MONTE CARLO METHODS FOR REINFORCEMENT LEARNING

SAN DIEGO MACHINE LEARNING

JUNE 12, 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

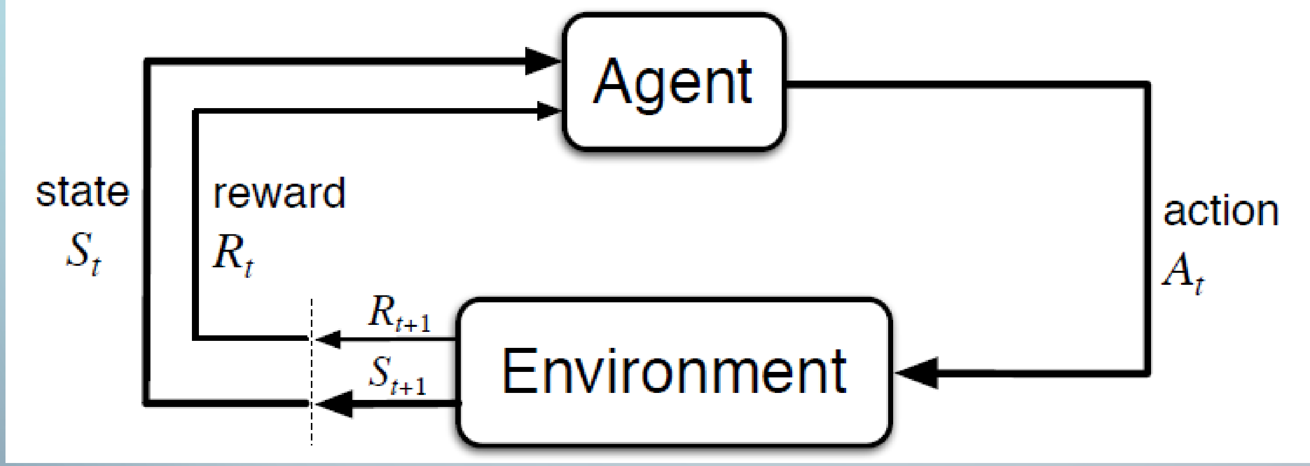
- Recap what reinforcement learning (RL) is
  - Elements and formulation as Markov decision processes (MDP)
  - Terminology and notation used in RL
  - The Bellman equations
- Discuss Monte Carlo Methods
  - Monte Carlo prediction
  - Monte Carlo control
  - On-policy vs. Off-policy
  - Importance Sampling
  - Blackjack code example

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

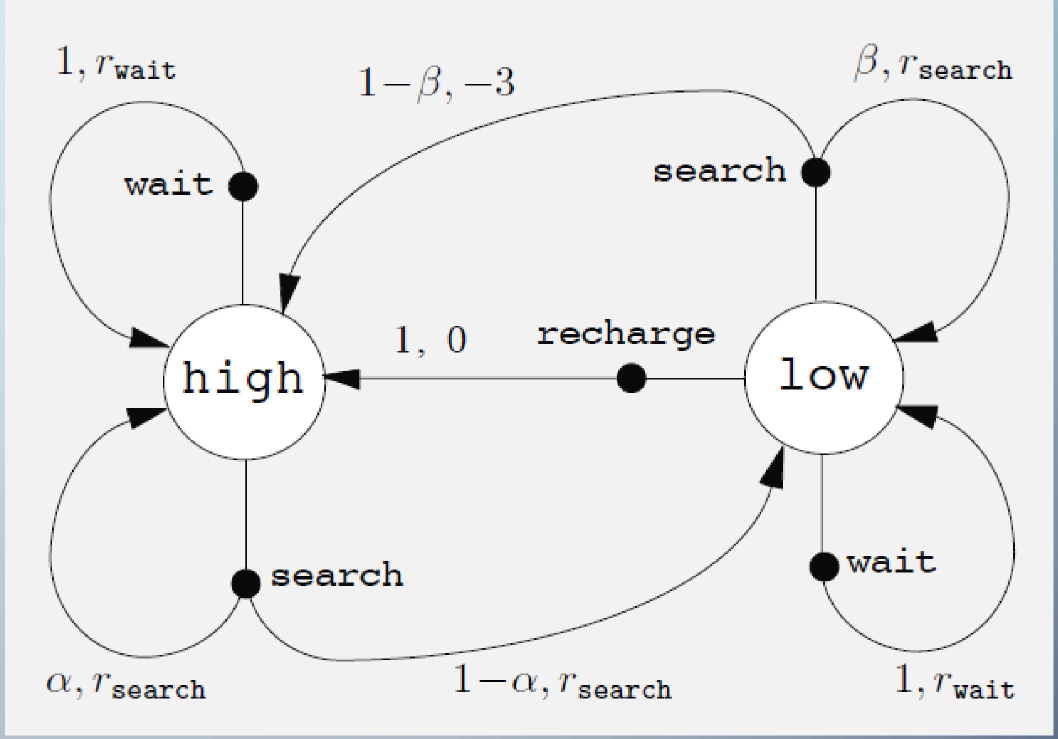
- Elements of the fully observable Markov Decision Process (MDP):
  - State - at each time step t, the environment is in some state S_t
  - Action - at each time step t, the agent chooses an action A_t
  - Reward - after taking the action, the agent is given a reward signal R_{t+1} and subsequently finds 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	**<u>S</u>**tate
a	**<u>A</u>**ction
r	**<u>R</u>**eward
γ	Discount rate
G	Return – sum of all future rewards
p	Transition **<u>p</u>**robability
v	**<u>V</u>**alue function for states
q	Value function for state-action pairs
π	Policy (**<u>π</u>**ολιτική)
*	Optimal choices, e.g. π_*

BELLMAN EQUATION

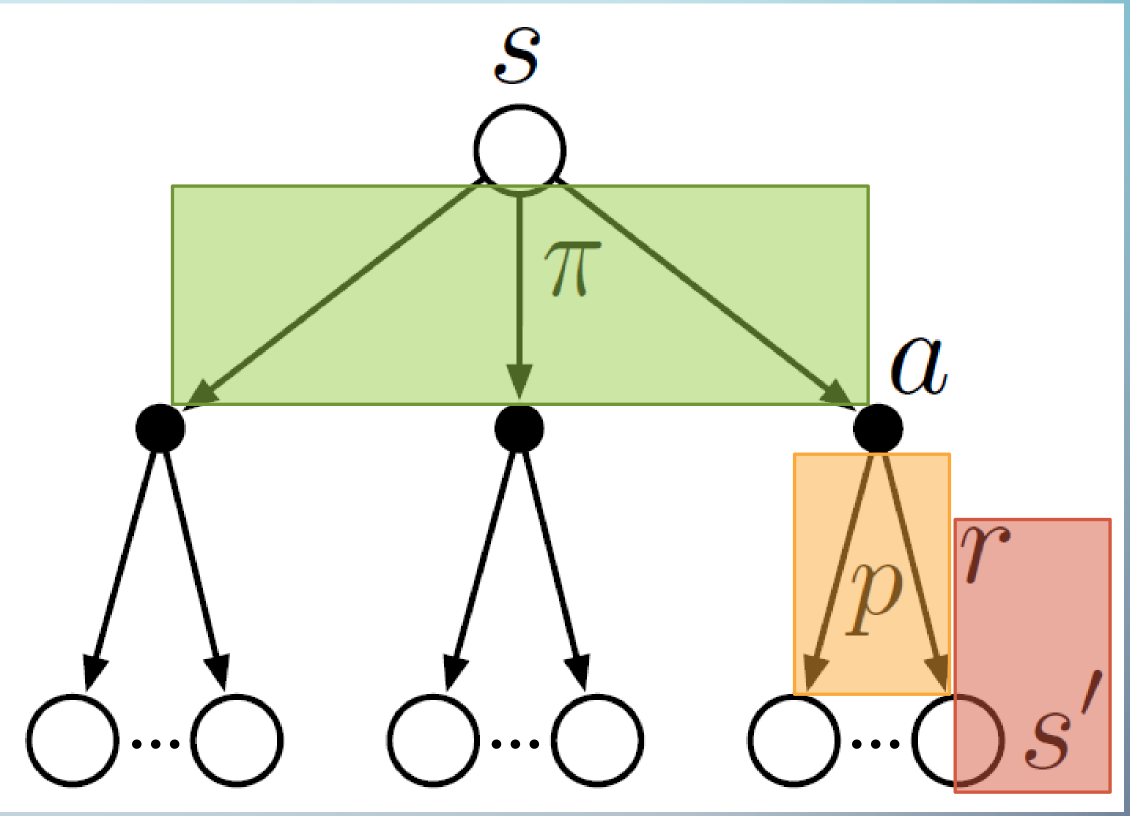
- The value function for state s under policy π is a sum of the rewards received and the value functions for each future state s' times the probability of winding up there
- Formally:

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

BELLMAN EQUATION VISUALIZED

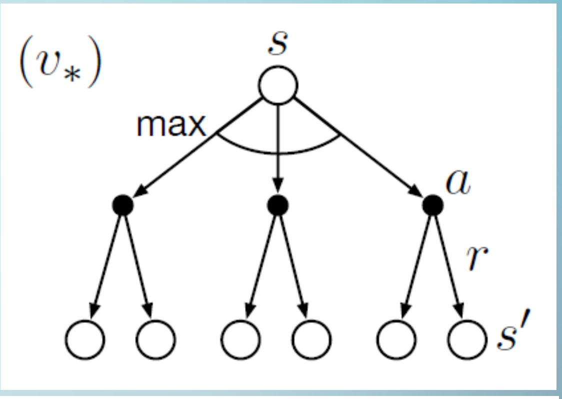
This is a *backup diagram* for $v_\pi(s)$. To compute it:

- We need to sum over each branch of $\pi()$, 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'*



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

BELLMAN OPTIMALITY EQUATIONS

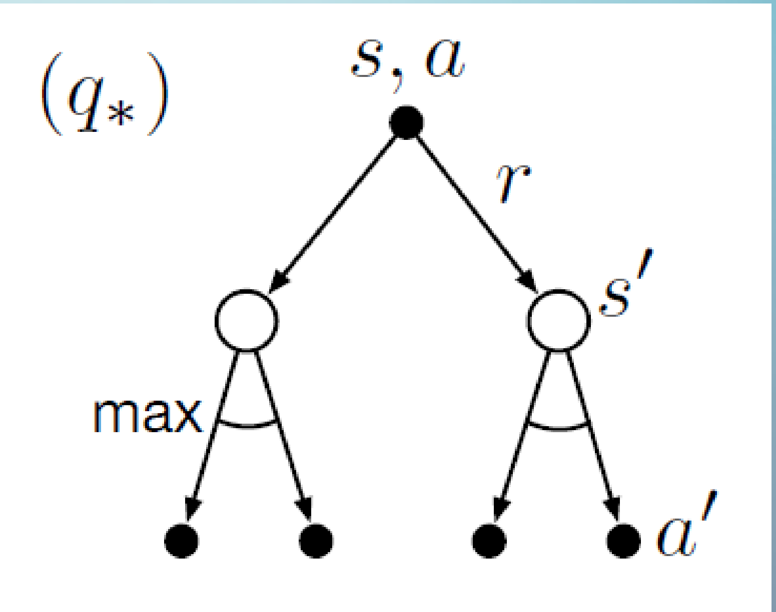


- The *Bellman optimality equation* says the optimal value for a state must be the same as the return from the best action
- We can rewrite it recursively

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

BELLMAN OPTIMALITY EQUATIONS

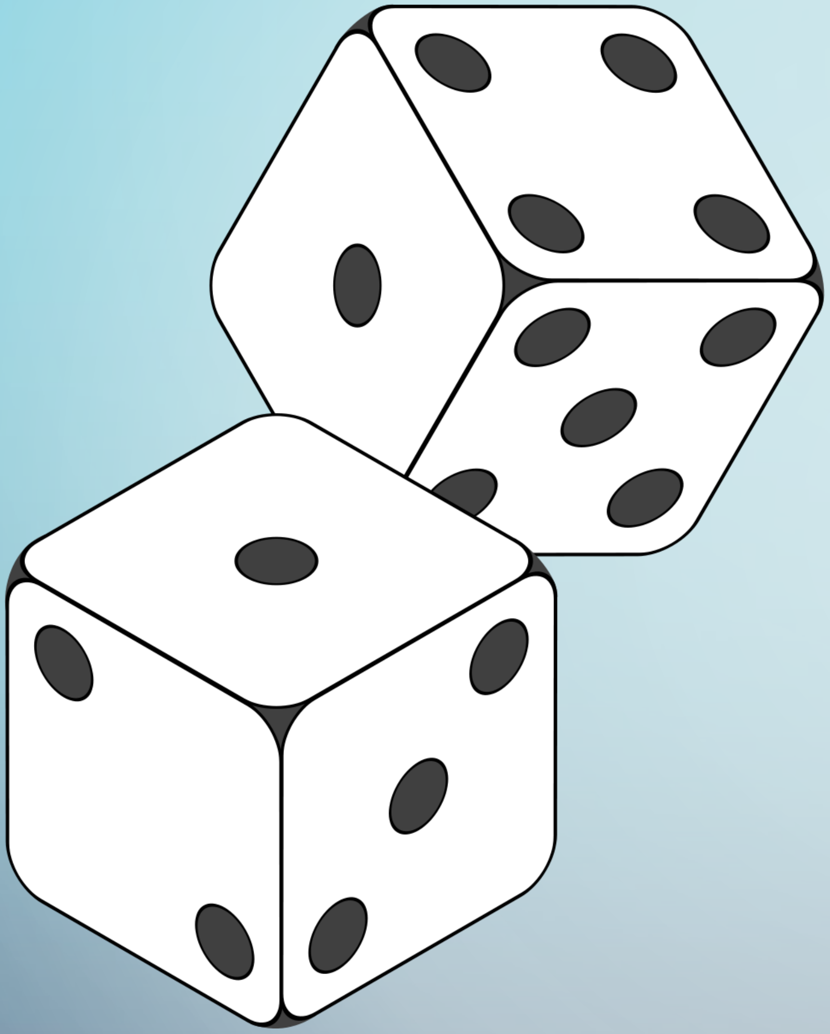
- The *Bellman optimality equation* for state-action pairs is very similar.
- The optimal value for a state-action pair must be the same as the return from the reward and best next action
- It also can be written recursively



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

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



Monte Carlo Methods

MONTE CARLO METHODS

- Monte Carlo methods use experience of the environment to estimate value functions
- They do not require knowledge of the environment's dynamics
- Monte Carlo methods average sampled returns
  - Because we're using returns, works for episodic tasks
  - There are many situations where it's easier to obtain samples transitions than to compute exact transition probability distributions

MONTE CARLO PREDICTION

- The simple case is estimating state values, $v_\pi(s)$. We can follow policy π and average the returns obtained after passing through s
  - Two options are to only track the first time we visit each state in an episode, or to track for every visit to each state

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
      Append G to $Returns(S_t)$
      $V(S_t) \leftarrow \text{average}(Returns(S_t))$

MONTE CARLO PREDICTION

- We can extend this process to the value function for state-action pairs, namely $q_\pi(s, a)$
- Sampling and estimating action values doesn't require a model
- The difficulty is that many state-action pairs may never be visited
- *Exploring starts* specifies that each episode start in particular state-action pairs, and that all pairs must have nonzero probability
  - This is easy for blackjack, but how would you do this for a self-driving car

MONTE CARLO CONTROL

- Policy iteration looks like this:

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

  and we can do the same thing with Monte Carlo evaluation
- When we have estimated $q_\pi(s, a)$ for state-action pairs, then policy improvement is simpler – just choose the max action
- How long do we run each evaluation step?
  - Theory requires infinite samples to ensure convergence
  - A simple approach is just a single episode

MONTE CARLO EXPLORING STARTS

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

  $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$

  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):

  Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0

  Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

    $G \leftarrow \gamma G + R_{t+1}$

    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

      Append G to $Returns(S_t, A_t)$

      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

      $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

ON-POLICY VS. OFF-POLICY

- Exploring starts is an *on-policy* method for ensuring all actions are selected often.
  - In on-policy, our agent is following the same policy it is trying to learn
- Alternative, could use an ε-greedy (or more general ε-soft) policy
  - These guarantee every action will be taken at least ε amount of the time
  - They learn a near-optimal policy that still explores
- Another approach is to use an *off-policy* method
  - In off-policy, you keep a separate policy to track what to do (the *behavior* policy), from the policy you are learning (the *target* policy)
  - Off-policy methods are more powerful and general, but more complex

IMPORTANCE SAMPLING

- For off-policy, we will continue to call our target policy π, and now we will also have a behavior policy *b*
- Since Monte Carlo is about averaging the returns received from sample episodes, if we follow *b*, then we will get average returns under *b*, not under π
- Importance sampling uses the ratio between how often each action is taken under π and under *b* to scale the returns
  - *Ordinary importance sampling* uses average of scaled returns
  - *Weighted importance sampling* divides by the sum of the ratios instead of the number of episodes
    - This helps reduce variance, even though it is introducing unwanted bias

OFF-POLICY MC CONTROL

- For on-policy Monte Carlo, we used general policy iteration, doing one episode of evaluation before policy improvement
- Here is the same thing for off-policy Monte Carlo, except instead storing all of our returns, we incrementally update

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s, a) \in \mathbb{R}$ (arbitrarily)
  $C(s, a) \leftarrow 0$
  $\pi(s) \leftarrow \arg\max_a Q(s, a)$ (with ties broken consistently)

Loop forever (for each episode):
  $b \leftarrow$ any soft policy
  Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

MONTE CARLO SUMMARY

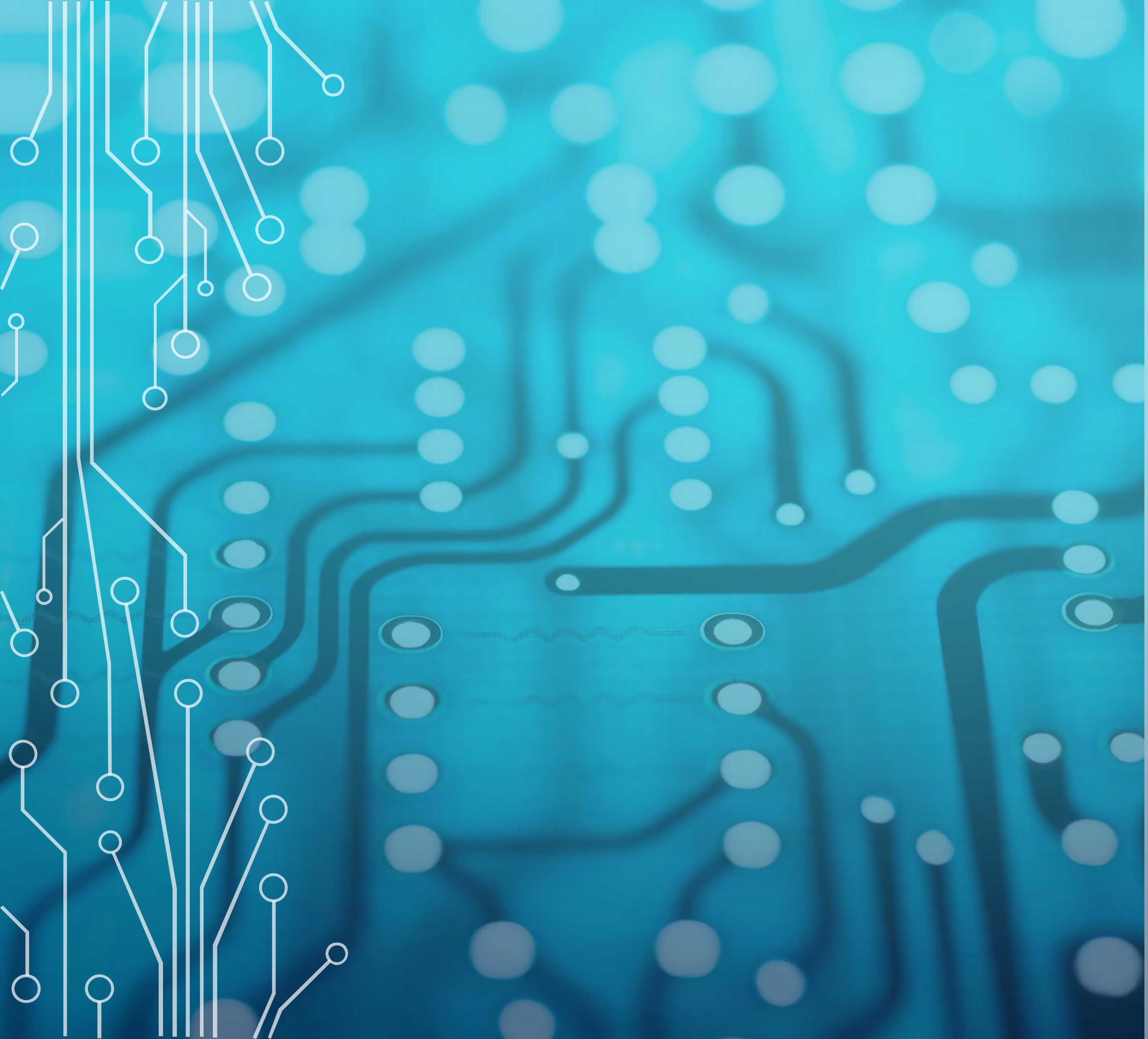
- Three big benefits of Monte Carlo methods
  - Can learn directly from interaction, without a model
  - Can work with simulated episodes even where transition probabilities are difficult to precisely calculate
  - Learning can be focused on certain states more than others
  - Also, it turns out MC methods may work better when Markov property violated because they don't bootstrap
- MC methods require sufficient exploration (or else value of certain states/actions won't be accurate)
- MC methods are the first time we have seen *off-policy* prediction using a *behavior policy*

CODE EXAMPLE

- On GitHub, Python code for examples in the Sutton & Barto book are in this repository: https://github.com/ShangtongZhang/reinforcement-learning-an-introduction
- We will look at some of the code in the `Chapter05` folder, in the file `blackjack.py`

RECAP

- Review what reinforcement learning (RL) is
  - Elements and formulation as Markov decision processes (MDP)
  - Terminology and notation used in RL
  - The Bellman equations
- Discuss Monte Carlo Methods
  - Monte Carlo prediction
  - Monte Carlo control
  - On-policy vs. Off-policy
  - Importance Sampling
  - Blackjack code example



QUESTIONS

&

DISCUSSION

NEXT SESSION

- The next session will be about Temporal-Difference Learning, on Sat. June 19
- This TD material is in chapter 6 of Sutton & Barto