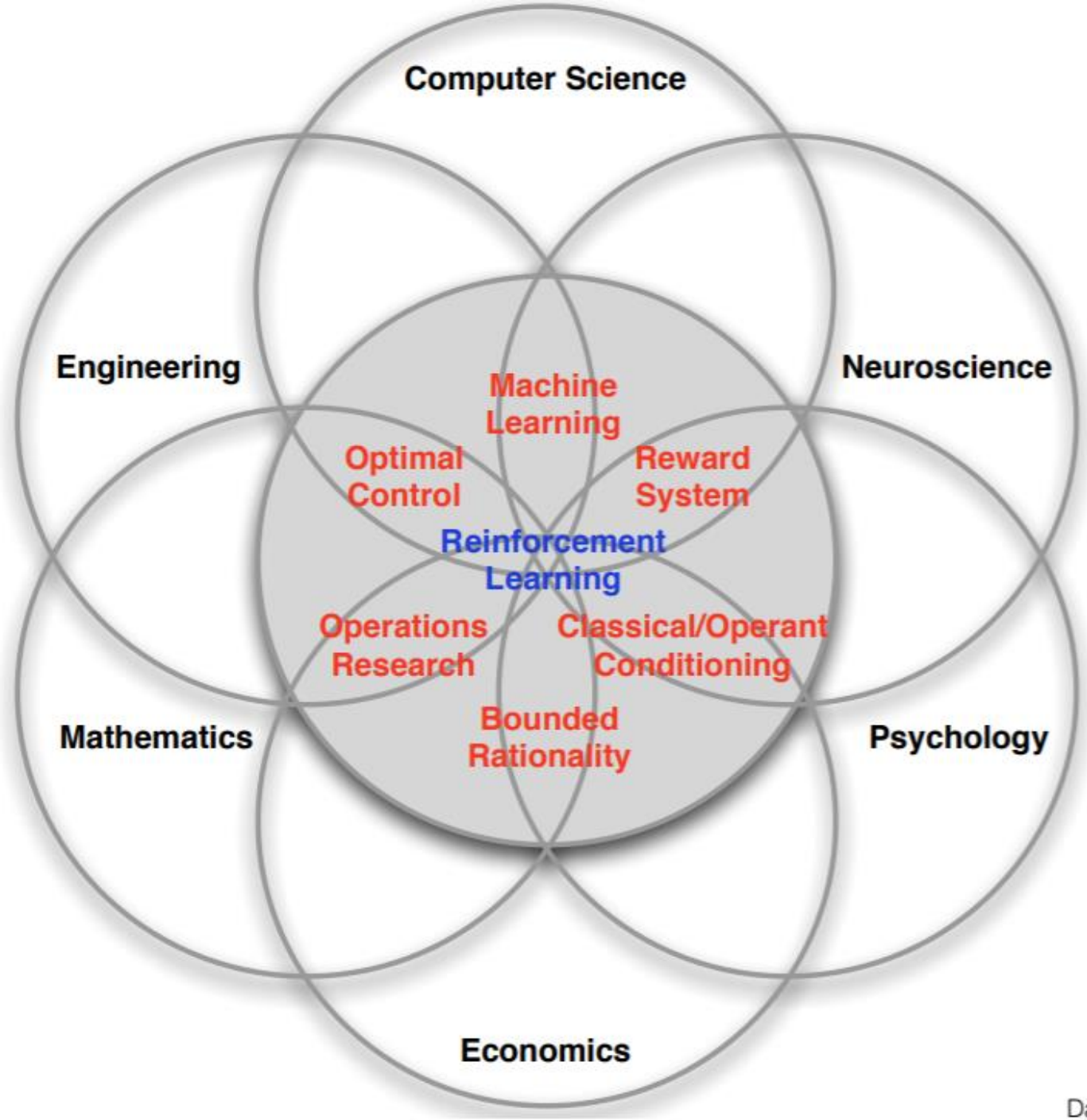
Basics of Reinforcement Learning*

by Alina Vereshchaka

*Slides are based on David Silver's course "Reinforcement Learning" and the book by Sutton, Richard S., and Andrew G. Barto. "Reinforcement Learning: An introduction (2 edition)" (2018)

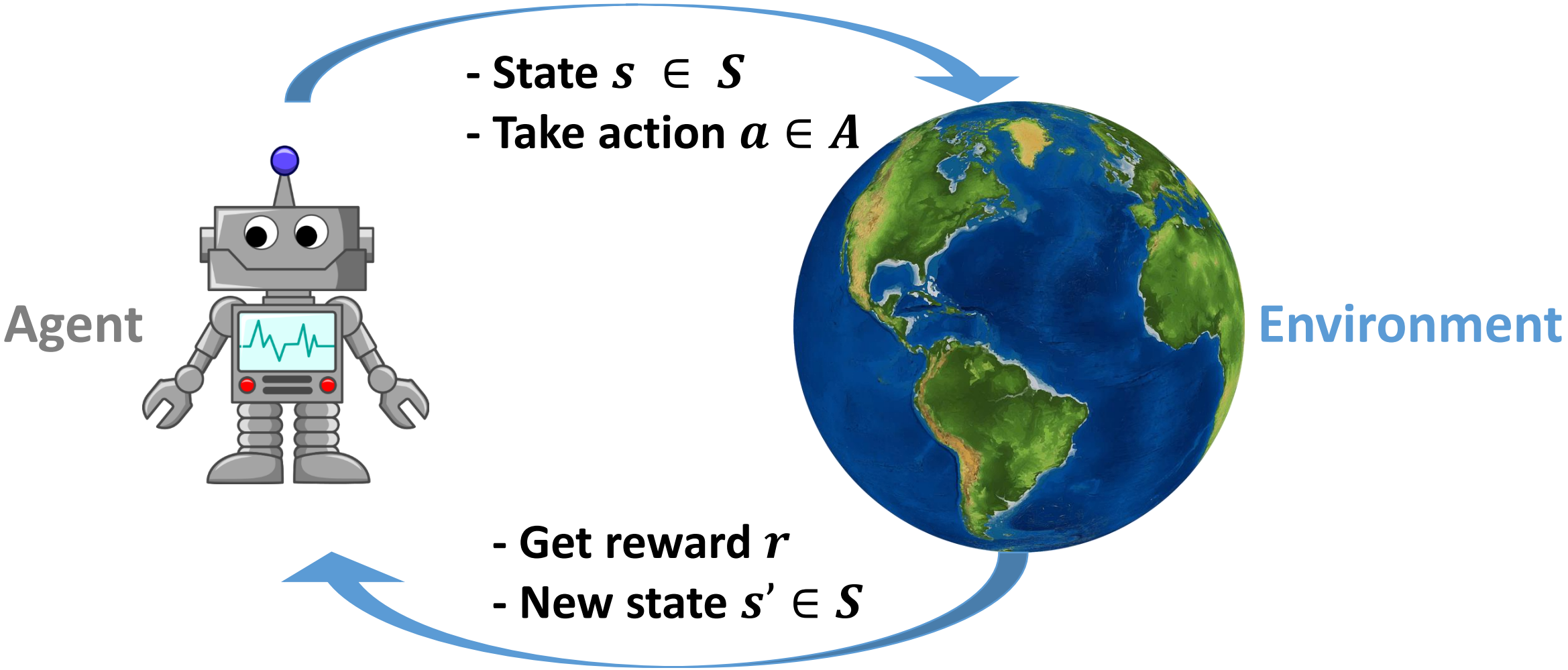
Outline

- Markov Decision Process Definition
- RL Problem Definition
- Dynamic Programming
- Monte-Carlo Learning
- Temporal-Difference Learning
- SARSA
- Q-Learning
- n-Step Prediction



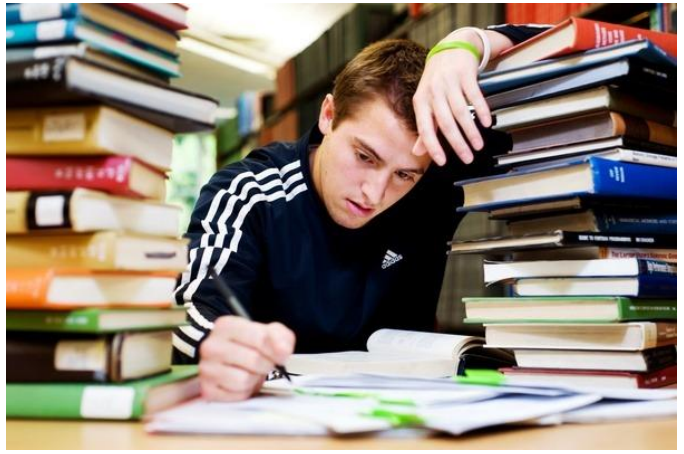
David Silver 2015

What is Reinforcement learning?

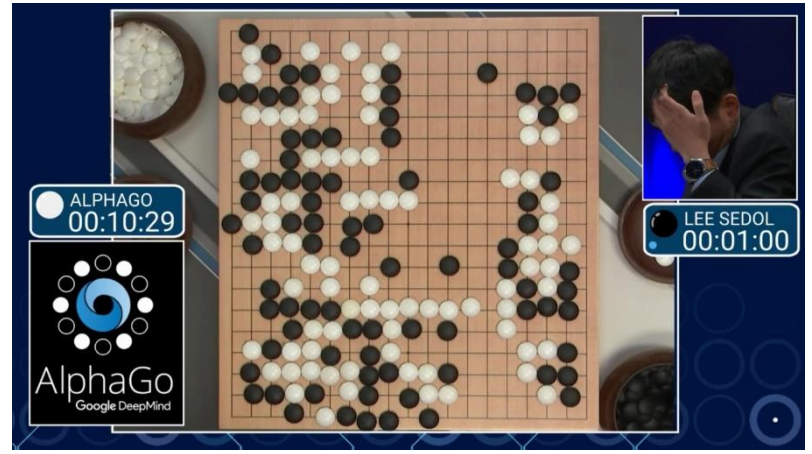


Reinforcement Learning in real world

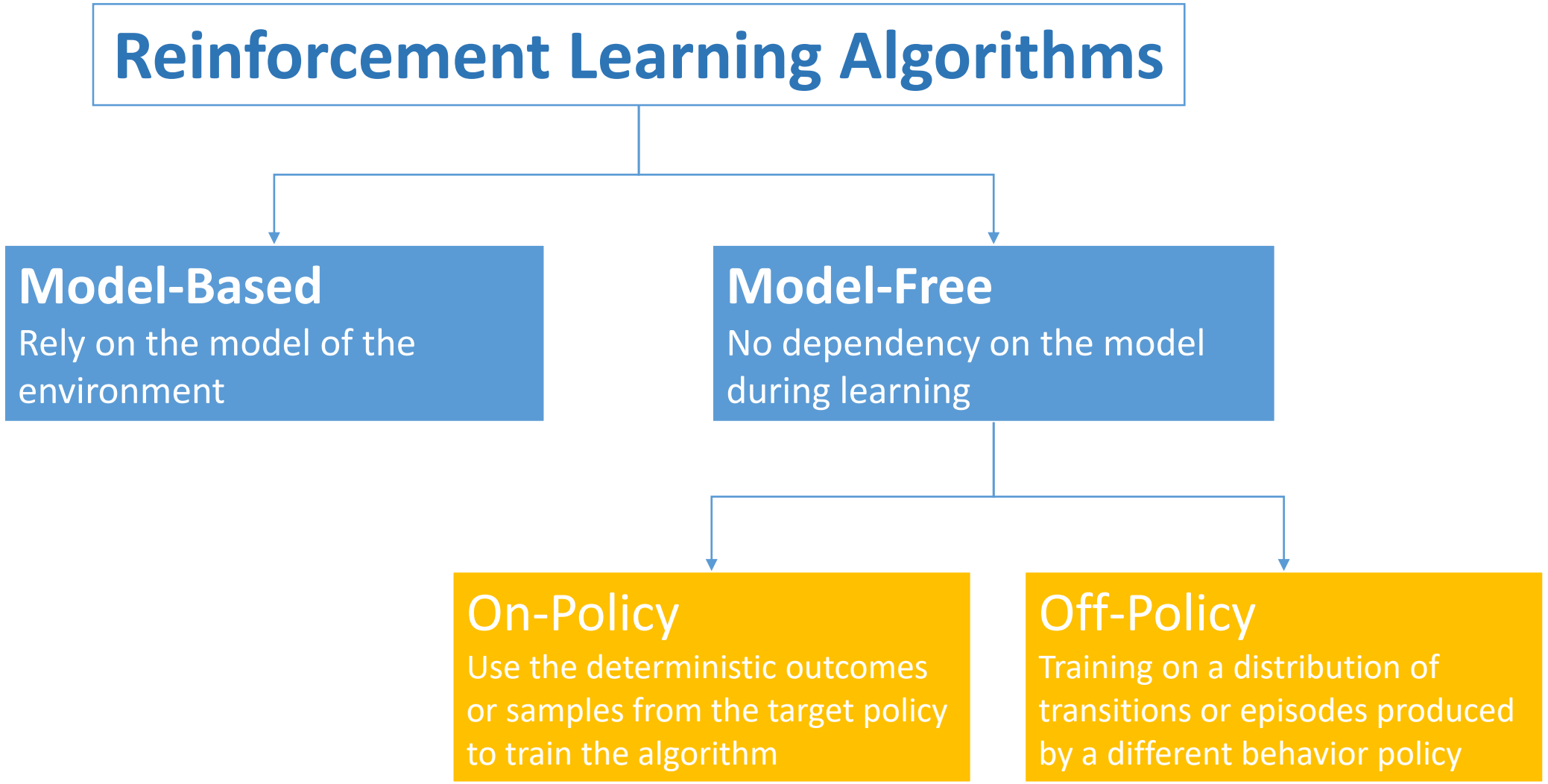
State: tasty smell, sound
Actions: give a paw
Reward: food



State: want a good grade
Actions: studying
Reward: good grade



State: opponent movement, current board state
Actions: make a move
Reward: points


Reinforcement Learning Algorithms
Model-Based
Rely on the model of the environment
Model-Free
No dependency on the model during learning
On-Policy
Use the deterministic outcomes or samples from the target policy to train the algorithm
Off-Policy
Training on a distribution of transitions or episodes produced by a different behavior policy

Markov Chain

Definition

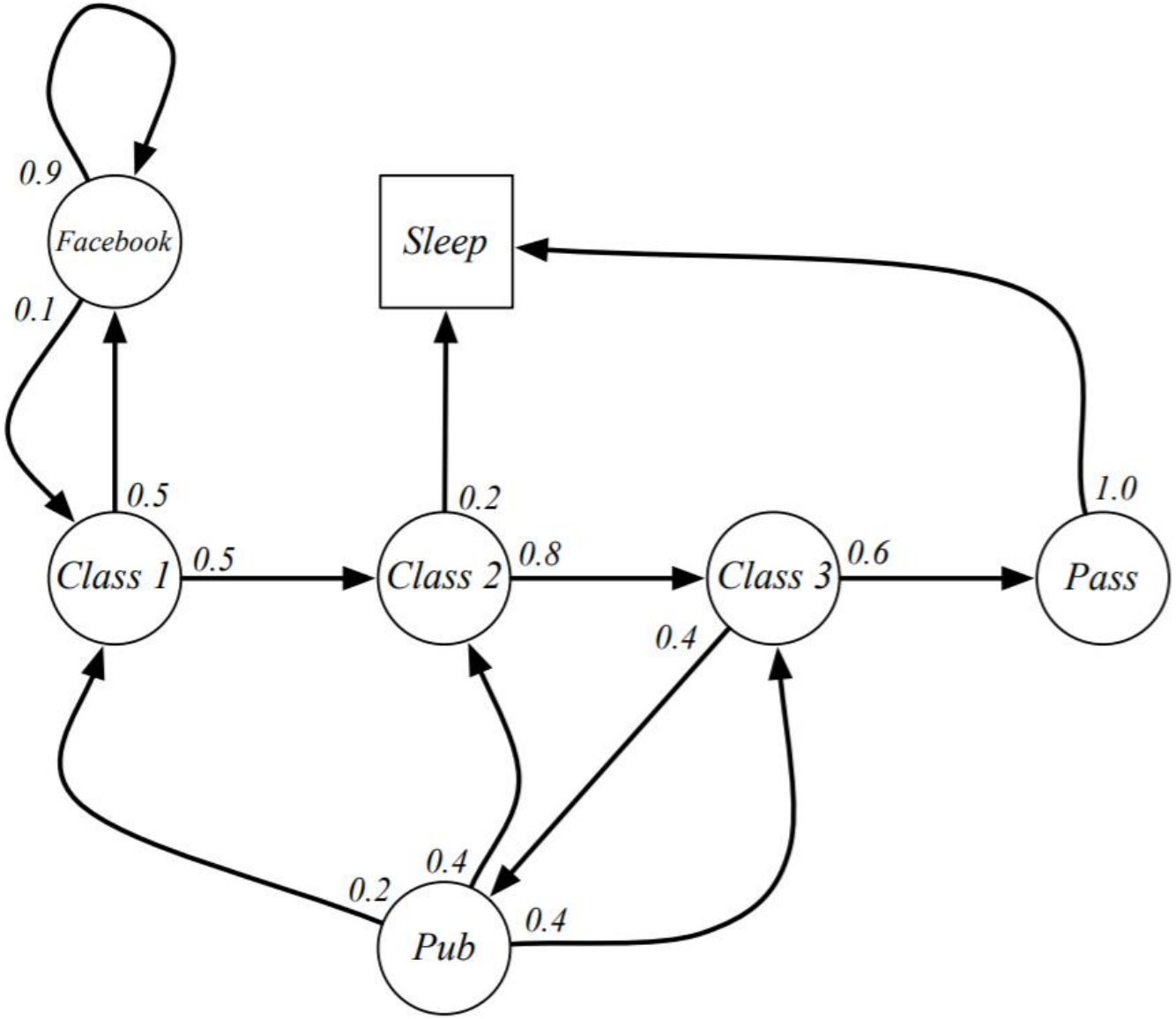
A *Markov Process* (or *Markov Chain*) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$

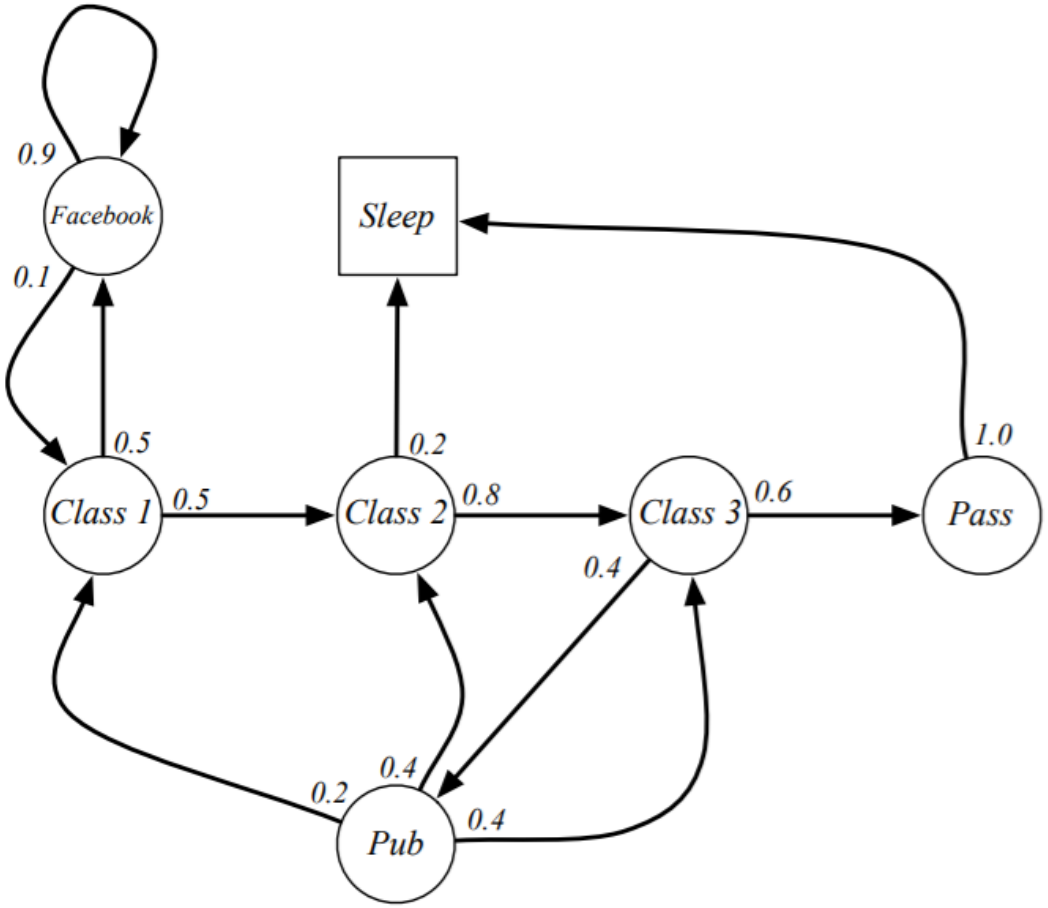
Markovian property: $\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$

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

Example: Student Markov Chain



Example: Student Markov Chain Episodes

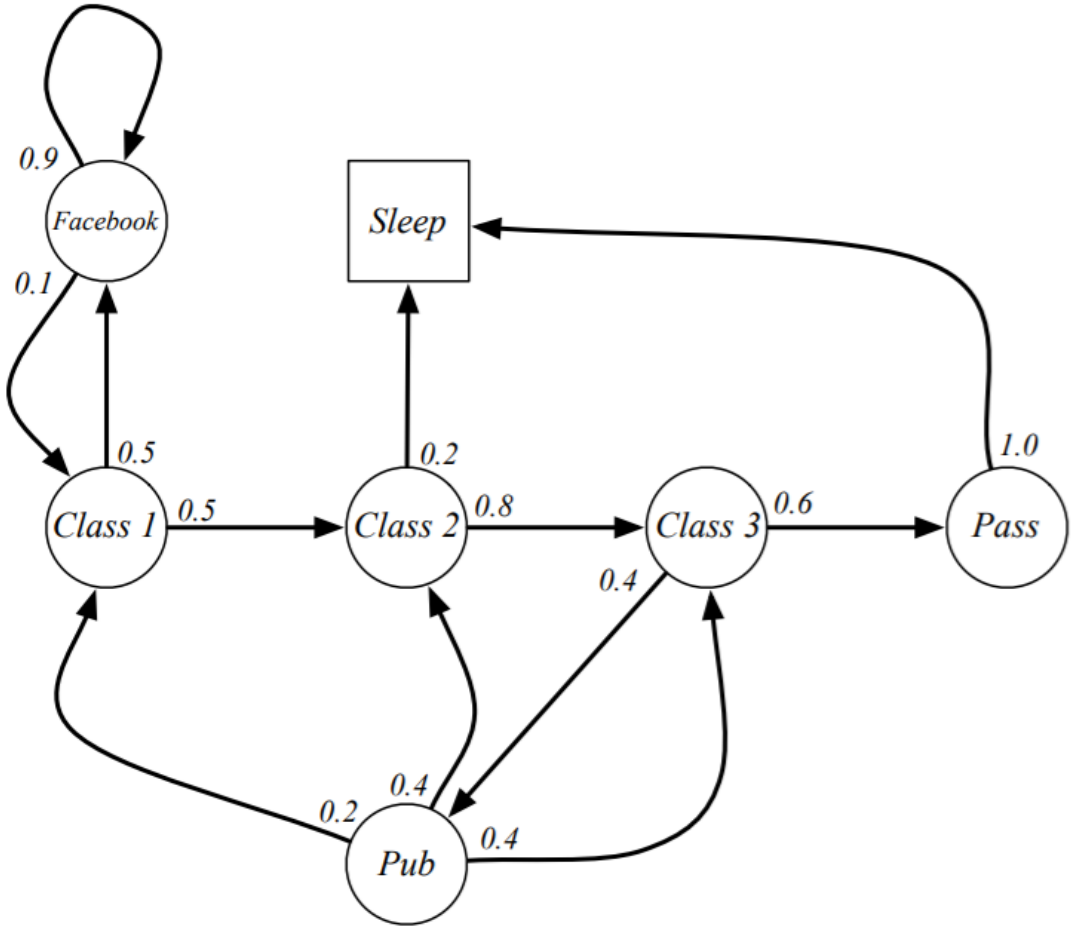


Sample episodes for Student Markov Chain starting from $S_1 = \text{C1}$

$$S_1, S_2, ..., S_T$$

- 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

Example: Transition Matrix



$$\mathcal{P} = \begin{array}{c} \\ C1 \\ C2 \\ C3 \\ Pass \\ Pub \\ FB \\ Sleep \end{array} \begin{array}{c} \begin{array}{ccccccc} C1 & C2 & C3 & Pass & Pub & FB & Sleep \end{array} \\ \begin{bmatrix} & 0.5 & & & & 0.5 & \\ & & 0.8 & & & & 0.2 \\ & & & 0.6 & 0.4 & & \\ & & & & & & 1.0 \\ 0.2 & 0.4 & 0.4 & & & & \\ 0.1 & & & & & 0.9 & \\ & & & & & & 1 \end{bmatrix} \end{array}$$

Markov Reward Process (MRP)

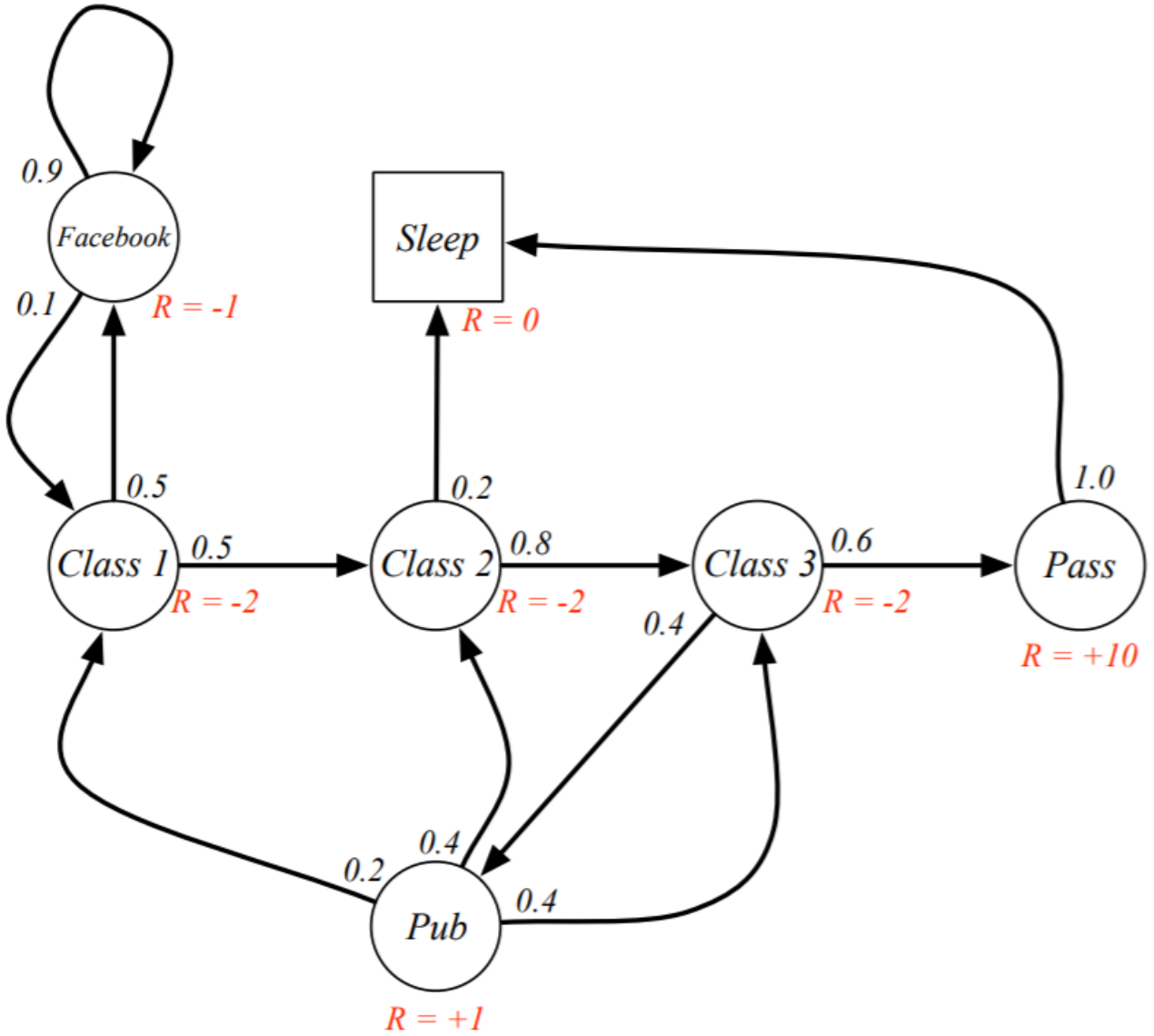
A Markov reward process is a Markov chain with values.

Definition

A *Markov Reward Process* is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s]$
- γ is a discount factor, $\gamma \in [0, 1]$

Example: Student MRP



Discounting factor $\boldsymbol{\gamma}$

The **discounting factor** $(\gamma \in [0, 1))$ penalize the rewards in the future. Reward at time **k** worth only $\boldsymbol{\gamma^{k-1}}$

Motivation:

- The future rewards may have higher uncertainty (stock market)
- The future rewards do not provide immediate benefits (As human beings, we might prefer to have fun today rather than 5 years later ;)
- Discounting provides mathematical convenience (we don't need to track future steps infinitely to compute return)
- It is sometimes possible to use undiscounted Markov reward processes (i.e. $\gamma = 1$), e.g. if all sequences terminate.

Return (G_t)

Definition

The *return* G_t is the total discounted reward from time-step t.

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

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

Bellman Equation for Return

Expected reward is a total sum of discounted rewards going forward.

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\
&= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \\
&= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

Example: Returns for Student MRP

Sample returns for Student MRP:
Starting from $S_1 = \text{C1}$ with $\gamma = \frac{1}{2}$

$$G_1 = R_2 + \gamma R_3 + ... + \gamma^{T-2} R_T$$

C1 C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8}$	$=$	-2.25
C1 FB FB C1 C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16}$	$=$	-3.125
C1 C2 C3 Pub C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \ldots$	$=$	-3.41
C1 FB FB C1 C2 C3 Pub C1 ...	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \ldots$	$=$	-3.20
FB FB FB C1 C2 C3 Pub C2 Sleep			

Value Function **v(s)**

The value function $v(s)$ gives the long-term value of state s

Definition

The *state value function* $v(s)$ of an MRP is the expected return starting from state s

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

Bellman Equation for MRPs

The value function can be decomposed into two parts:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

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

Markov Decision Process

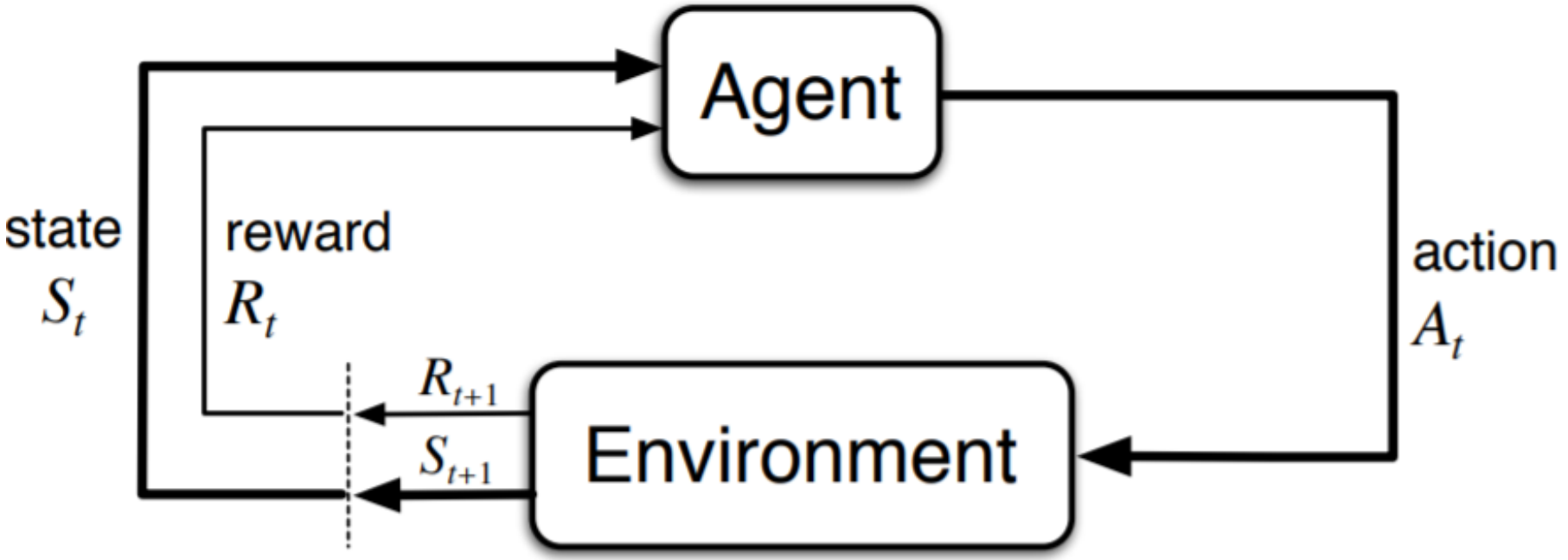
A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

Definition

A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

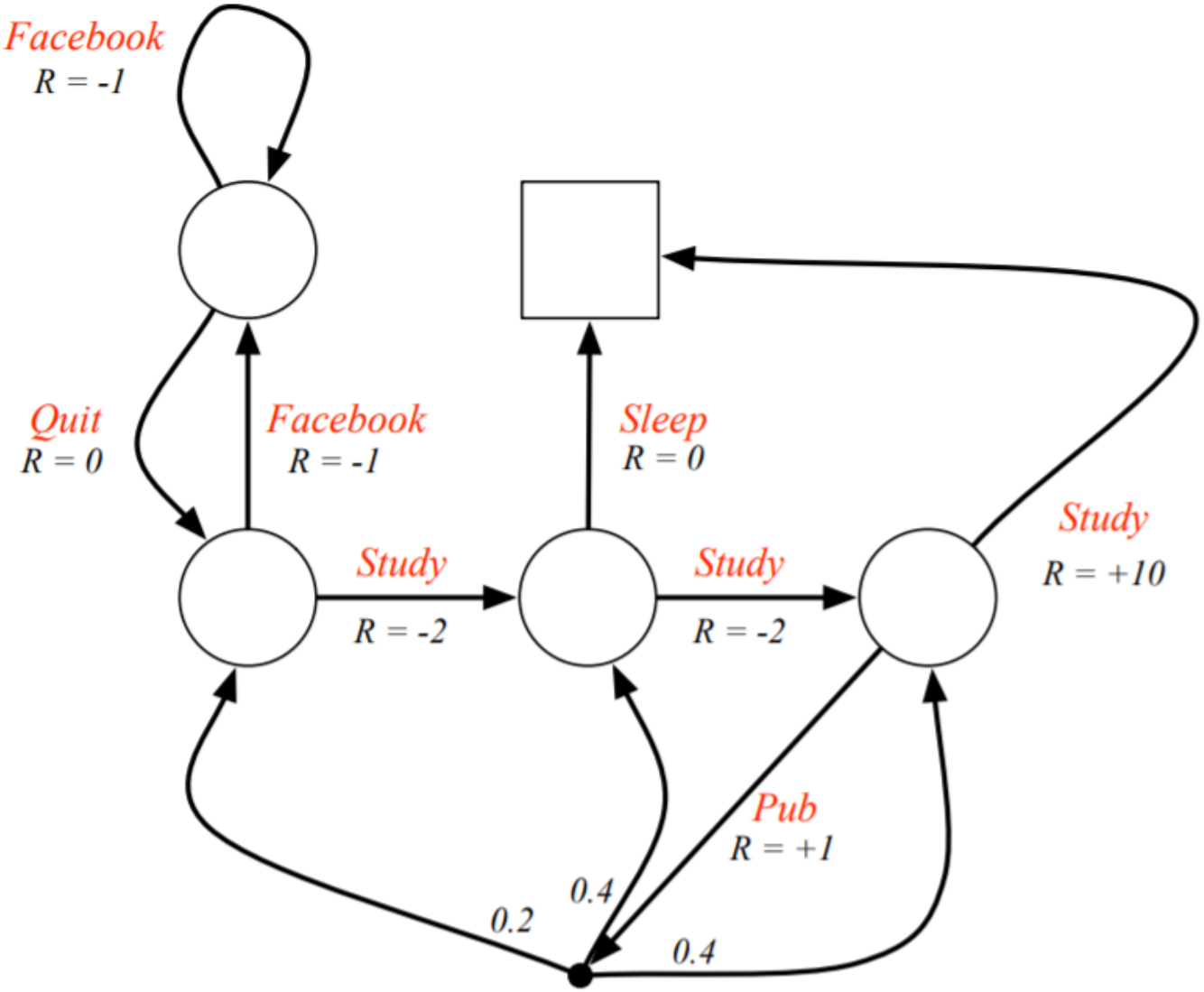
- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^{a}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^{a}_{s} = \mathbb{E}\left[R_{t+1}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

Finite Markov Decision Processes (MDP)



Daily life trajectory: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, \ldots, S_T$

Example: Student MDP



Policy ($\boldsymbol{\pi}$)

Definition

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$$

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
- i.e. Policies are *stationary* (time-independent),
  $A_t \sim \pi(\cdot|S_t), \forall t > 0$

Value Functions $v_\pi(s), q_\pi(s, a)$

Definition

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

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

Definition

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

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

Bellman Expectation Equation

The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

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

The action-value function can similarly be decomposed,

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

Optimal Value Function

Definition

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

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

The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

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

Finding an Optimal Policy

An optimal policy can be found by maximising over $q_*(s, a)$,

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_*(s, a) \\ 0 & \textit{otherwise} \end{cases}$$

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy

POMDPs

A Partially Observable Markov Decision Process is an MDP with hidden state. It is a hidden Markov model with actions.

Definition

A *POMDP* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{O}$ is a finite set of observations
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
- $\mathcal{Z}$ is an observation function, $\mathcal{Z}^a_{s'o} = \mathbb{P}[O_{t+1} = o \mid S_{t+1} = s', A_t = a]$
- γ is a discount factor $\gamma \in [0, 1]$.

MDP: Definitions

Episode **(s, a, s', r)** - agent in state **s** takes action **a** to arrive in the next state **s'** and obtain reward **r**

Transition probability function P(s', r|s, a):

$$\mathrm{P}(\mathrm{s'},\ \mathrm{r}\ |s,a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a]$$

$$\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s', r|s, a) = 1$$

State-transition function P(s'|s, a):

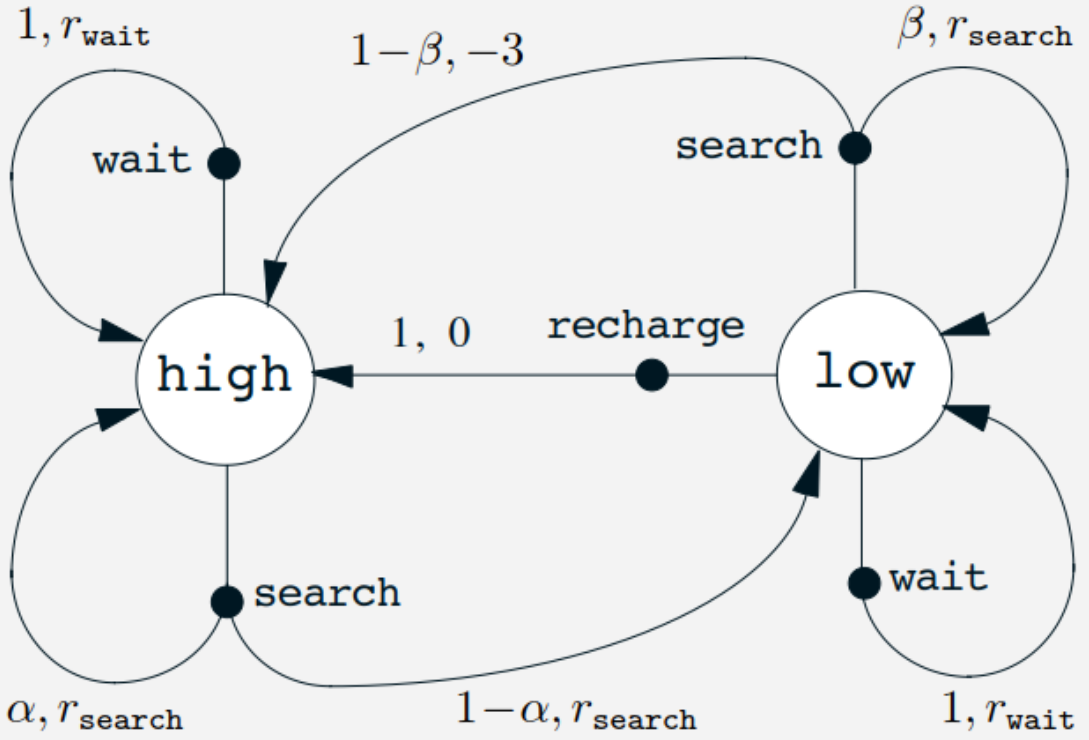
$$\mathrm{P}(\mathrm{s'}|s,a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] = \sum_{r\in\mathcal{R}} P(s', r|s, a)$$

Expected reward function R(s, a):

$$\mathrm{R}(\mathrm{s},\ \mathrm{a}) = \mathrm{E}\ [\mathrm{R}_{t+1}|S_t = s, A_t = a] = \sum_{r\in\mathcal{R}} r\sum_{s'\in\mathcal{S}} P(s', r|s, a)$$

Case study: Recycling Robot

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1-\alpha$	$r_{\texttt{search}}$
low	search	high	$1-\beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	-



Policy

- A policy is the agent's behavior
- It is a map from state to action
- It can be either deterministic or stochastic:

Deterministic:

$$\pi(s) = a$$

Stochastic:

$$\pi(a|s) = \mathbb{P}_\pi[A = a|S = s]$$

Action-value method (Q)

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

As the denominator goes to infinity, by the law of large numbers, $Q_t(a)$ converges to $q^*(a)$- *sample-average method* for estimating action values because each estimate is an average of the sample of relevant rewards.

Greedy action: $A_t \doteq \arg\max_a Q_t(a)$

Definitions

State-value function:

$$V_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s]$$
$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right], \text{for all } s \in S$$

Action-value function:

$$Q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$
$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right]$$

State-value function w.r.t. probability distribution over possible actions and the Q-values:

$$\mathrm{V}_\pi(s) = \sum_{a\in\mathcal{A}} Q_\pi(s,a)\pi(a|s)$$

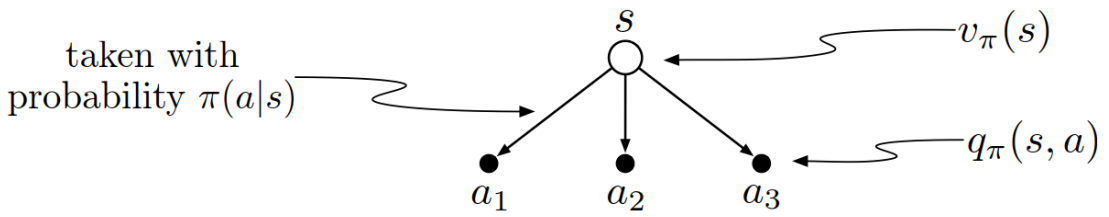
Action advantage function:

$$\mathrm{A}_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

Bellman Equation for state-value function

Bellman equation decomposes the value function into the immediate reward plus the discounted future values.

$$
\begin{aligned}
V_\pi(s) &\doteq \mathbb{E}_\pi[G_t|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots |S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\Big] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma V_\pi(s')\Big], \text{for all } s \in S
\end{aligned}
$$



Similarly for Q-value:

$$Q(s, a) = \mathrm{E}\,[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$$

Optimal Value and Policy

- The optimal value function produces the maximum return:

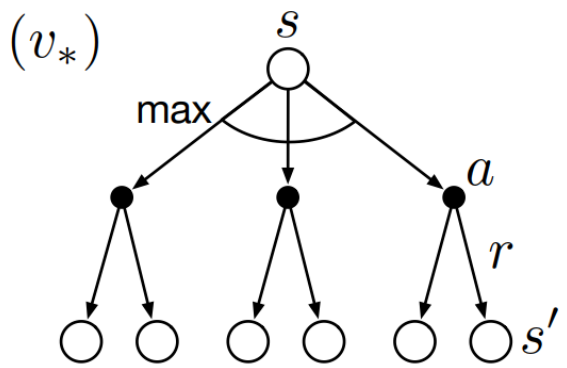
$$\mathrm{V}_*(s) = \max_\pi V_\pi(s)$$
$$Q_*(s,a) = \max_\pi Q_\pi(s,a)$$

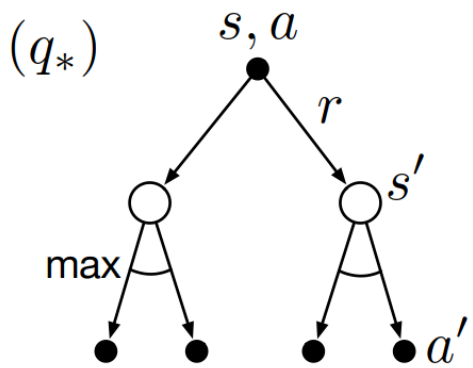
- The optimal policy achieves optimal value functions:

$$\pi_* = \arg\max_\pi V_\pi(s)$$
$$\pi_* = \arg\max_\pi Q_\pi(s,a)$$

- Thus we have

$$\mathrm{V}_{\pi_*}(s) = V_*(s)$$
$$\mathrm{Q}_{\pi_*}(s,a) = Q_*(s,a)$$





Bellman optimality equations

State-Value:

$$
\begin{aligned}
V_\pi(s) &= \max_{a \in A(s)} Q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi *}[G_t | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi *}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi *}[R_{t+1} + \gamma V_*(S_{t+1}) | S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r | s, a) \Big[r + \gamma V_*(s') \Big]
\end{aligned}
$$

State-Action:

$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a \Big] \\
&= \sum_{s',r} p(s', r, a) \Big[r + \gamma Q_*(s', a') \Big]
\end{aligned}
$$

Dynamic Programming (DP)

- DP is a model-based approach
- A method for solving complex problems, by breaking them down into subproblems:
  - Solve the subproblems
  - Combine solutions to subproblems
- Iteratively evaluates value functions and improving policy following Bellman equations.
- Assumes full knowledge of the MDP
- Bellman equations give recursive decomposition

DP: Iterative Policy Evaluation

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup
- $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_\pi$
- Using *synchronous* backups,
  - At each iteration $k + 1$
  - For all states $s \in \mathcal{S}$
  - Update $v_{k+1}(s)$ from $v_k(s')$
  - where s' is a successor state of s

DP: Policy Evaluation & Improvement

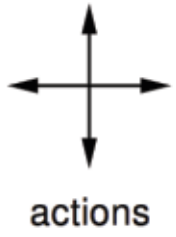
Iterative policy evaluation:

$$V_{k+1}(s) \doteq \mathbb{E}_\pi[R_{t+1} + \gamma V_k(S_{t+1}) | S_t = s]$$
$$= \sum_a \pi(a|s) \sum_{s',r} P(s', r|s, a)(r + \gamma V_k(s'))$$

Policy Improvement:

$$Q_\pi(s, a) \doteq \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a]$$
$$= \sum_{s',r} p(s', r|s, a) \Big[r + \gamma V_\pi(s') \Big]$$

DP in Small Gridworld



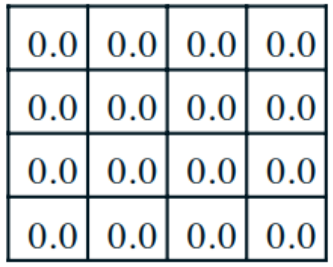
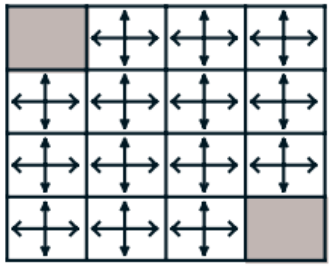
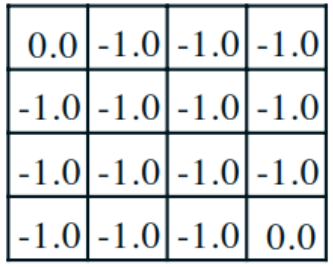
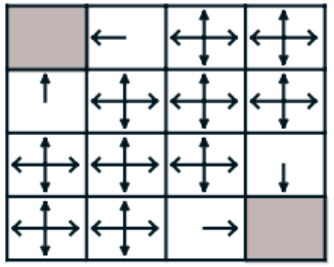
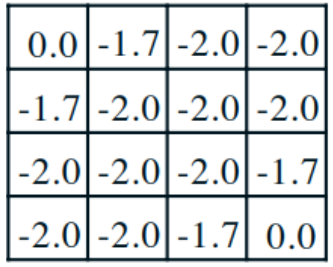
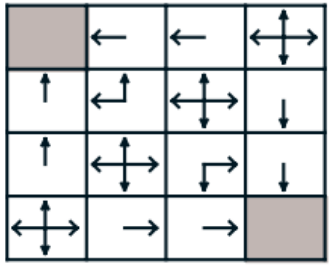
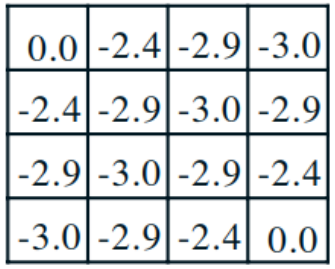
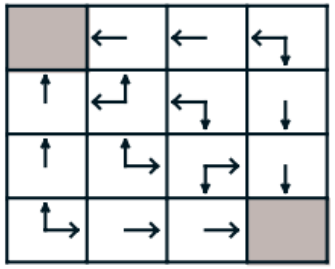
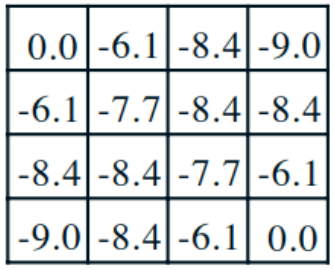
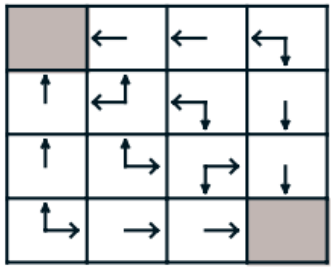
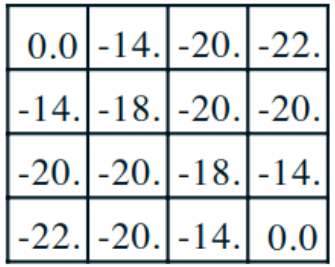
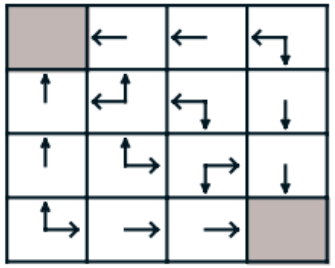
	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$r = -1$
on all transitions

- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states $1, \ldots, 14$
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

Convergence of iterative policy

	v_k for the random policy	greedy policy w.r.t. v_k
$k = 0$		
$k = 1$		
$k = 2$		
$k = 3$		
$k = 10$		
$k = \infty$		

optimal policy

How to Improve a Policy

- Given a policy π
  - Evaluate the policy π

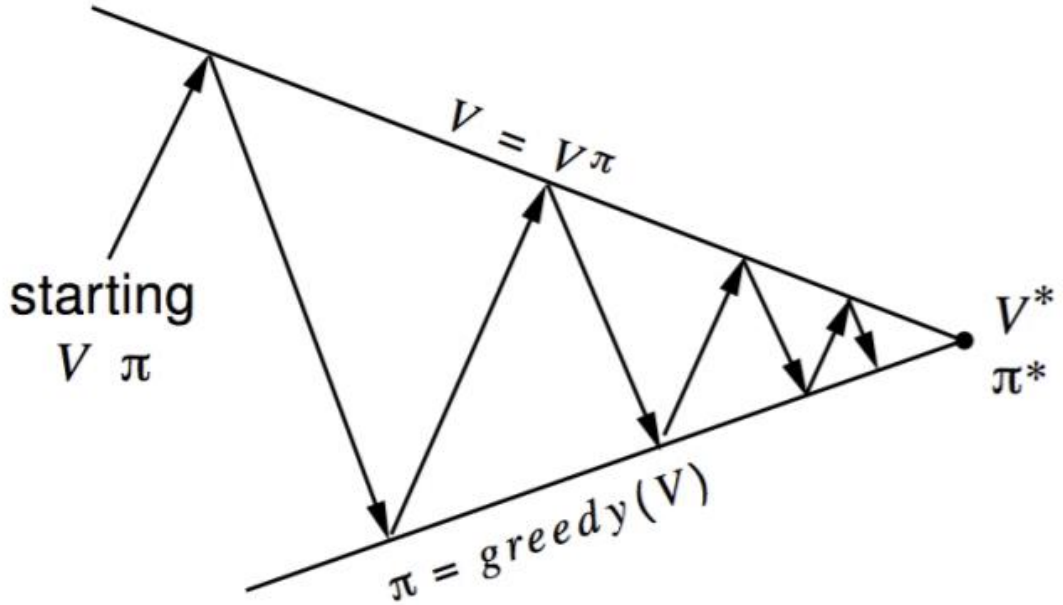
$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + ... | S_t = s\right]$$

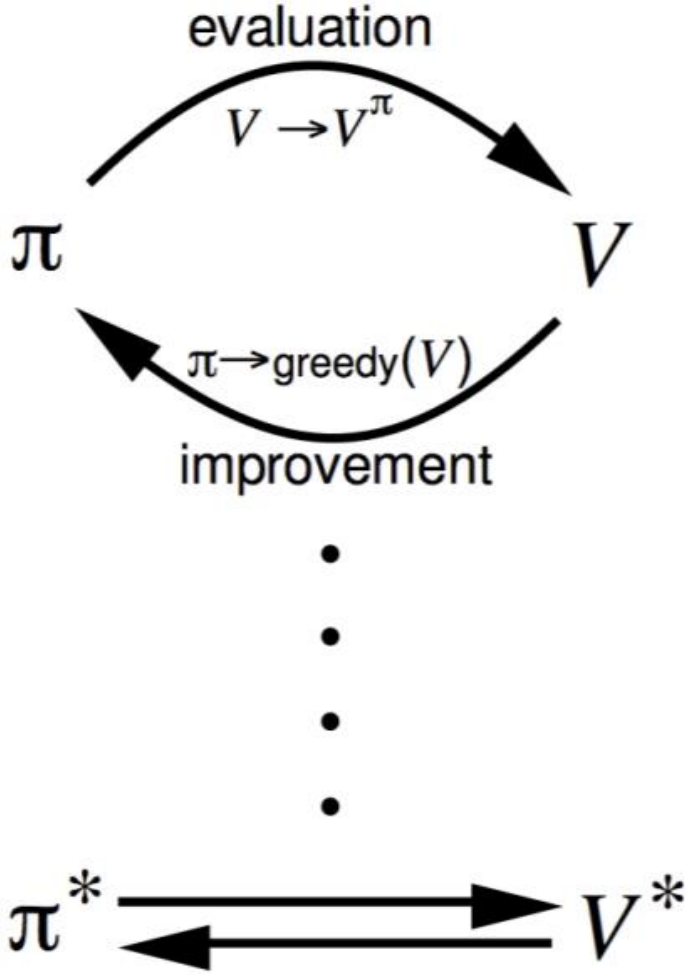
  - Improve the policy by acting greedily with respect to v_π

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

- In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
- In general, need more iterations of improvement / evaluation
- But this process of policy iteration always converges to $\pi*$

Policy Iteration





Policy evaluation Estimate v_π

Iterative policy evaluation

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

Greedy policy improvement

Policy Improvement

- Consider a deterministic policy, $a = \pi(s)$
- We can *improve* the policy by acting greedily

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

- This improves the value from any state s over one step,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- It therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] = v_{\pi'}(s) \end{aligned}$$

Policy Improvement

- If improvements stop,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

- Then the Bellman optimality equation has been satisfied

$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a)$$

- Therefore $v_\pi(s) = v_*(s)$ for all $s \in \mathcal{S}$
- so π is an optimal policy

Model-Free Predictions

- Monte-Carlo Learning
- Temporal-Difference Learning
- SARSA
- Q-learning
- n-step Bootstrapping

Monte-Carlo Method

- Monte-Carlo (MC) learns from episodes of experience
- MC is model-free: no knowledge of MDP transition/rewards
- MC learns from complete episodes S1,A1,R2,…,ST (all the episodes must eventually terminate): no bootstrapping
- MC uses simple idea: value = mean return

Monte-Carlo Method

- 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

Monte-Carlo Method

Empirical mean return for state *s*:

$$V(s) = \frac{\sum_{t=1}^{T} 1[S_t = s] G_t}{\sum_{t=1}^{T} 1[S_t = s]}$$

Action-value function:

$$Q(s,a) = \frac{\sum_{t=1}^{T} 1[S_t = s, A_t = a] G_t}{\sum_{t=1}^{T} 1[S_t = s, A_t = a]}$$

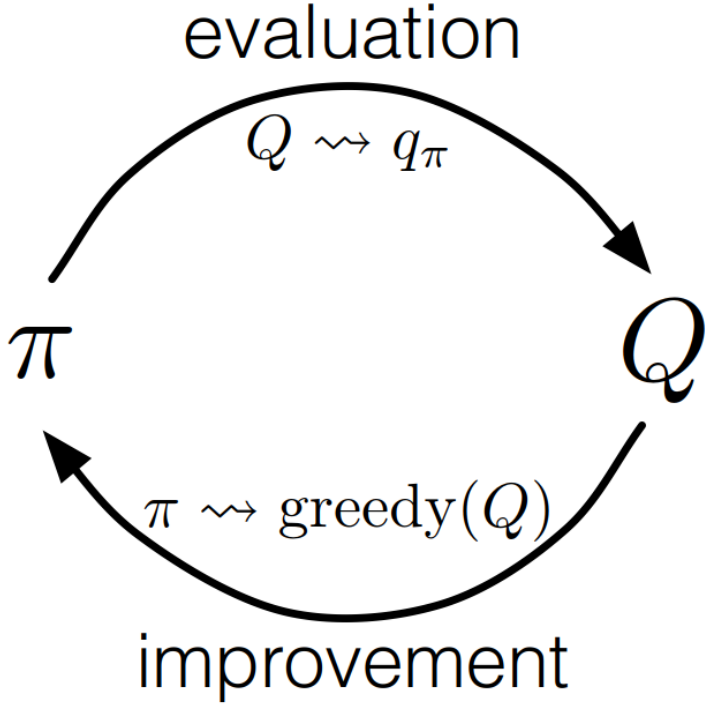
Learning the optimal policy by MC

1. Improve the policy greedily with respect to the current value function:

$$\pi'(s) \arg\max_{a \in \mathcal{A}} Q(s,a)$$

2. Generate a new episode with the new policy π (using ε-greedy algorithm)
3. Estimate Q using the new episode:

$$q_\pi(s,a) = \frac{\sum_{t=1}^{T} \left(1[S_t=s, A_t=a] \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \right)}{\sum_{t=1}^{T} 1[S_t=s, A_t=a]}$$



Temporal-Difference(TD) Learning

- Temporal-Difference (TD) Learning is model-free and learns from episodes of experience.
- TD is *model-free*: no knowledge of MDP transition/rewards
- TD learning can learn from **incomplete** episodes, by bootstrapping.
- Updates targets with regard to existing estimates rather than exclusively relying on actual rewards and complete returns

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*

TD Algorithm

Tabular TD(0) for estimating v_π

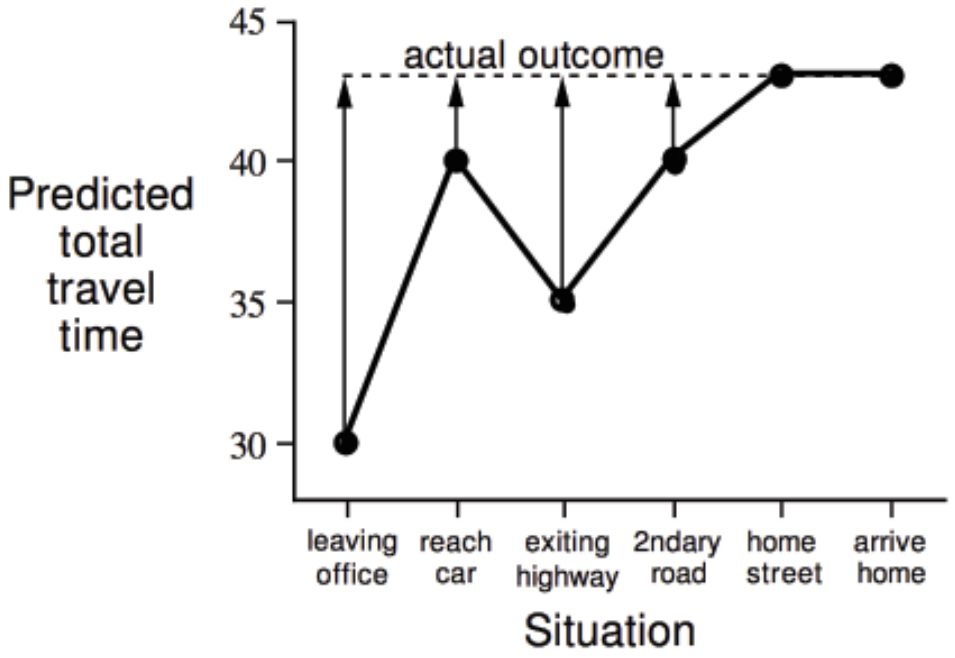
Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

Example: Driving Home

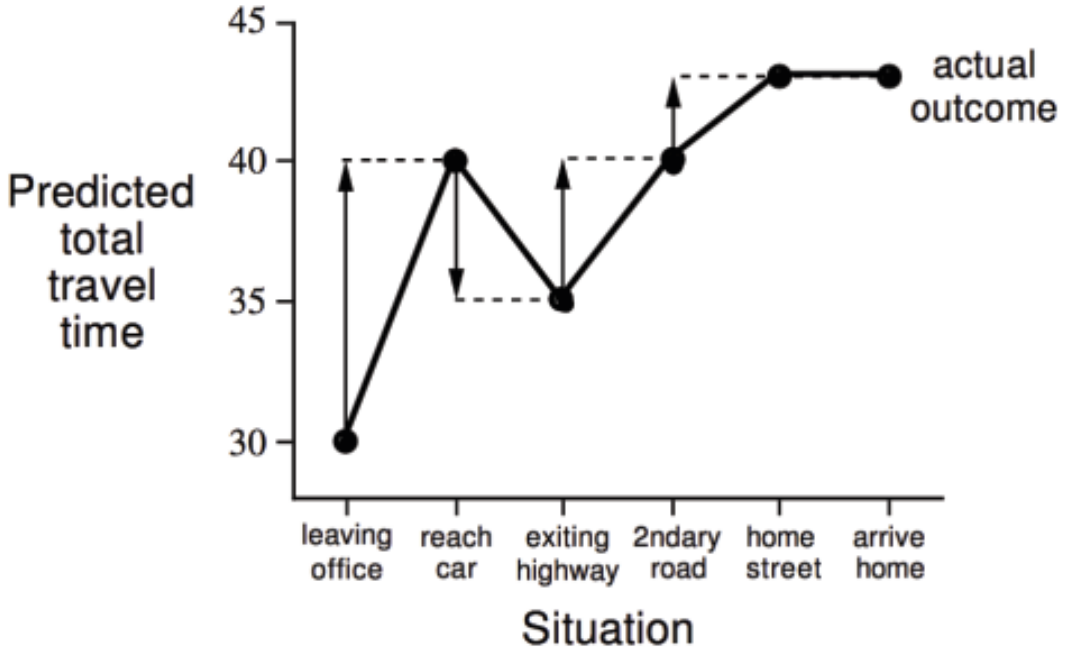
State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

Example: Driving Home

Changes recommended by Monte Carlo methods (α=1)



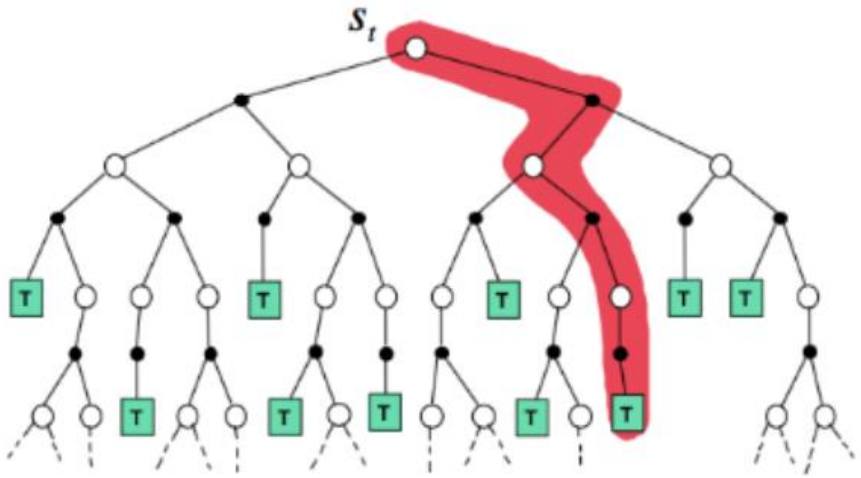
Changes recommended by TD methods (α=1)



Comparison of state value functions

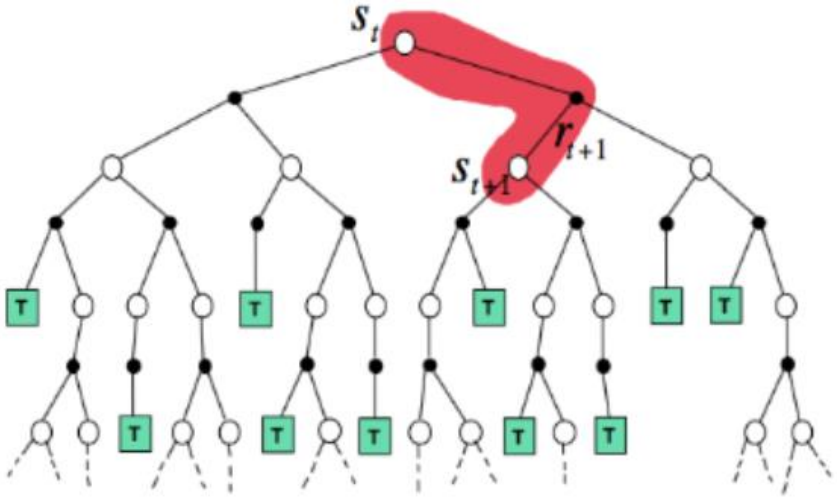
Monte-Carlo

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



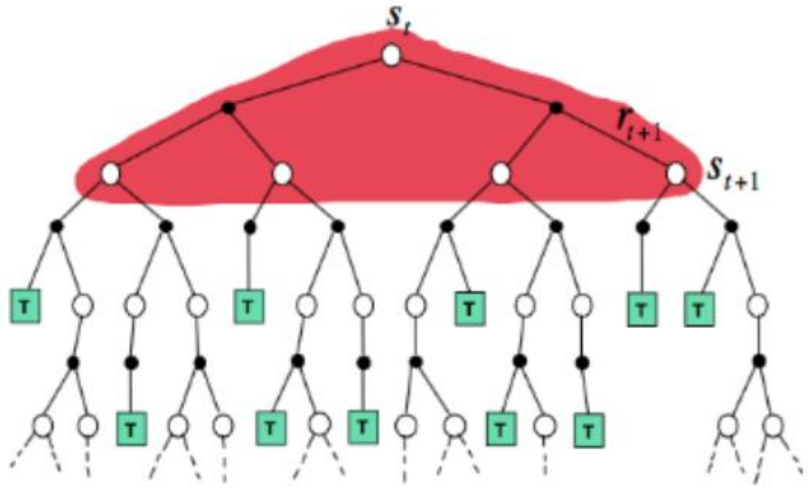
Temporal-Difference

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



Dynamic Programming

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



SARSA: On-Policy TD control

Updating Q-value by following a sequence of $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1} \ldots, S_T$

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

$A_t = \arg\max_{a \in \mathcal{A}} Q(S_t, a)$

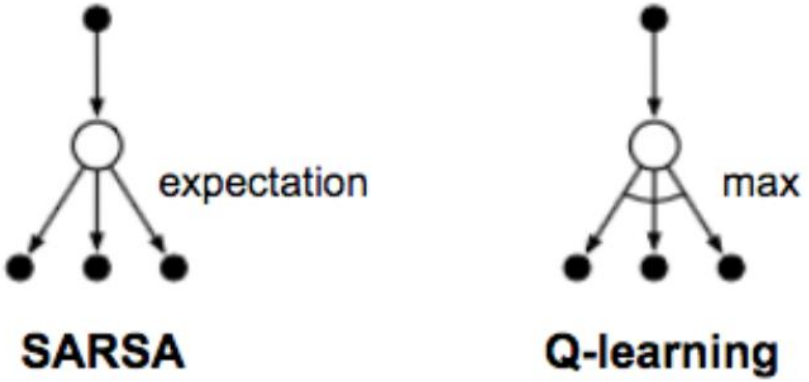
Expected SARSA:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\left[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1},A_{t+1}) \mid S_{t+1}] - Q(S_t,A_t)\right]$$
$$\leftarrow Q(S_t,A_t) + \alpha\left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1},a) - Q(S_t,A_t)\right],$$

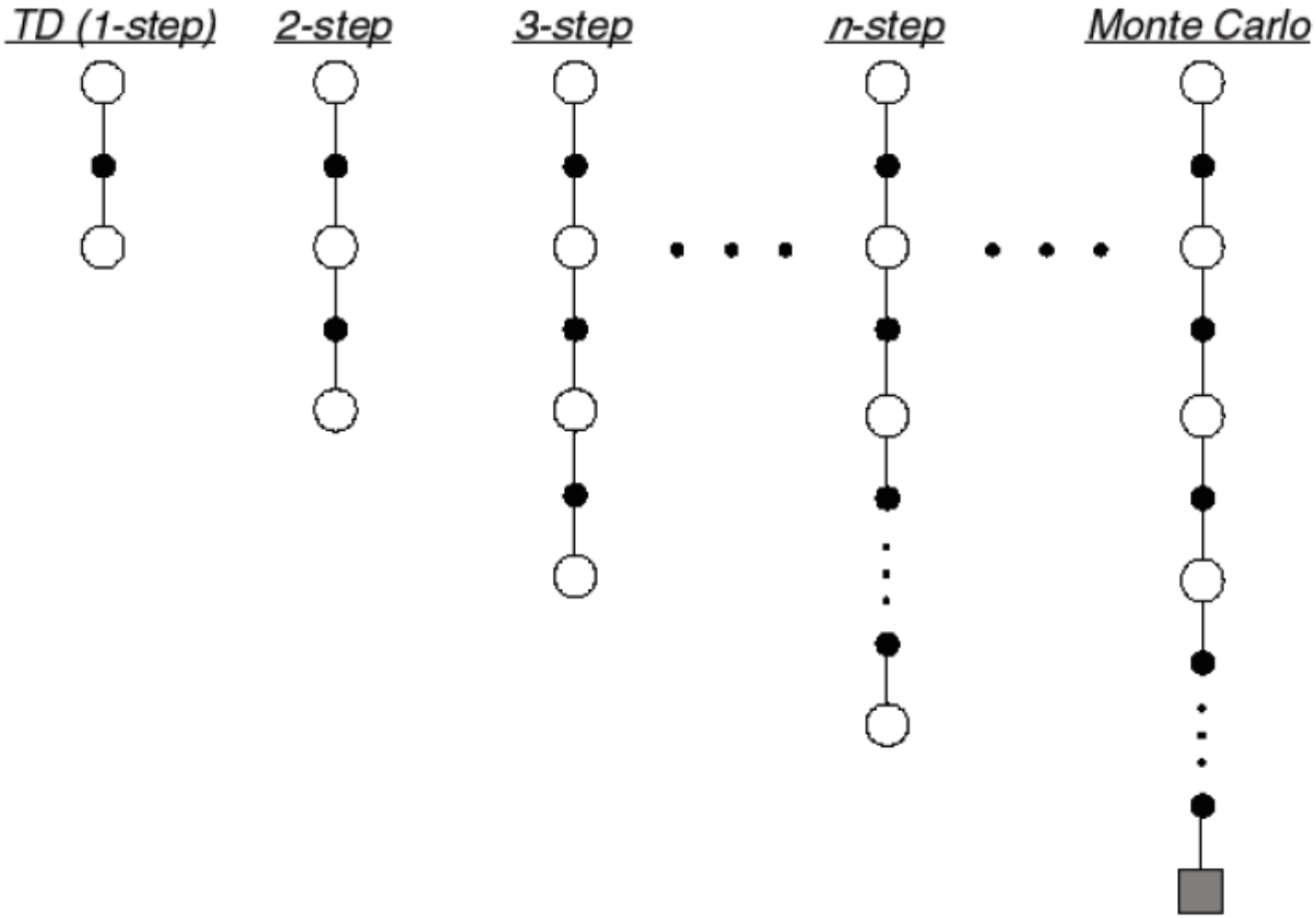
Q-Learning: Off-policy TD control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy) $A_t = \arg\max_{a \in \mathcal{A}} Q(S_t, a)$
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$
  until S is terminal



n-Step Algorithm



n-Step Return

- Consider the following n-step returns for $n = 1, 2, \infty$:

$$
\begin{array}{lll}
n = 1 & (TD) & G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \\
n = 2 & & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\
\vdots & & \vdots \\
n = \infty & (MC) & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T
\end{array}
$$

- Define the n-step return

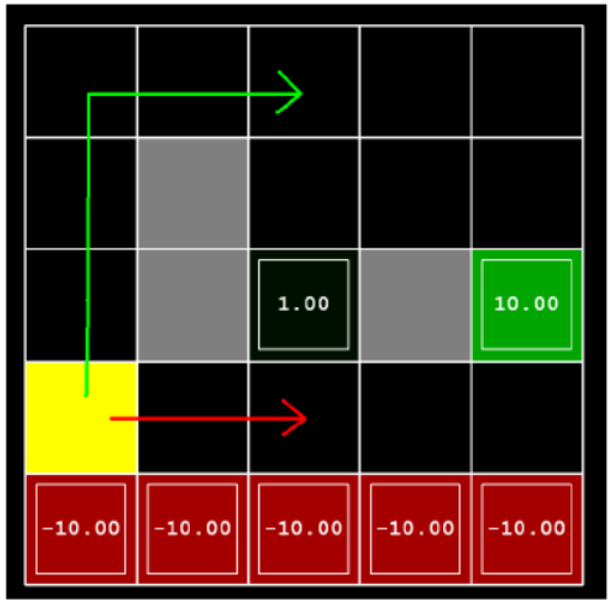
$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- n-step temporal-difference learning

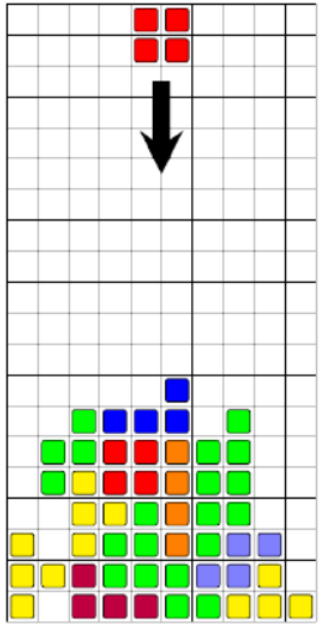
$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{(n)} - V(S_t) \right)$$

Can tabular methods scale?

We need a huge amount of storage for tabular methods. For instance, for discrete environments such as Atari breakout, this could be 10^{308}



Gridworld
10^1



Tetris
10^60

Atari
10^308 (ram) 10^16992 (pixels)

References

- David Silver’s course slides “Reinforcement Learning”
- Sutton, Richard S., and Andrew G. Barto. "Reinforcement Learning: An introduction (2 edition)" (2018)