TEMPORAL-DIFFERENCE LEARNING

SAN DIEGO MACHINE LEARNING

JUNE 19, 2021

HOW TO PARTICIPATE

- One discussion leader, and everyone welcome to participate
- Majority of material comes from Reinforcement Learning by Sutton and Barto
- Options to approach the content:
  - Treat this as a standalone webinar
  - Read the book first, and come with questions and discussion items
  - Use this meetup as a primer and read the chapters afterward
- Ask questions
- Give feedback. Too fast or too slow? Want to see more of something or less of something else?
- Have fun!

AGENDA

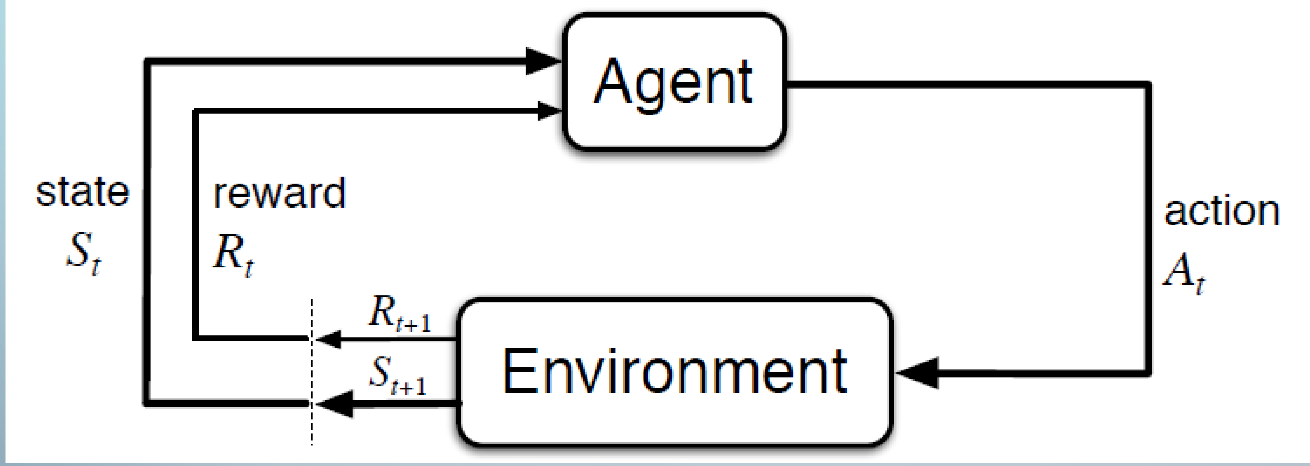
- Recap what reinforcement learning (RL) is
  - Elements and formulation as Markov decision processes (MDP)
  - Terminology and notation used in RL
  - The Bellman equations
  - Generalized policy iteration
- Introduce temporal-difference (TD) learning
  - Temporal-difference update
  - TD(0)
  - SARSA
  - Q-learning

REINFORCEMENT LEARNING

- Reinforcement learning (RL) is about an *agent* learning from interacting with its uncertain *environment*
  - The agent interacts by choosing from a set of allowed *actions*
  - It gets feedback from a numeric *reward* signal
  - Goal is to maximize the *return*, which is the total rewards received
- Reinforcement learning is about exploring the environment and recording useful information for the future
- RL is sequential decision making; time is intrinsic

MARKOV DECISION PROCESSES

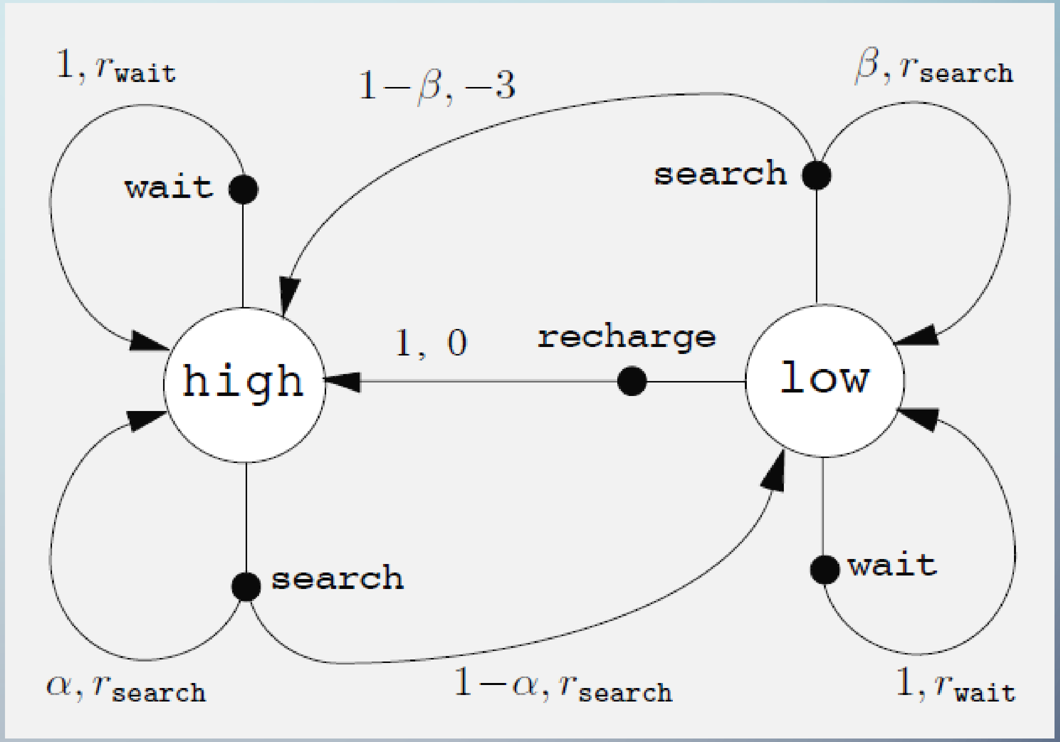
- Elements of the fully observable Markov Decision Process (MDP):
  - State - at each time step t, the environment is in some state S_t
  - Action - at each time step t, the agent chooses an action A_t
  - Reward - after taking the action, the agent is given a reward signal R_{t+1} and subsequently finds itself in a new state S_{t+1}



- In a *Markov* Decision Process, the transition at any given time t only depends on the state S_t and action chosen A_t

MDPS AS A GRAPH

- Sometimes it is easier to visualize a MDP as a directed graph
  - The states are nodes (big white circles)
  - The actions are edges leading from nodes (here with small black circles)
  - The rewards are values along directed edges that take you to a new state
- Here is the recycling robot from the book:



REINFORCEMENT LEARNING NOTATION

Letter	Used for
s	**S**tate
a	**A**ction
r	**R**eward
γ	Discount rate
G	Return – sum of all future rewards
p	Transition **p**robability
v	**V**alue function for states
q	Value function for state-action pairs
π	Policy (**π**ολιτική)
*	Optimal choices, e.g. π_*

BELLMAN EQUATION

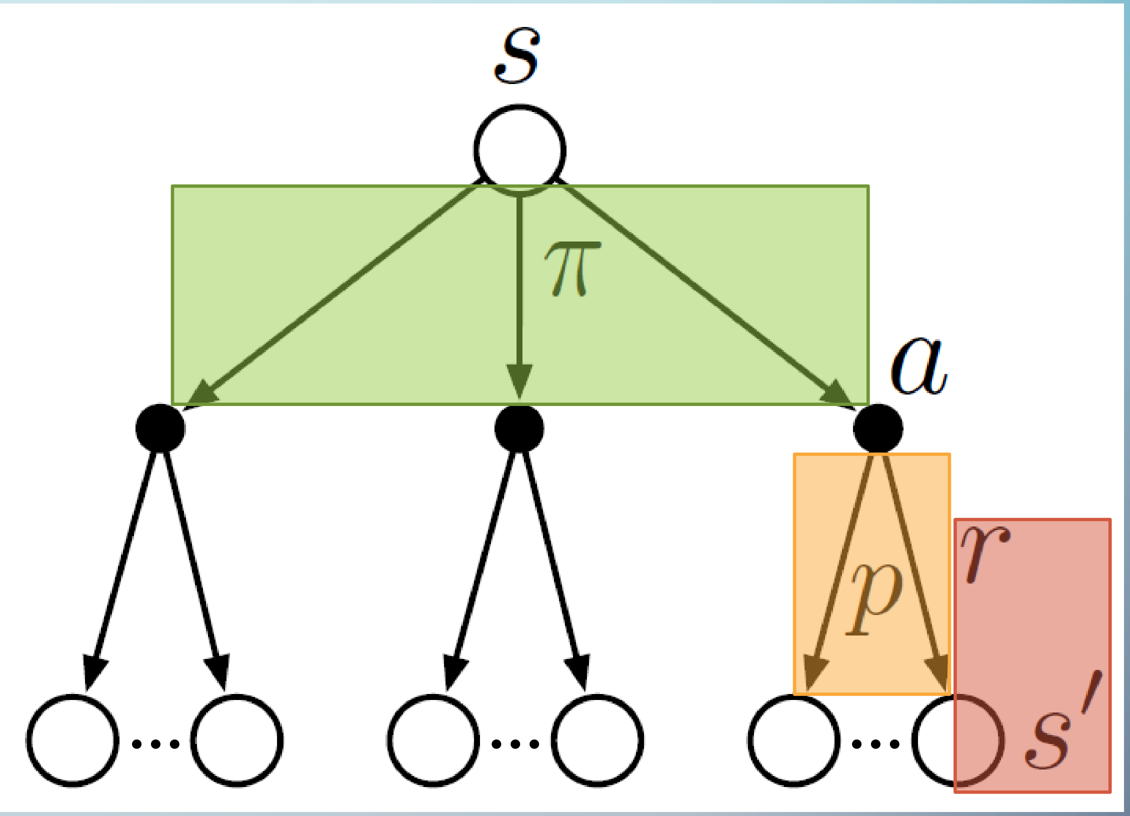
- The value function for state s under policy π is a sum of the rewards received and the value functions for each future state s' times the probability of winding up there
- Formally:

$$v_{\pi}(s) = \sum_{a} \underbrace{\pi(a,s)}_{\substack{\text{Probability you} \\ \text{take action } a}} \sum_{s',r} \underbrace{p(s',r|s,a)}_{\substack{\text{Probability you} \\ \text{get reward } r \\ \text{and end in state } s'}} \underbrace{[r + \gamma v_{\pi}(s')]}_{\substack{\text{Reward plus} \\ \text{discounted value} \\ \text{of new state } s'}}$$

BELLMAN EQUATION VISUALIZED

This is a *backup diagram* for $v_\pi(s)$. To compute it:

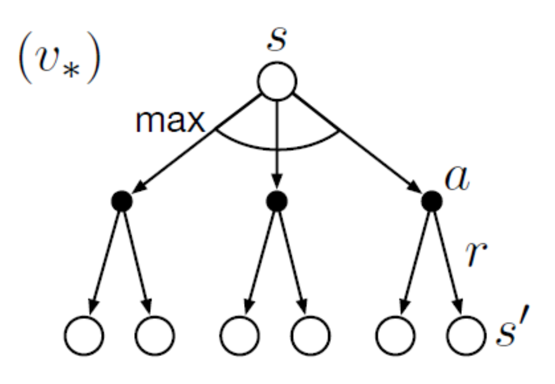
- We need to sum over each branch of π(), based on the probability of each action *a*
- And sum over of each branch of *p*(), based on probability we wind up in state *s’*
- The quantity we sum is the reward and the discounted value of possible state *s’*



$$v_\pi(s) = \sum_a \pi(a,s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right]$$

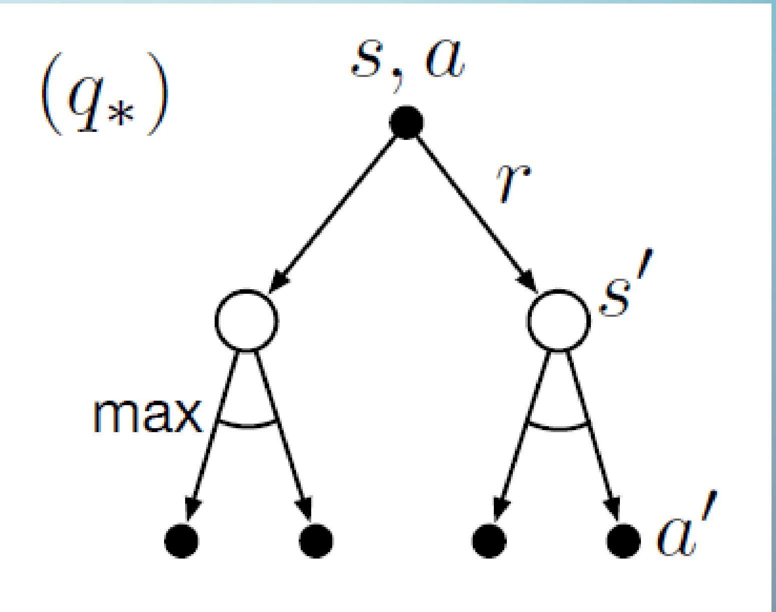
BELLMAN OPTIMALITY EQUATIONS – V()

- The *Bellman optimality equation* says the optimal value for a state must be the same as the return from the best action
- We can rewrite it recursively



$$
\begin{aligned}
v_*(s) &= \max_{a\in\mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] && \text{(by (3.9))} \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] && (3.18) \\
&= \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_*(s')\big]. && (3.19)
\end{aligned}
$$

BELLMAN OPTIMALITY EQUATIONS – Q()



- The *Bellman optimality equation* for state-action pairs is very similar.
- The optimal value for a state-action pair must be the same as the return from the reward and best next action
- It also can be written recursively

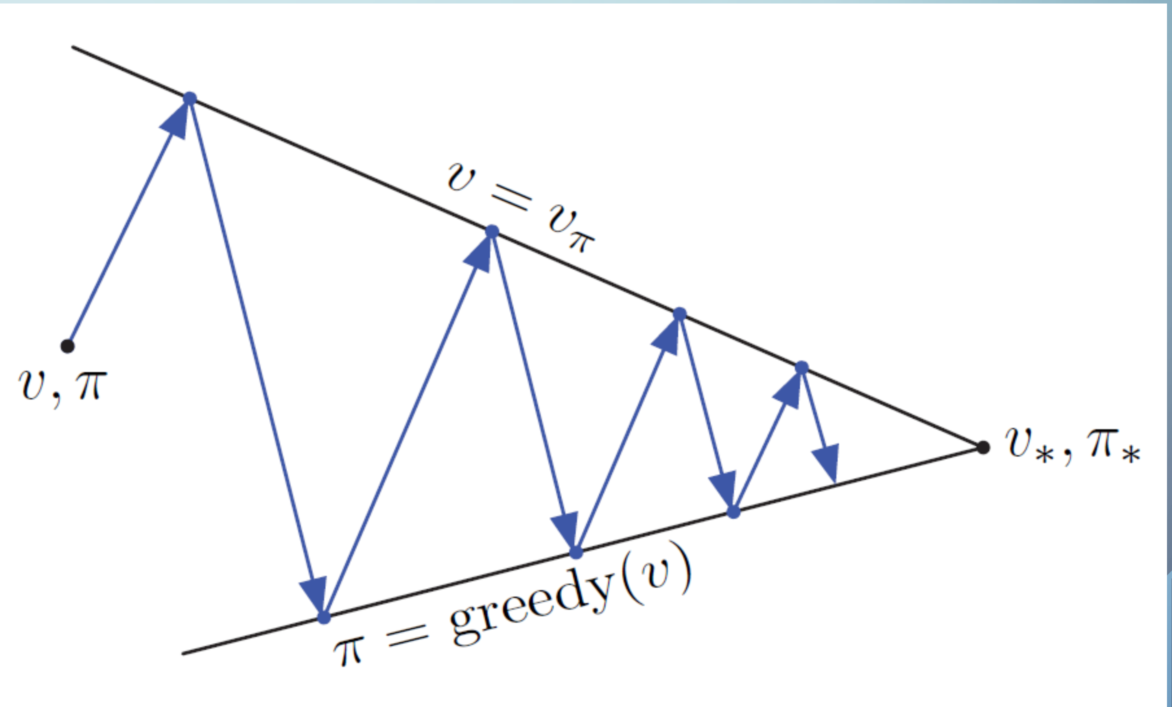
$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big]. \end{aligned} \tag{3.20}$$

POLICY ITERATION

- The book shows a sequence like this:

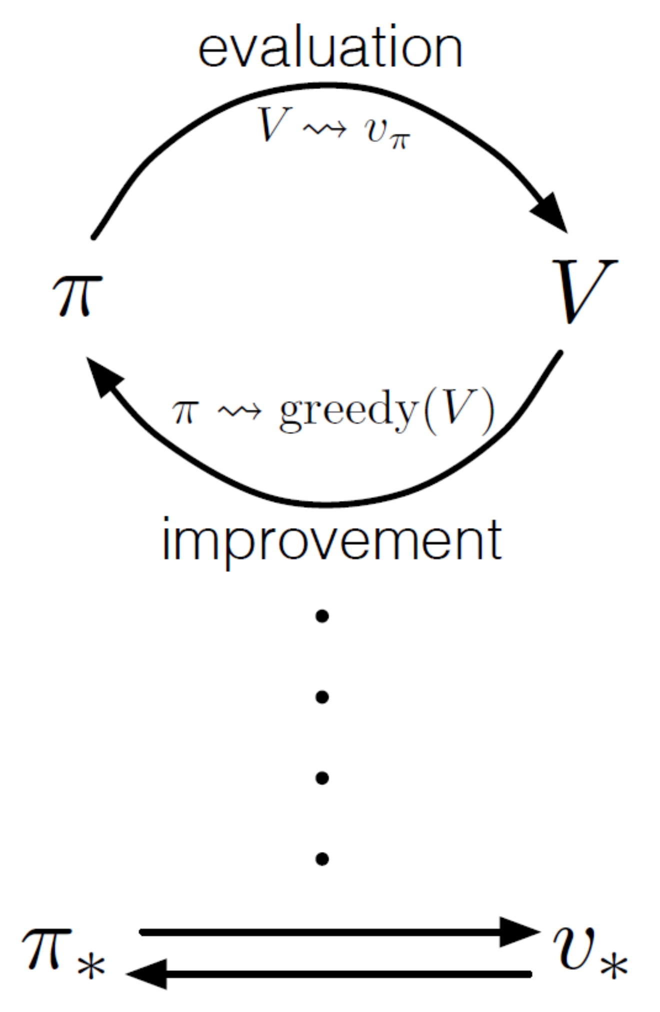
$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_1 \xrightarrow{E} \ \dots \ \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

- The arrows with E's are full cycles of iterative policy evaluation
- And the arrows with I's are policy improvement



GENERALIZED POLICY ITERATION

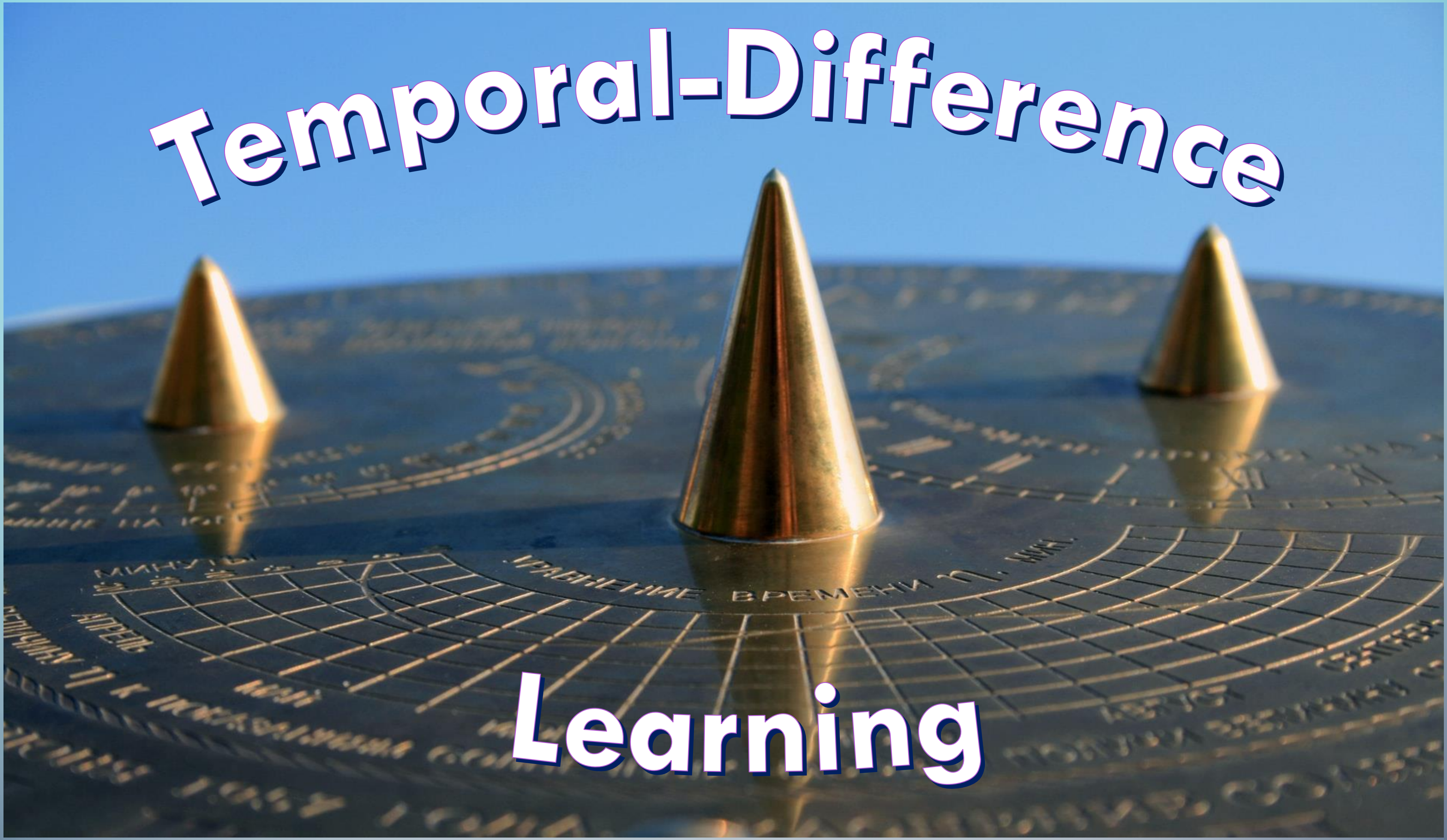
- The term *generalized policy iteration* (GPI) refers to the general idea of letting policy evaluation and policy improvement processes interact
  - Doesn’t matter how fully each evaluation or improvement step runs, or if they exactly alternate



REINFORCEMENT LEARNING CONTROL

- With this foundation, there’s a lot we can tackle
  - Algorithms for learning
  - Dealing with memory and compute limitations
  - Getting models to converge quickly
- We also still have many challenges
  - Reward design – effectively communicating the real goal
  - Sparse rewards
  - Credit assignment – which actions in trajectory contributed
  - Exploration vs. exploitation

Temporal-Difference Learning



TEMPORAL-DIFFERENCE PREDICTION

- The dynamic programming (DP) update rule says to calculate a new estimate for $v_\pi(s)$:

$$v_{k+1}(s) = \sum_{a} \pi(a, s) \sum_{s', r} p(s', r|s, a)[r + \gamma v_k(s')]$$

  This update goes really wide, but only one step ahead

- Monte Carlo (MC) methods average sampled returns.
  If we tweak Monte Carlo to use a constant step size α, update is:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

  This update is narrow, sampling only one trajectory, and
  it waits all the way until the end of the episode to do this update

TEMPORAL-DIFFERENCE PREDICTION [2]

- Another option is to stay narrow, like Monte Carlo, and only go one step ahead before updating, like dynamic prog.
- This is the way temporal-difference (TD) does it's update
- Again, the Monte Carlo update after full episode:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

- Our estimate of G_t is $R_{t+1} + \gamma V(S_{t+1})$
- Here is the temporal-difference update after one step:

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

TEMPORAL-DIFFERENCE PREDICTION [3]

- Temporal-difference update:

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

- TD, like MC, doesn't require knowledge of environment dynamics
- TD, like MC, samples from experience
- TD differs from MC (but is like DP) b/c it updates after every step
- TD also differs from MC (but is like DP) in that it bootstraps
  - TD estimates $V(S_t)$ using $V(S_{t+1})$

TD(0)

- Using this update, we can create a simple algorithm to estimate v_π for any policy π. This algorithm is called TD(0).

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\left[R + \gamma V(S') - V(S)\right]$
        $S \leftarrow S'$
    until S is terminal

TD ADVANTAGES

- Advantage over DP because you don't need to know environment dynamics
- Advantages over MC because:
  - TD is naturally online and fully incremental
  - Waiting to end of episode is sometimes too long
  - Importance sampling doesn't work in practice (esp. long episodes)
  - TD converges faster than MC
  - Example 6.4 in the book shows a subtle difference where MC converges to a V() that minimizes mean-squared error for the rewards given. However TD minimizes error for a MDP model of the sample rewards.

TD CONTROL

- Can we use the TD(0) algorithm shown for estimating $v_\pi(s)$ and plug it into policy iteration to find the optimal policy v_*?
- If we don't know the environment dynamics (the $p()$ function), then the answer is no
  - Policy improvement greedily chooses actions for each state, but if we don't know the rewards and next states, we can't do improvement
- We showed TD for $v_\pi(s)$ because it's simplest, but for real problems we are going to use TD for $q_\pi(s, a)$. Here's the update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

SARSA

- We can now look at the SARSA algorithm for on-policy TD control
  - Notice there is no policy improvement step

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

Q-LEARNING

- Now comes the Q-learning algorithm for off-policy TD control
  - Huge breakthrough by Watkins in 1989
  - Rather than learning q_π, it directly goes after q_*, independent of the behavior policy being followed
  - Does not need importance sampling
  - Note that we can derive a behavior policy from Q that always explores, and don't have to explicitly store a separate behavior policy
- Here's the Q-learning update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

Q-LEARNING [2]

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

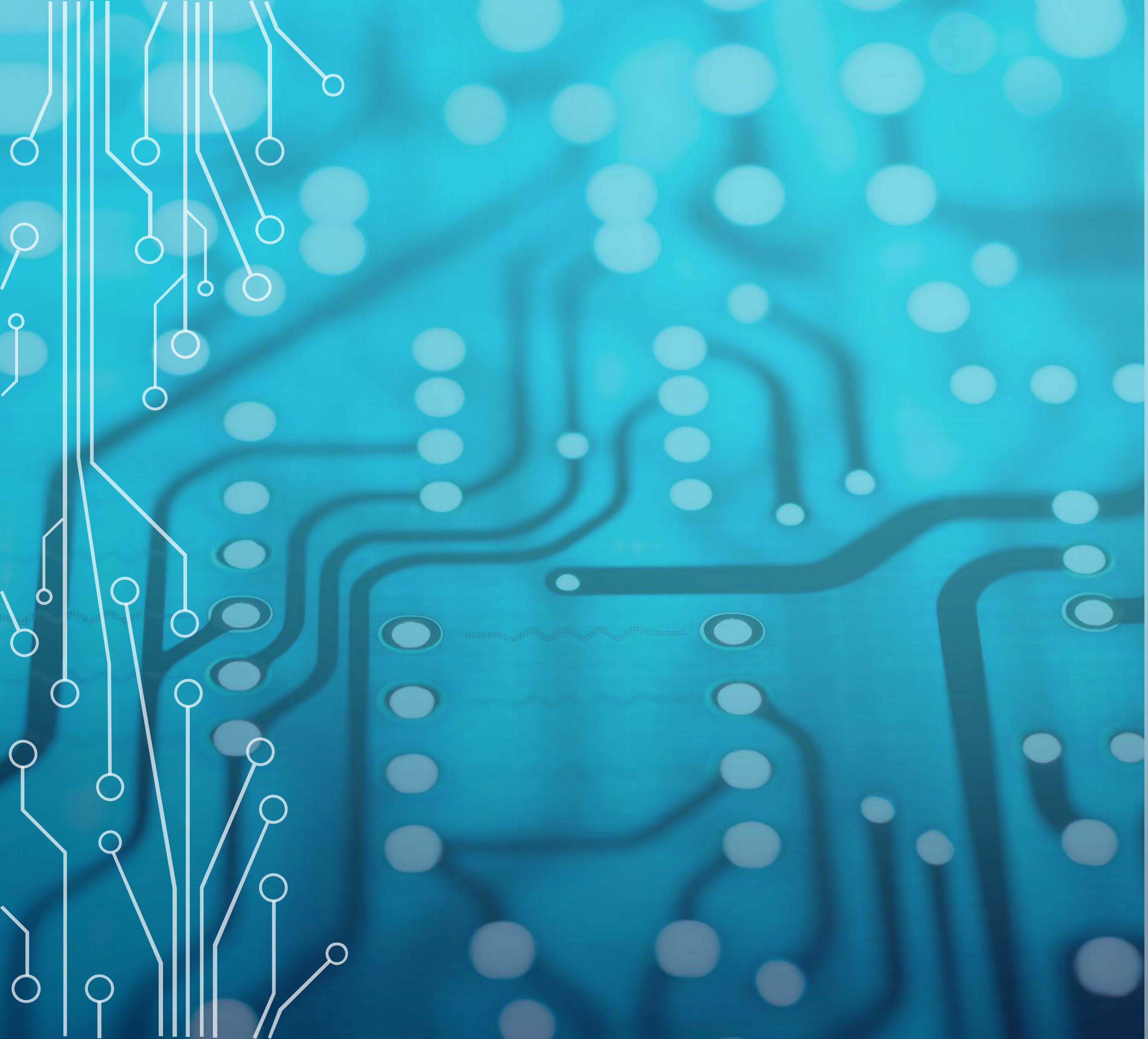
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

ADDITIONAL SARSA/TD ALGORITHMS

- Expected SARSA
  - SARSA adds the reward to a single action's discounted future reward estimated by Q(). Expected SARSA adds the reward to the expected value from all the one-step discounted future rewards coming from Q().
- Double Q-learning
  - Maximization bias is a problem with RL algorithms
    - Imagine you had 1,000 actions all mean zero variance 1, what would max() be?
  - Double Q-learning keeps two separate Q tables. You pick a "best" action using the argmax() from one, but you use the Q() value from the other
- Afterstates
  - If you know 100% the single state after an action, can improve algorithms to use the *afterstate* instead of the current state

RECAP

- Review what reinforcement learning (RL) is
  - Elements and formulation as Markov decision processes (MDP)
  - Terminology and notation used in RL
  - The Bellman equations
  - Generalized policy iteration
- Introduce temporal-difference (TD) learning
  - Temporal-difference update
  - TD(0)
  - SARSA
  - Q-learning



QUESTIONS

&

DISCUSSION

NEXT SESSION

- Next session we will play with code examples finding optimal policies using Monte Carlo, SARSA, and Q-learning,
  on Sat. June 26
- We may also discuss some additional material about temporal-difference learning, from chapter 7 of Sutton & Barto
- The following week, Sat. July 3, there will be no book club