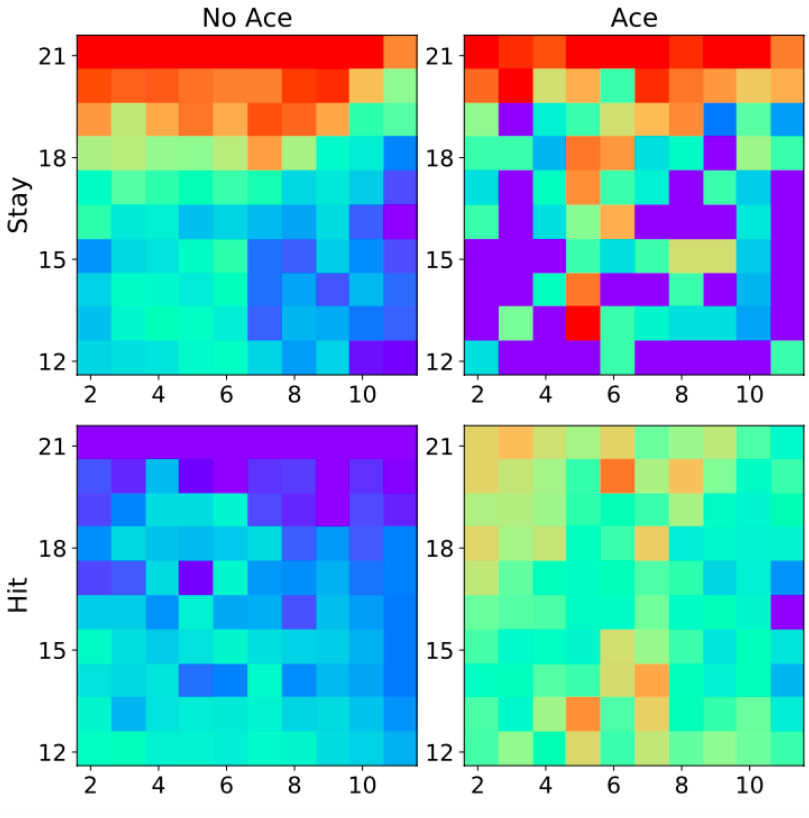


(SARSAMAX)

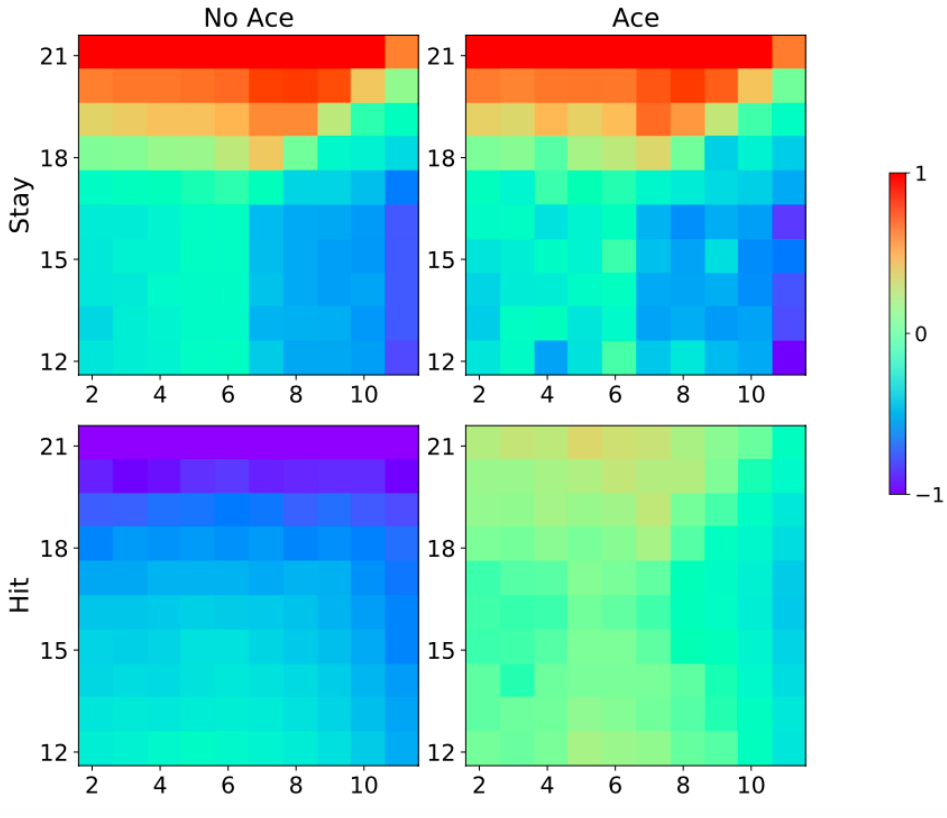
$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S,A) \right)$$

Ex: Blackjack

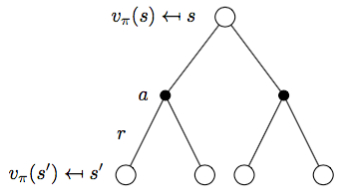
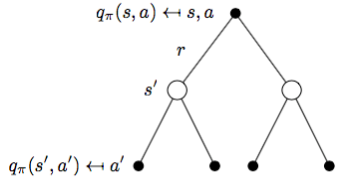
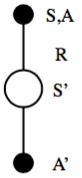
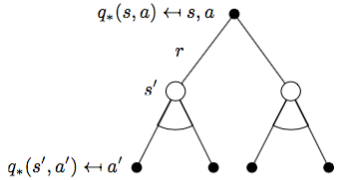
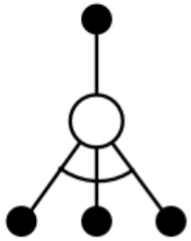
$\mathbf{m = 10,000:}$



$\mathbf{m = 500,000:}$



Relation between DP and TD

	Full Backup (DP)	*Sample Backup (TD)*
Bellman Expectation Equation for $v_\pi(s)$	 Iterative Policy Evaluation	 TD Learning
Bellman Expectation Equation for $q_\pi(s, a)$	 Q-Policy Iteration	 Sarsa
Bellman Optimality Equation for $q_*(s, a)$	 Q-Value Iteration	 Q-Learning

Update Eqns for DP and TD

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation	TD Learning
$V(s) \leftarrow \mathbb{E}\left[R + \gamma V(S') \mid s\right]$	$V(S) \overset{\alpha}{\leftarrow} R + \gamma V(S')$
Q-Policy Iteration	Sarsa
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma Q(S',A') \mid s,a\right]$	$Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma Q(S',A')$
Q-Value Iteration	Q-Learning
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	$Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$

where $x \overset{\alpha}{\leftarrow} y \equiv x \leftarrow x + \alpha(y - x)$