# INTRODUCTION TO REINFORCEMENT LEARNING

SAN DIEGO MACHINE LEARNING

MAY 15, 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

- Explain what reinforcement learning (RL) is and how it differs from supervised and unsupervised learning
- Describe the basic elements of RL
- Define RL in terms of Markov decision processes
- Introduce the terminology and notation used in RL
- Show the Bellman equations and simple ways to read them
- Show how optimal solutions can be found for very simple RL problems
- Share some of the challenges with general RL problems
- Give a taste for what more advanced RL algorithms do

# KINDS OF MACHINE 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 total rewards received
- Supervised learning is learning from ground truth labels
  - Many problems don't have sufficient representative examples, e.g. backgammon, or experts don't even know the best actions
- Unsupervised learning might seem similar, also no labels
  - But unsupervised learning doesn't have an environment or a reward signal

# EXAMPLES OF REINFORCEMENT LEARNING

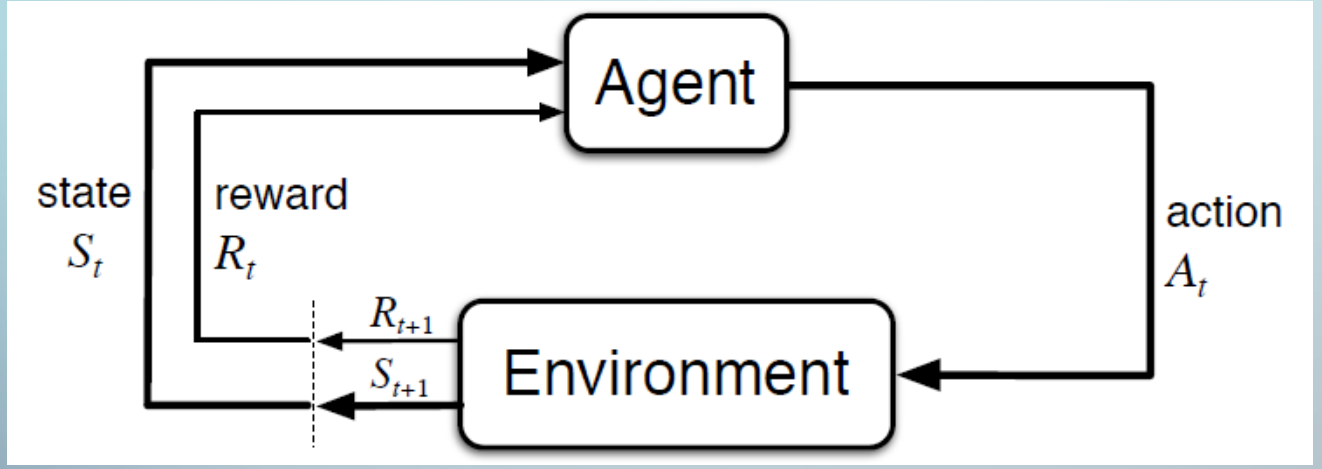
- Many kinds of problems can easily be posed as a RL problem, with an agent that sequentially chooses actions in an environment that represents everything not under the direct control of the agent
  - Chess or Atari games
  - Control a power station
  - Manage an investment portfolio
  - Navigate a maze
  - Make a robot walk
- What other examples can you come up with?
  - Note that the agent boundary can be set in different places, and sometimes smaller RL  problems are nested inside others

# REINFORCEMENT LEARNING

- Reinforcement learning is about exploring the environment and recording useful information for the future
- RL is sequential decision making; time is intrinsic
- Situations that look similar might differ due to history
- Good or bad consequences might lie far into the future
- The environment might be stationary (unchanging) or dynamic
  - In dynamic environments, more explore/exploit tradeoff concerns
- We want a way to model this potentially very complex interaction of the agent with the environment over time

# 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 $S_{t+1}$



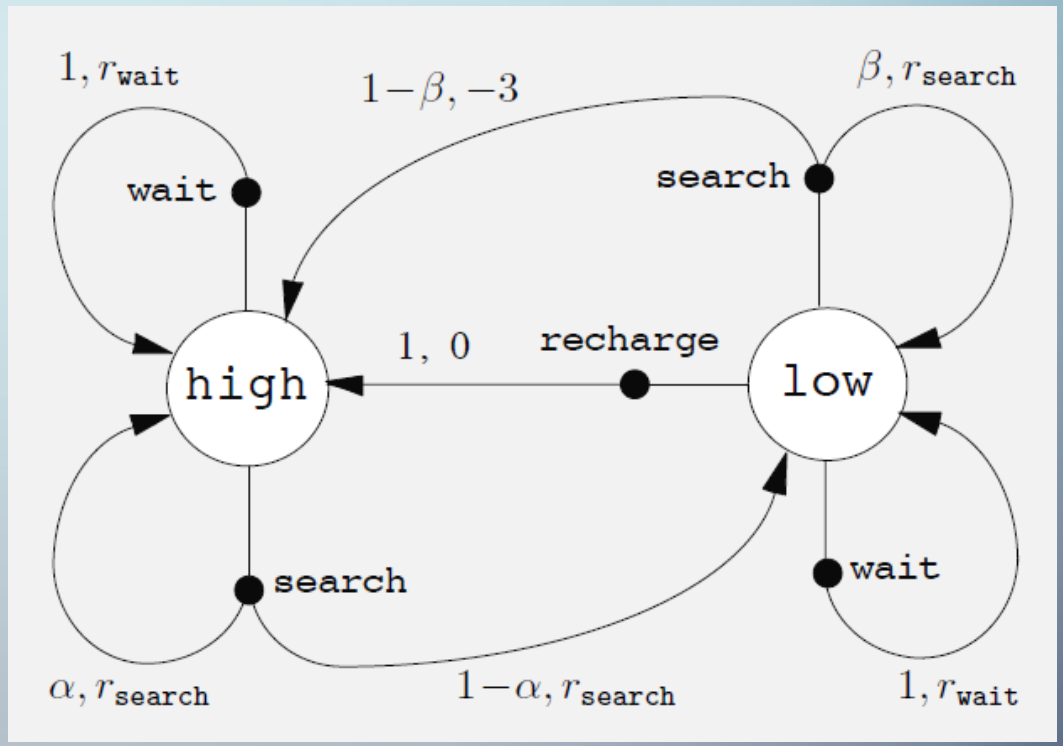

- Repeating the above process creates a trajectory like this:
  $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$

# MARKOV DECISION PROCESSES

- In a *Markov* Decision Process, the transition at any given time $t$ only depends on the state $S_t$ and action chosen $A_t$
  - The transition must not depend on any history prior to $S_t$ (memoryless)
  - Said another way, the state $S_t$ must include any information from the past needed to understand the future
    - E.g. if you're playing Pac-Man, even if you, the ghosts, and everything else on the screen is identical, the state is not the same if you have one life left vs. three lives left. Therefore, the state at any time must include the number of lives.
- The transitions can be deterministic (e.g. chess) or stochastic (e.g. a slot machine)
  - We can model everything as stochastic, where deterministic transitions will have probability 1

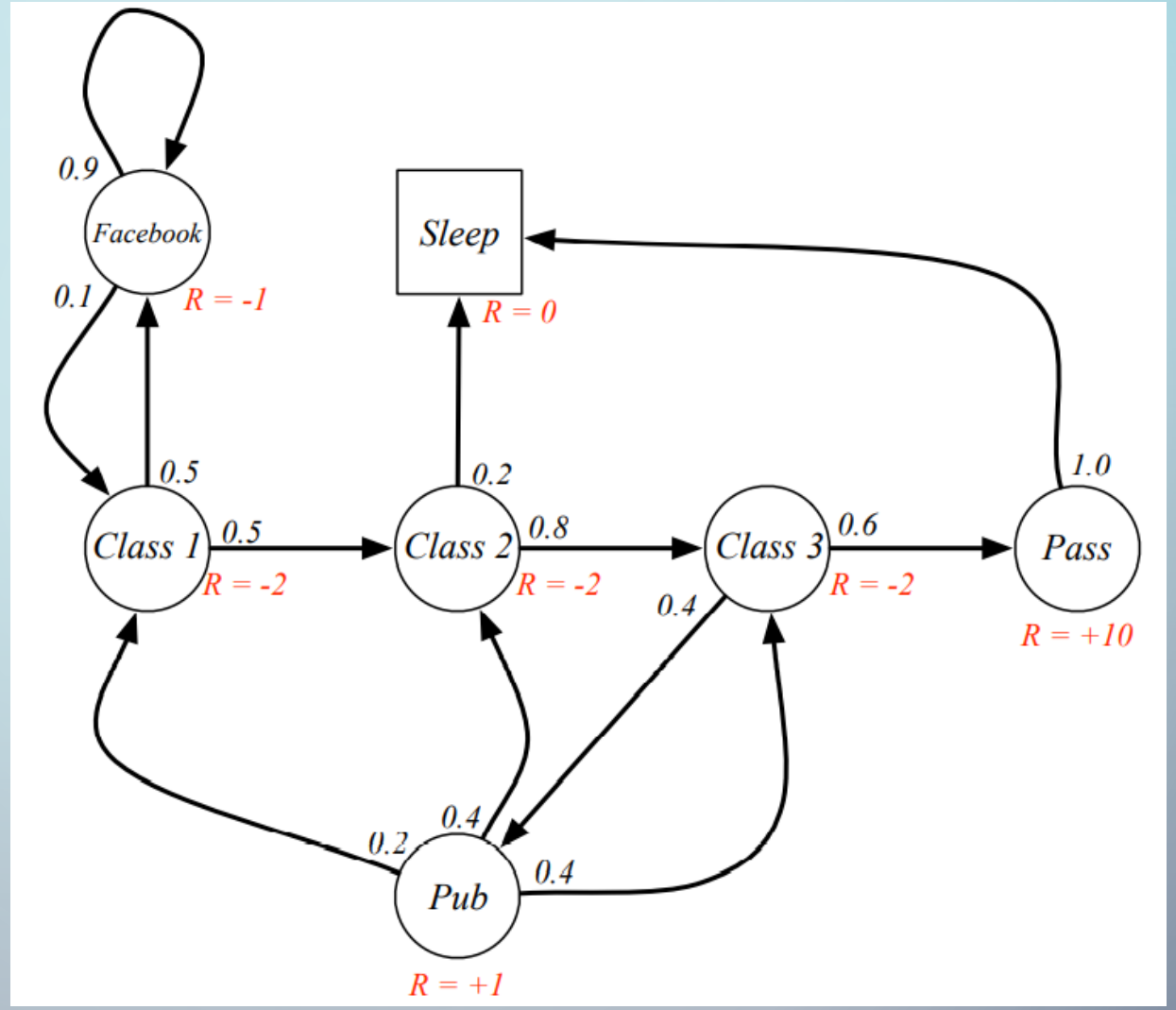
# MDPS AS A GRAPH

- 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:



# MDPS AS A GRAPH

- Another example:



# EXAMPLES OF MDPS

- From the book:
  - **Bioreactor –** Suppose reinforcement learning is being applied to determine moment-by-moment temperatures and stirring rates for a bioreactor (a large vat of nutrients and bacteria used to produce useful chemicals).
  - **Pick-and-Place Robot –** Consider using reinforcement learning to control the motion of a robot arm in a repetitive pick-and-place task.
- What other examples of tasks can you represent using the MDP framework?

# MDP NOTATION

- The reward and resultant state can be expressed as the output of functions taking only two parameters – current state and action
- In the Sutton & Barto book, the following notation is used: $p(s', r \mid s, a)$
  - The vertical bar is the symbol used for conditional probability
- This function $p()$ returns the probability (between 0 and 1)
  - Starting from state $s$
  - Taking action $a$
  - We wind up in state $s'$ and get reward $r$

# GOALS AND REWARDS

- The goal of reinforcement learning is to maximize total reward
- If our task naturally breaks into *episodes*, then we have a finite sum rewards, and the *return* $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$
- We can also *discount* future reward so they are worth less the farther out they are (a common practice in finance)
  - If we have a discount rate $\gamma$ (the Greek letter gamma) then:
    $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$

$$G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1}$$


It is useful to express a function recursively as a function of itself

- It can also be shown that $G_t = R_{t+1} + \gamma G_{t+1}$ (eq. 3.9 on p. 55)

# POLICIES AND VALUE FUNCTIONS

- A *policy* gives the probability for each action $a$ from state $s$ only
  - A policy is represented by π, the Greek letter pi
  - The notation for the probability of action $a$ in state $s$ is $\pi(a|s)$
- The *value function* of a state $s$ when following policy π is the expected return (not immediate reward) when starting from $s$ and choosing actions according to the policy π
  - This is denoted as $v_\pi(s)$
- Similarly, the value of taking action $a$ from state $s$ when following policy π is the expected return (not immediate reward) starting from $s$, taking action $a$, and thereafter following policy π
  - This is denoted as $q_\pi(s, a)$

# BELLMAN EQUATION

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

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

**Probability you take action *a***

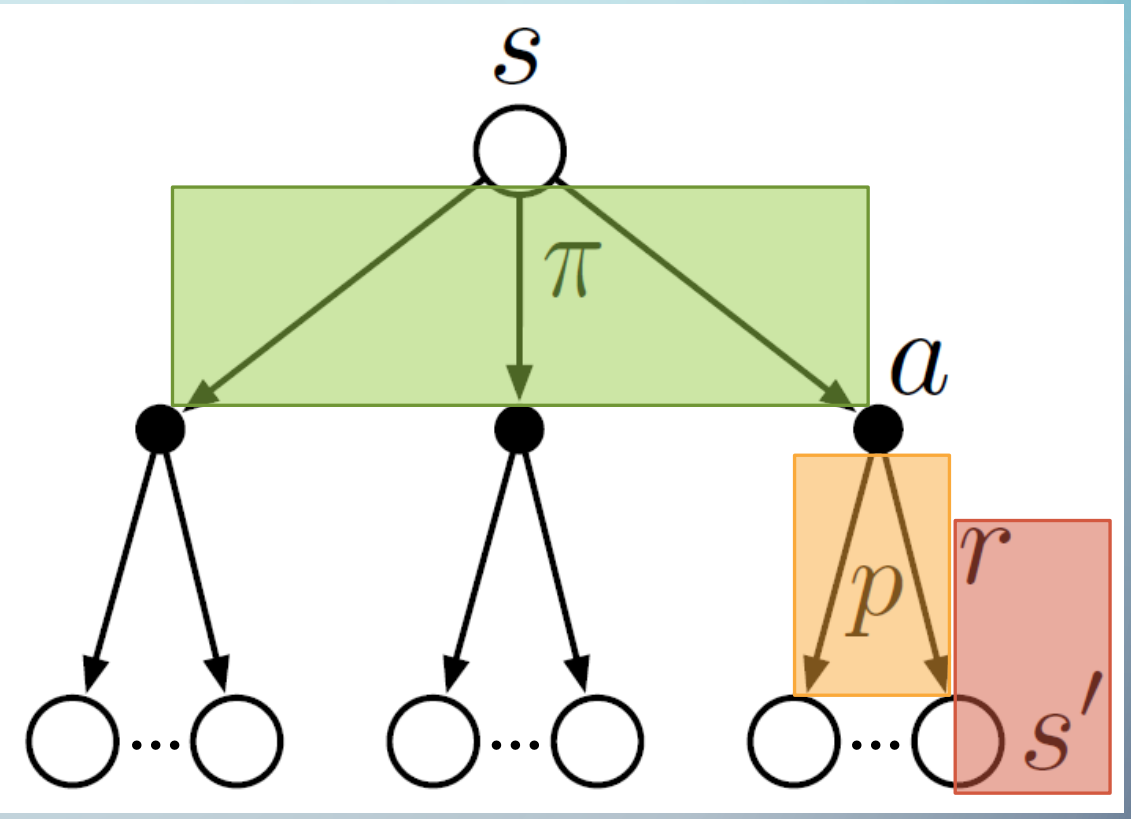
**Probability you get reward *r* and end in state *s'***

**Reward plus discounted value of new state *s'***

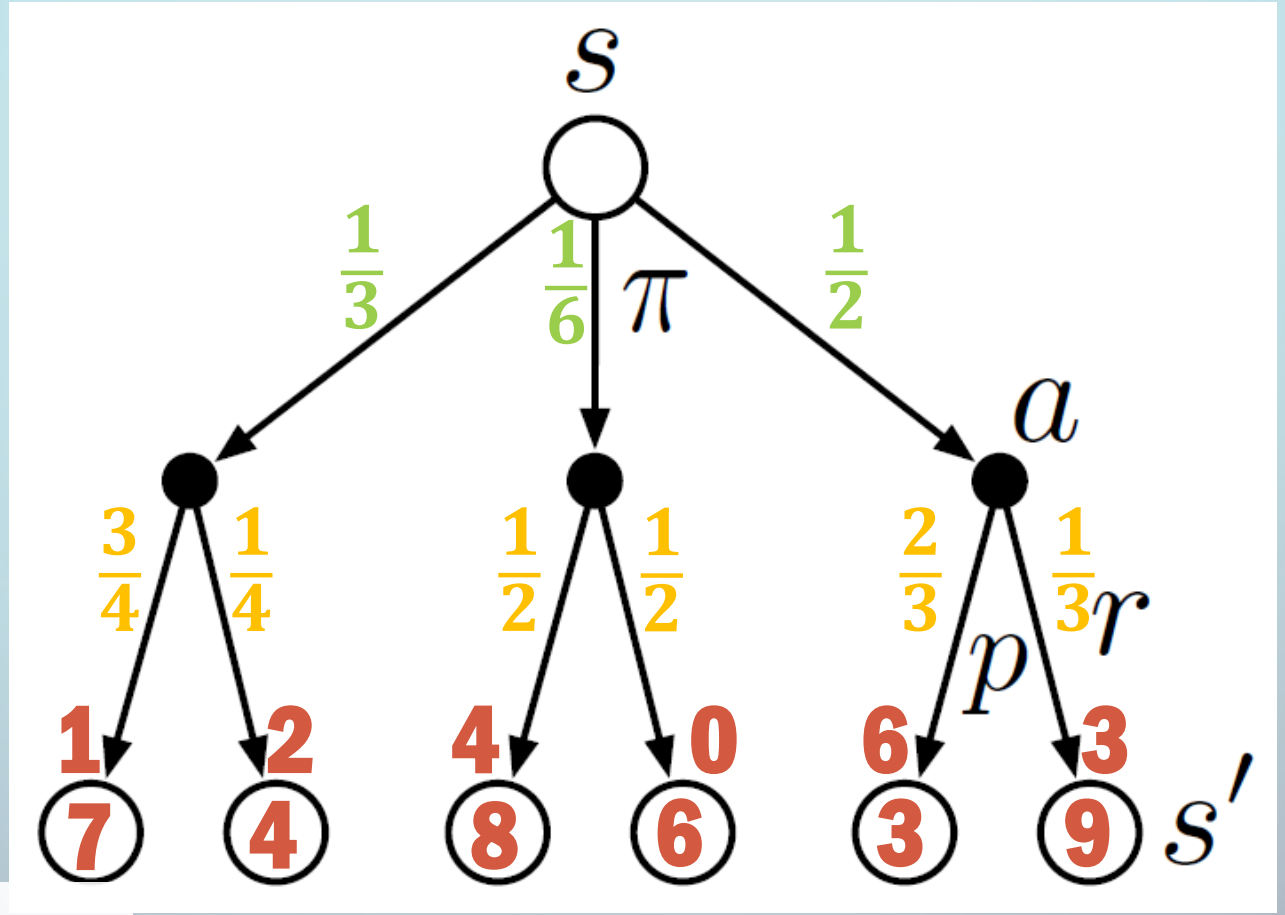
# BELLMAN EQUATION VISUALIZED

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'*



# BELLMAN EQUATION EXAMPLE




$\frac{1}{3} \cdot \frac{3}{4} \cdot (1 + 7) = 2$ 0.5 1 0.5 3 2

$v_\pi(s) = 9$

# BELLMAN EQUATION NOTES

- Here's the Bellman Equation again:

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

- Remember, any time you see $\sum_a \pi(a,s)(XXX)$, the sum of π() times XXX is just adding up the XXXs from each action branch
- And any time you see $\sum_{s',r} p(s',r|s,a)(XXX)$, The sum of $p$() times XXX is just adding up the XXXs from each resulting state branch

# OPTIMAL POLICIES

- Can we find a policy π that is better than all the others?
  - We say $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for every state $s$
  - In RL, we use * to indicate something is optimal
  - For finite MDPs, there is always an optimal policy $\pi_*$
    - Although there may be more than one equivalent optimal policy, we refer to $\pi_*$ as if it were a single one
- Just like we compare policies to find the optimal policy $\pi_*$
  - We can compare policies for the optimal state value function:

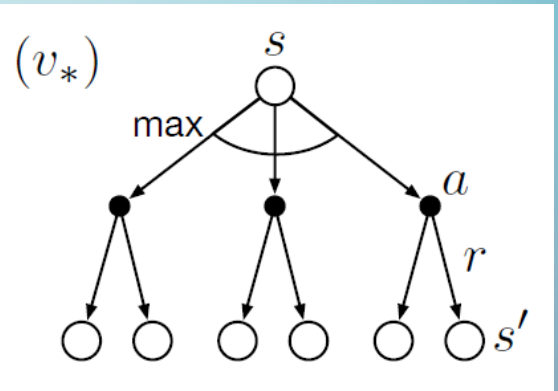
$$v_*(s) = \max_\pi v_\pi(s)$$

  - And we can compare polices for the optimal action-value function:

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

# BELLMAN OPTIMALITY EQUATIONS

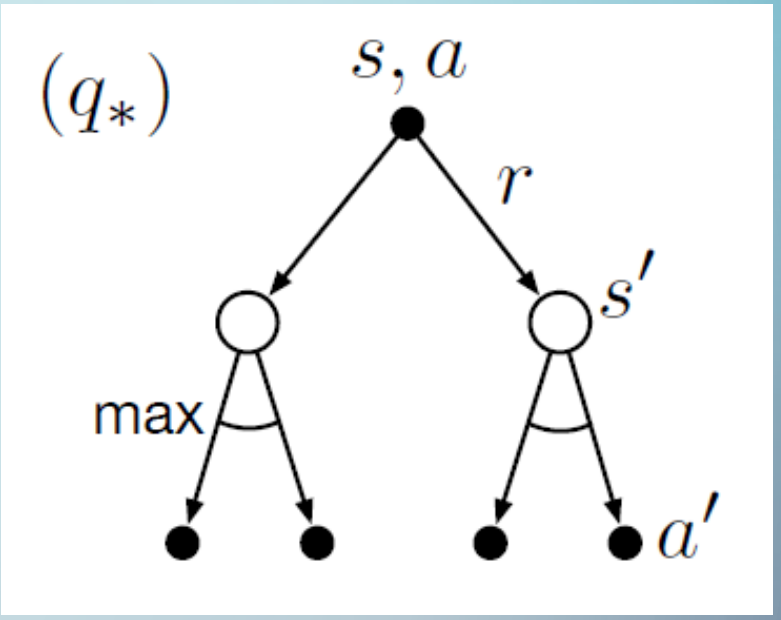
- The *Bellman optimality equation* says the optimal value for a state must be the same as the return from the best action
- We can rewrite it recursively




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

# BELLMAN OPTIMALITY EQUATIONS

- 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




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

# BELLMAN OPTIMALITY

- For finite MDPs, the Bellman optimality equation (eq. 3.19) has a unique solution
  - With $n$ states, you get $n$ equations in $n$ unknowns
  - If the function $p(s', r \mid s, a)$ – which captures the dynamics of your environment – is completely known, then you can solve for $v_*$
    - You could also solve for $q_*$
  - Once you have all values for $v_*$, the optimal policy $\pi_*$ is the greedy one
    - You perform a one-step search of the results of each actions
  - If you know $q_*$, it's even easier. Choose the action with maximum $q_*$
- For small problems with known $p$, you can solve for the optimal policy through a brute force, exhaustive search

# APPROXIMATING OPTIMALITY

- To do the brute force method just mentioned, you need 3 things:
  - The dynamics of the environment are perfectly known
  - The states fully have the Markov property
  - You have enough memory and compute
    - E.g. backgammon has $\sim 10^{20}$ states
- For lack of one or more of the above, we are approximating
- In practice, all but the most trivial problems must be approximated
- The field of reinforcement learning is about the study of ways to approximate value functions and find close to optimal policies

# INCREMENTAL ESTIMATION

- As we explore, we can track of estimates for certain quantities
  - E.g. average return for a slot machine, average wait at a water fountain
- Average wait after $n$ observations of $r$ is $Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} r_i$
- Instead of keeping track of all values of $r$, we can do this in an online fashion
  - Record two quantities: $n$ and the current average denoted $Q_n$
  - At time step $n$, new average $Q_{n+1} = Q_n + \frac{1}{n}(R_n - Q_n)$
- $Q_{n+1} = Q_n + \alpha(R_n - Q_n)$ is the general incremental update form
  - Iteratively add small step to estimate based on the error/difference seen

# INCREMENTAL ESTIMATION

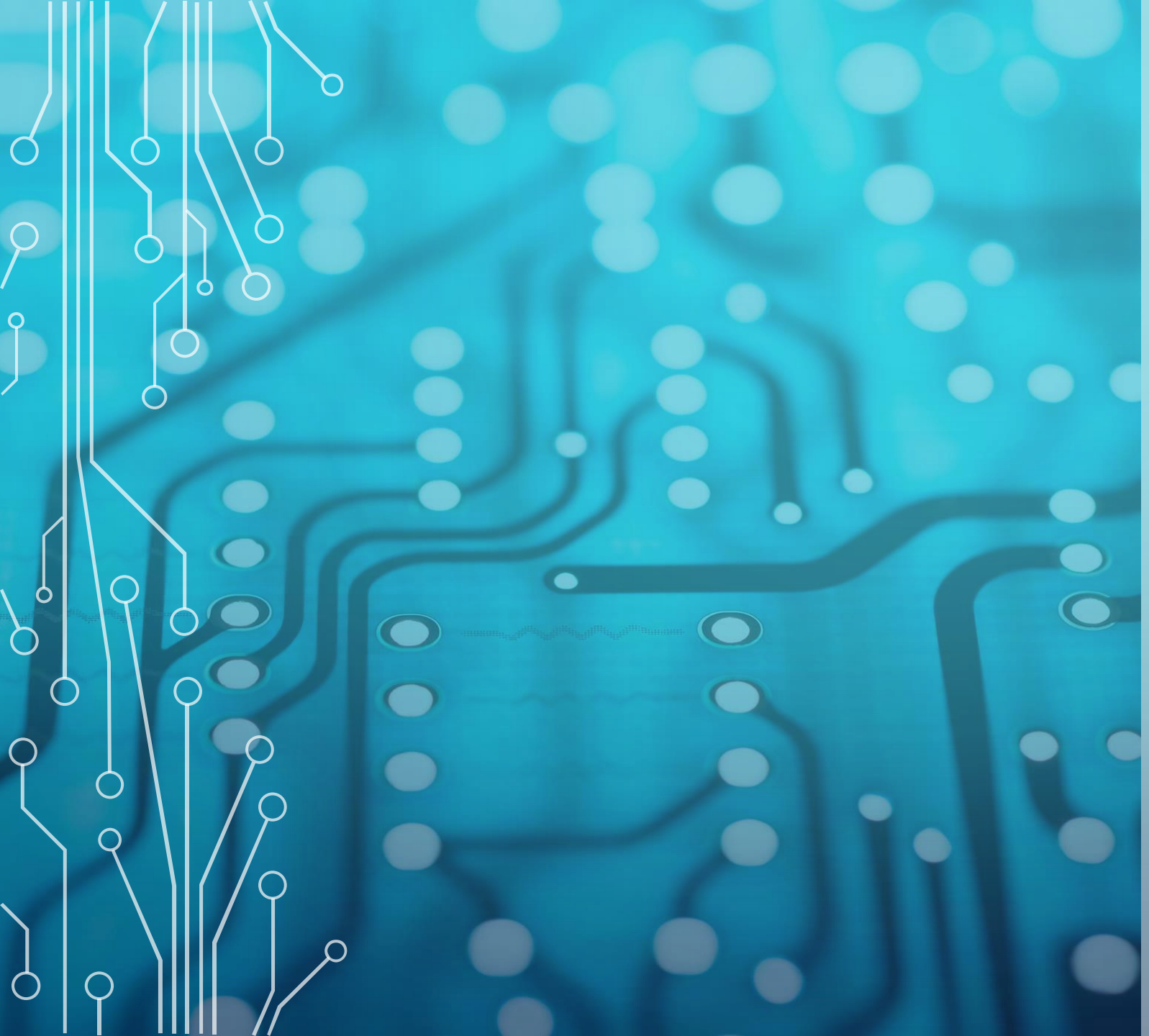
- You can apply iterative improvement to your state value function or your state-action value function
  - This is exactly what we will discuss in the next few sessions
- A lot of the difficulty with reinforcement learning is how to do this iteration in a way that converges
  - Reward design
  - Algorithms for learning
  - Getting model to converge quickly
  - Dealing with memory and compute limitations

# RECAP

- Explain what reinforcement learning (RL) is and how it differs from supervised and unsupervised learning
- Describe the basic elements of RL
- Define RL in terms of Markov decision processes
- Introduce the terminology and notation used in RL
- Show the Bellman equations and simple ways to read them
- Show how optimal solutions can be found for very simple RL problems
- Share some of the challenges with general RL problems
- Give a taste for what more advanced RL algorithms do

# REINFORCEMENT LEARNING NOTATION

| Letter | Used for |
|---|---|
| s | **S**tate |
| a | **A**ction |
| r | **R**eward |
| γ | Discount rate |
| 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. $\pi_*$ |



# QUESTIONS

# &

# DISCUSSION

# NEXT SESSION

- We will finish the content of these slides next week, Sat. May 22
- The following session will be about Dynamic Programming and Monte Carlo methods for reinforcement learning
- This material is in chapters 4 & 5 of Sutton & Barto
- We will likely skip Sat. May 29 (which is Memorial Day weekend) and discuss Dynamic Programming and Monte Carlo methods on Sat. June 5