POLICY GRADIENT METHODS IN REINFORCEMENT LEARNING

SAN DIEGO MACHINE LEARNING

JULY 31, 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

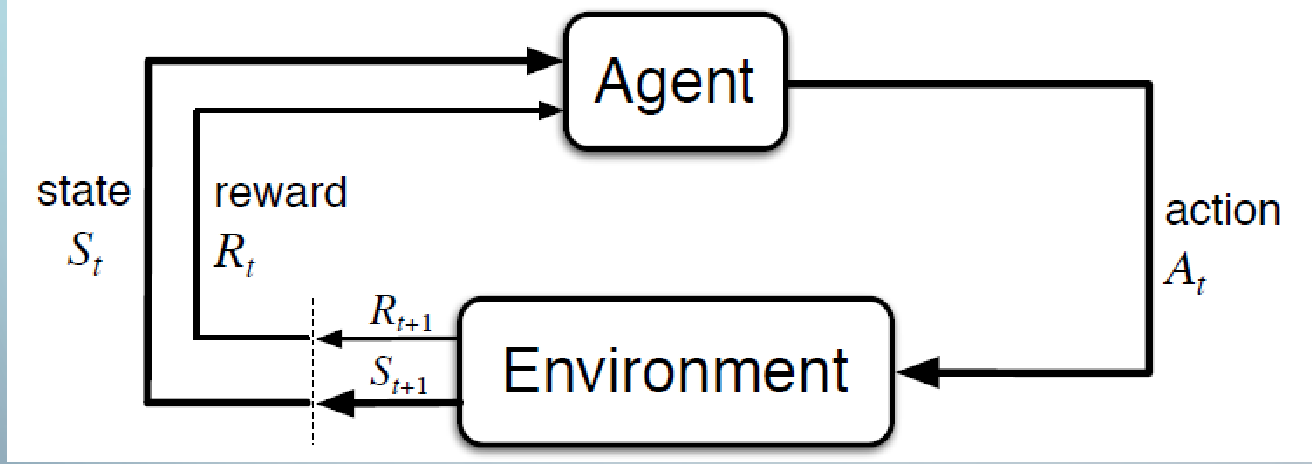
- Recap what reinforcement learning (RL) is
  - Elements and formulation as Markov decision processes (MDP)
  - Terminology and notation used in RL
  - The Bellman equations
  - Generalized policy iteration
- Policy gradient methods
  - Policy gradient
  - REINFORCE algorithm
  - Actor-critic methods
  - List some policy gradient methods

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

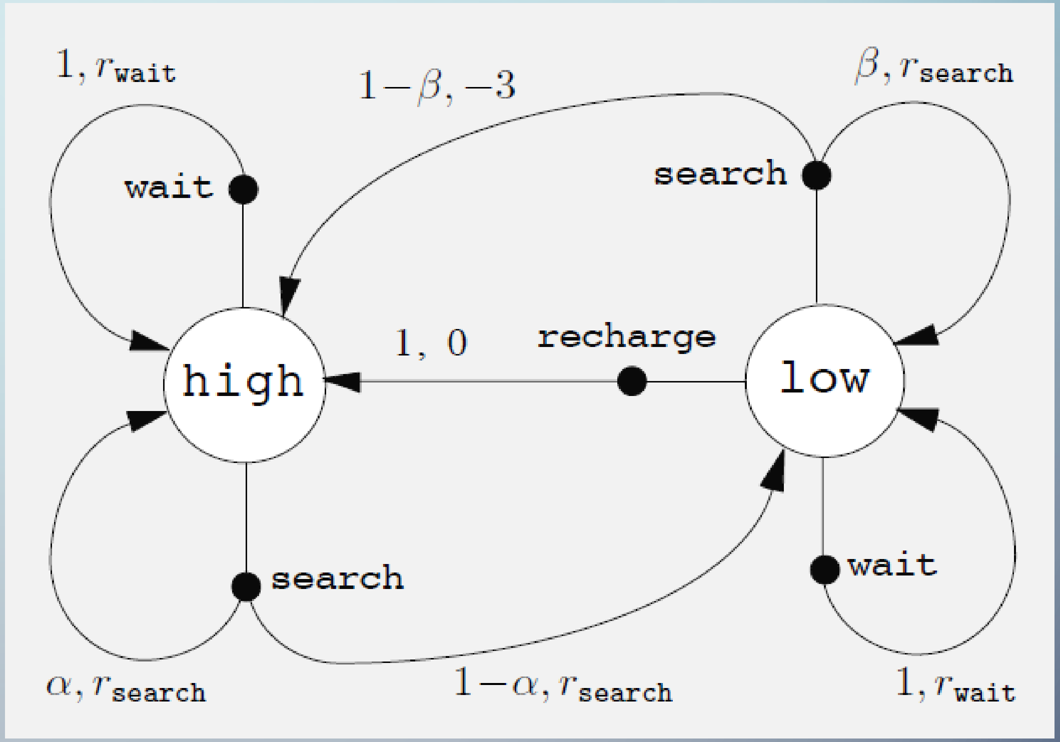
- Elements of the fully observable Markov Decision Process (MDP):
  - State - at each time step t, the environment is in some state S_t
  - Action - at each time step t, the agent chooses an action A_t
  - Reward - after taking the action, the agent is given a reward signal R_{t+1} and subsequently finds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

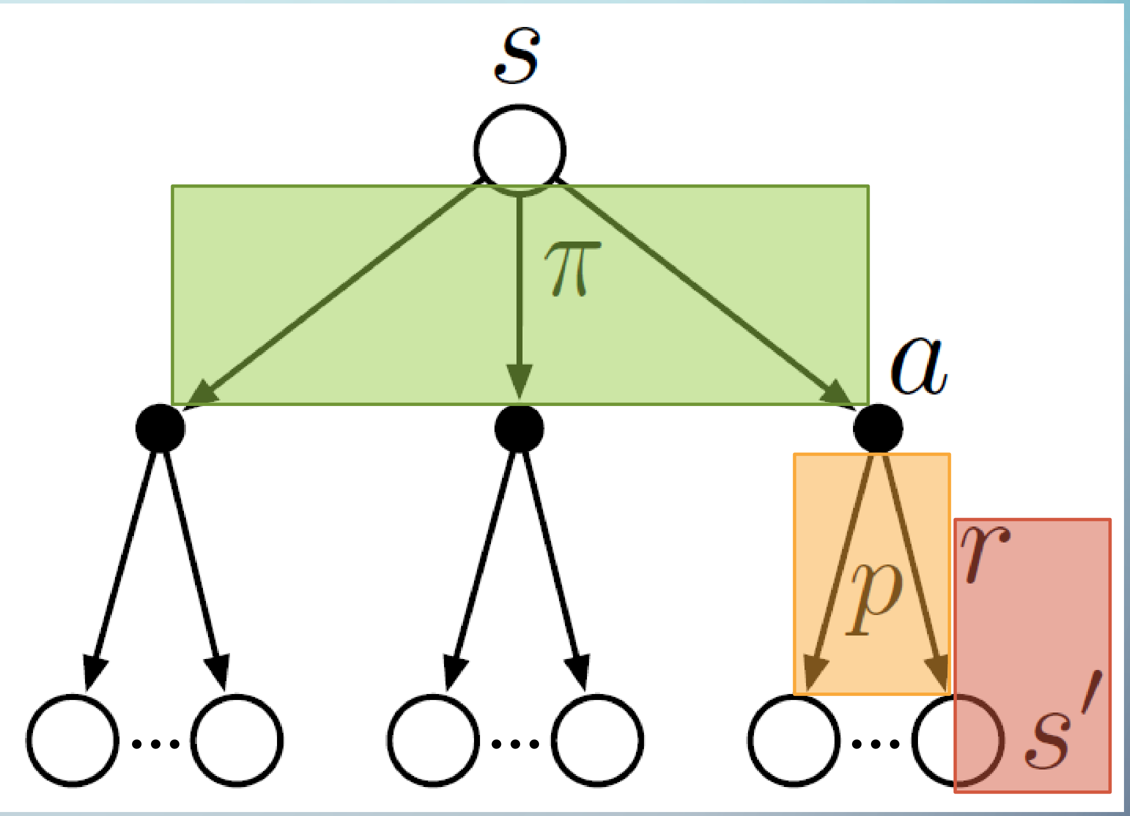
- The value function for state s under policy π is a sum of the rewards received and the value functions for each future state s' times the probability of winding up there
- Formally:

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

BELLMAN EQUATION VISUALIZED

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

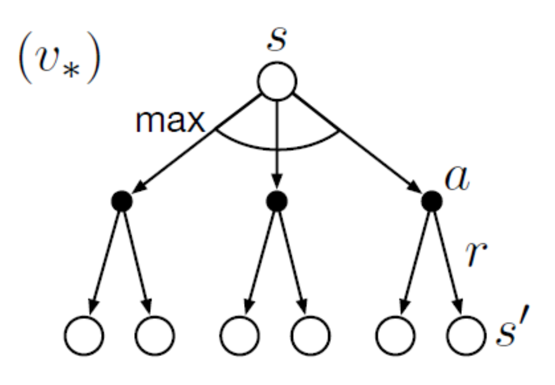
- We need to sum over each branch of π(), based on the probability of each action *a*
- And sum over of each branch of *p*(), based on probability we wind up in state *s'*
- The quantity we sum is the reward and the discounted value of possible state *s'*



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

BELLMAN OPTIMALITY EQUATIONS – V()

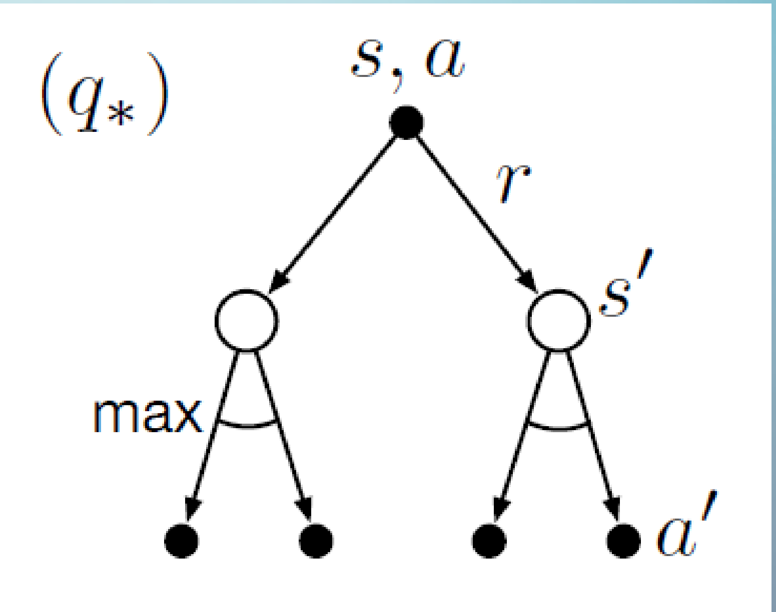
- The *Bellman optimality equation* says the optimal value for a state must be the same as the return from the best action
- We can rewrite it recursively



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

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



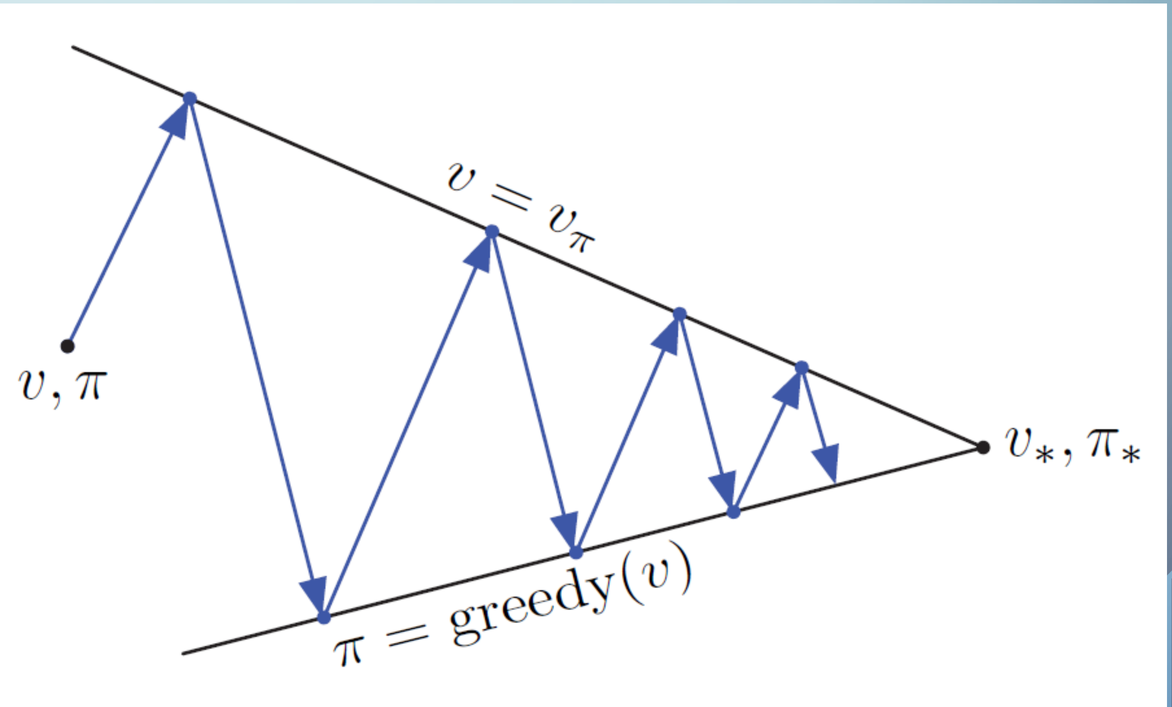
$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big] \\
&= \sum_{s',r} p(s', r | s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big]. \qquad (3.20)
\end{aligned}
$$

POLICY ITERATION

- The book shows a sequence like this:

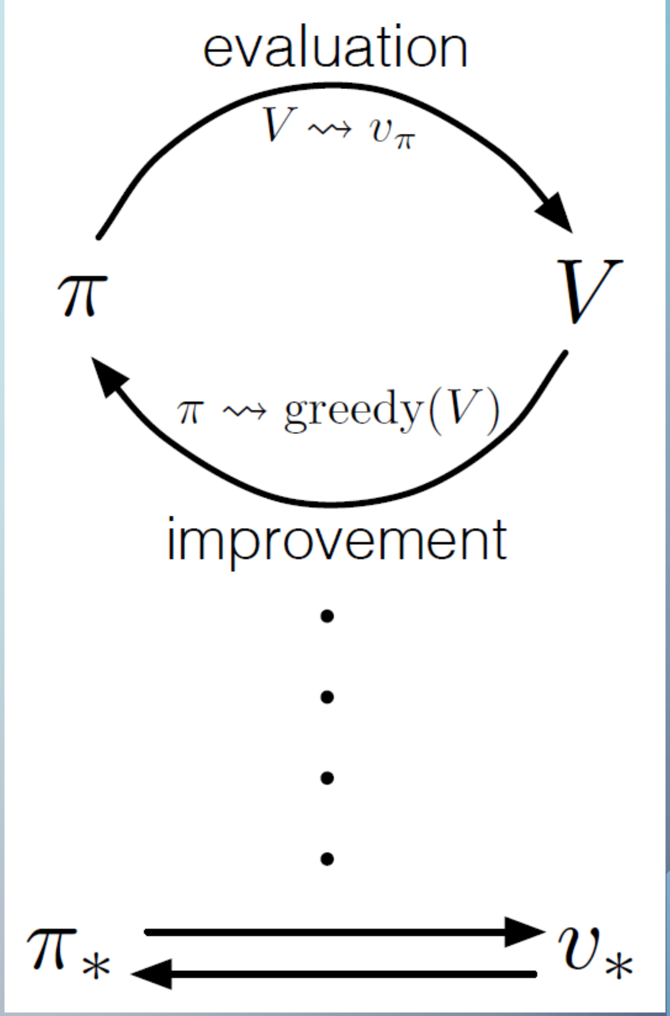
$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_1 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

- 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

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 $Returns(S_t, A_t)$
    - $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
    - $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

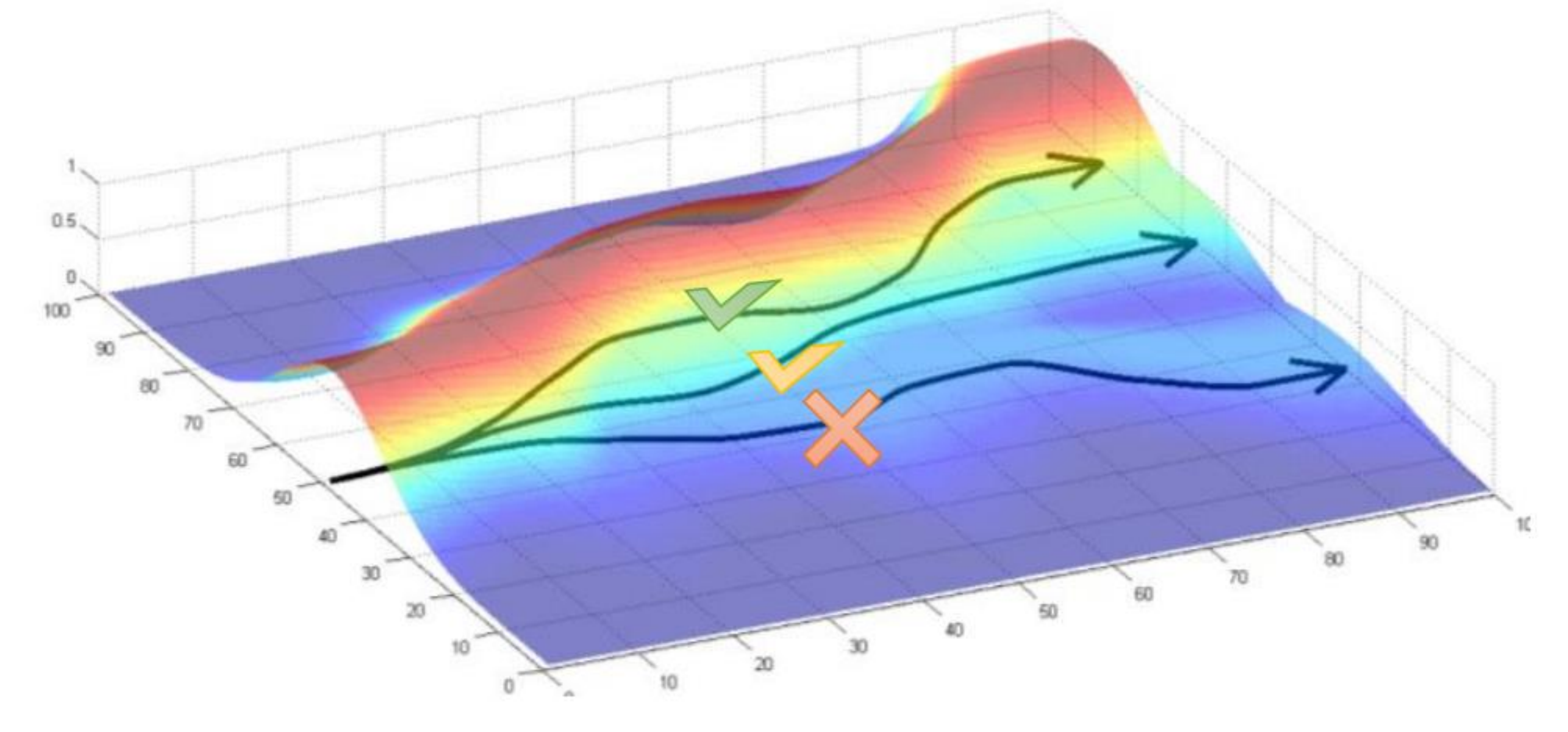
TD(0)

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 $V(\textit{terminal}) = 0$

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'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

POLICY GRADIENT METHODS

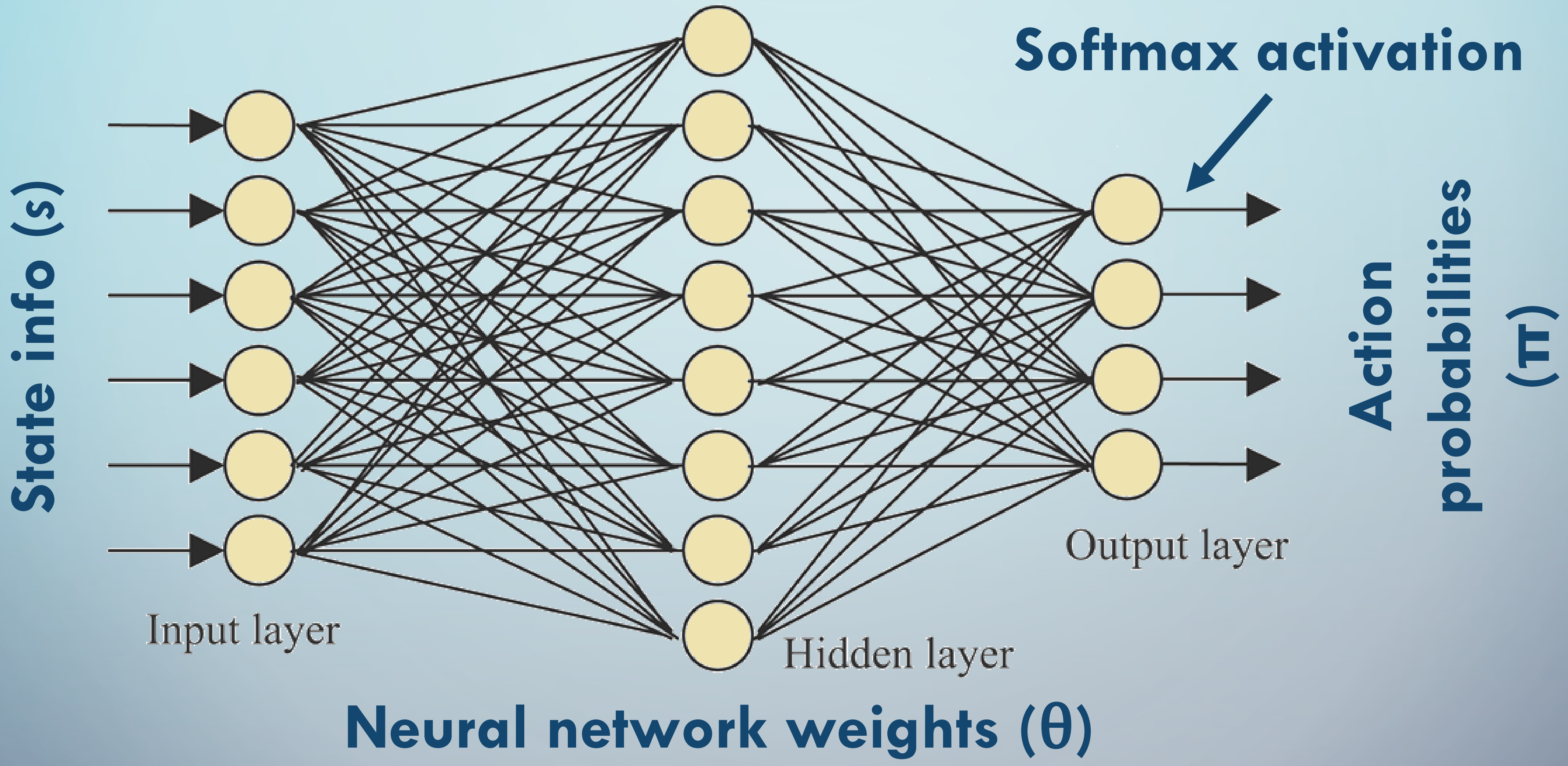


POLICY GRADIENTS

- Prior discussions were all about estimating value functions, and then using those to derive good policies
- With policy gradients, you build a parameterized policy
  - Start with some parameters θ (a vector)
  - Build a complex function f() which outputs a vector of action probabilities
- So we are generating actions without needing models or value functions as intermediaries
- The book uses the notation:
  - $\pi(a|s,\theta)=\Pr\{A_t = a|St = s, \theta_t = \theta\}$
- I'm saying in simpler language:
  - $\pi(a|s,\theta)=f(s,\theta)$

PARAMETERIZED POLICY

- A neural network example to construct this function f():



HOW POLICY GRADIENTS WORK

- After parameterizing our policy on θ…
- We create a scalar performance measure J(θ)
  - This function J(θ) clearly must have something to do with the rewards that we get, in order to be helpful
  - For episodic MDPs starting in state s_0, we define $J(\theta) \doteq v_{\pi_\theta}(s_0)$
  - We're familiar with a scalar loss function L(θ) for neural networks
- We wish to maximize performance, so we perform gradient *ascent*
  - Again, this is analogous to gradient descent on our loss function
- We can iterate small tweaks to θ with learning rate α:

$$\theta_{t+1} = \theta_t + \alpha \cdot \nabla J(\theta_t)$$

ADVANTAGES OF POLICY-BASED RL

- Before getting into more details about how policy gradients work:
- Advantages
  - Sometimes policy space is simpler than value space
  - Better convergence properties
  - Effective in high-dimensional and continuous action spaces
  - Can learn stochastic (mixed) policies
- Disadvantages
  - Alternatively, sometimes the value space is simpler than the policy space
  - Typically converges to a local optimum, not the global optimum
  - Evaluating a policy is typically (sample) inefficient and high variance

CALCULATING THE POLICY GRADIENT

- Conceptually, we will tweak our parameters based on the gradient of the performance function, $\nabla J(\theta_t)$, but how do we calculate this gradient?
- In supervised learning, the loss function is usually relatively simple, and we can easily calculate the partial derivative analytically
- Here, return is a long sum of products involving the environment's dynamics
- In the Andrew Ng ML course the gradient is calculated by finite differences, where you perturb each dimension by a small ϵ

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

where u_k is the unit vector in the k^{th} dimension of θ

POLICY GRADIENT THEOREM

- We might expect it to be difficult to tweak θ to steadily improve J(θ), because changing θ not only changes the policy's actions, but also indirectly changes the distribution of states you visit
- We're assuming we are doing model-free learning, and don't know the state distribution function of the environment
- The policy gradient theorem provides an analytic expression for the gradient of performance that does not use the derivative of the state distribution
- So we can calculate the gradient analytically without knowing the model dynamics

POLICY GRADIENT THEOREM [2]

- The policy gradient theorem tells us the gradient is proportional to the following quantity:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}), \tag{13.5}$$

  where $\mu(s)$ is the distribution of states when following policy π
  - The constant of proportionality has to do with the length of the episode, but since we are multiplying the gradient by a step size α, we can absorb this scaling factor into our choice of α
- We can reformulate the above as:

$$\nabla J(\theta) = E[q_\pi(s,a) \nabla \log \pi(a|s,\theta)]$$

REINFORCE: MONTE CARLO POLICY GRADIENT

- If we consider the perspective of a given state S_t and action A_t in an episodic MDP, the previous equation becomes:

$$\nabla J(\theta) = E[G_t \nabla \log \pi(A_t|S_t, \theta)]$$

- And our update rule is:

$$\theta_{t+1} = \theta_t + \alpha G_t \nabla \log \pi(A_t|S_t, \theta_t)$$

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

REINFORCE WITH BASELINE

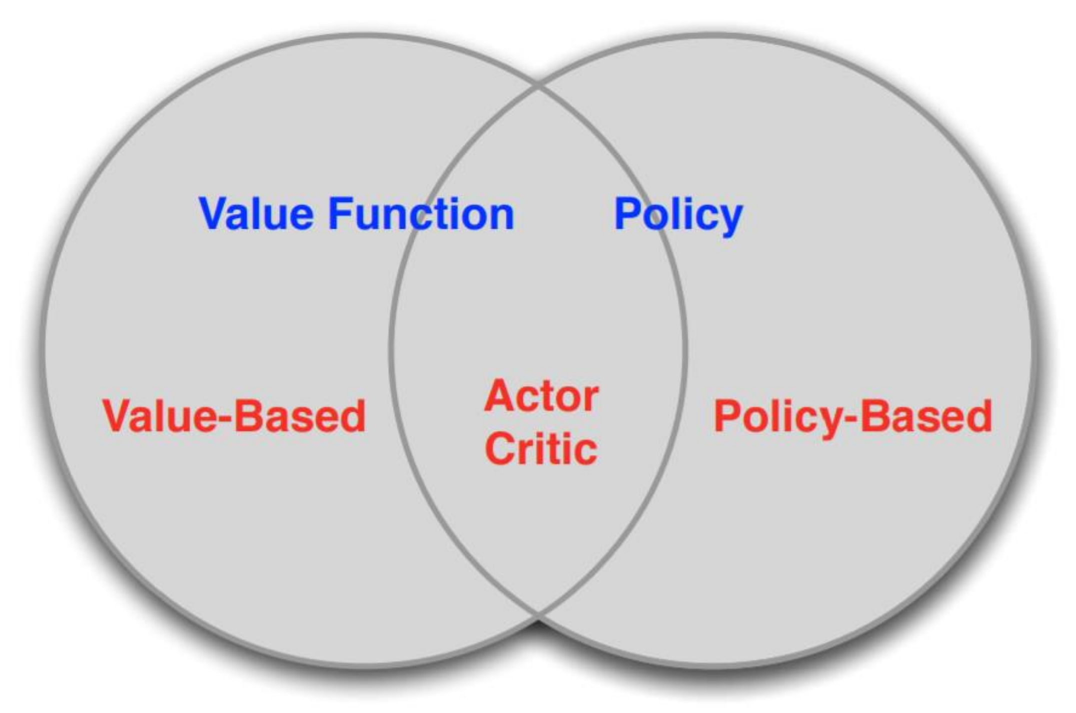
- The REINFORCE algorithm is the simplest form of policy gradient control
- The returns G_t are an unbiased estimate of $q_\pi(s, a)$, but they are high variance
- It can be shown that REINFORCE remains unbiased if you subtract a baseline that can depend on the state, but not the action. Our new update is:

$$\theta_{t+1} = \theta_t + \alpha(G_t - b(S_t))\nabla \log \pi(A_t|S_t, \theta_t)$$

  - A natural choice for b() would be the estimated value function $v(S_t)$
  - We can use our Monte Carlo samples to simultaneously update our estimate of the value function and do our policy gradient updates

ACTOR-CRITIC METHODS

- In actor-critic methods, our policy gradient learner is the *actor*, and the *critic* is a learned value function that provides some form of guidance/feedback to the way the actor learns
- A simple thing we can do is use one-step reward signal instead of full episodic returns, the same way we moved from Monte Carlo methods to temporal-difference learning
  - And just like TD, this introduces bias, but reduces variance



ONE-STEP ACTOR-CRITIC

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

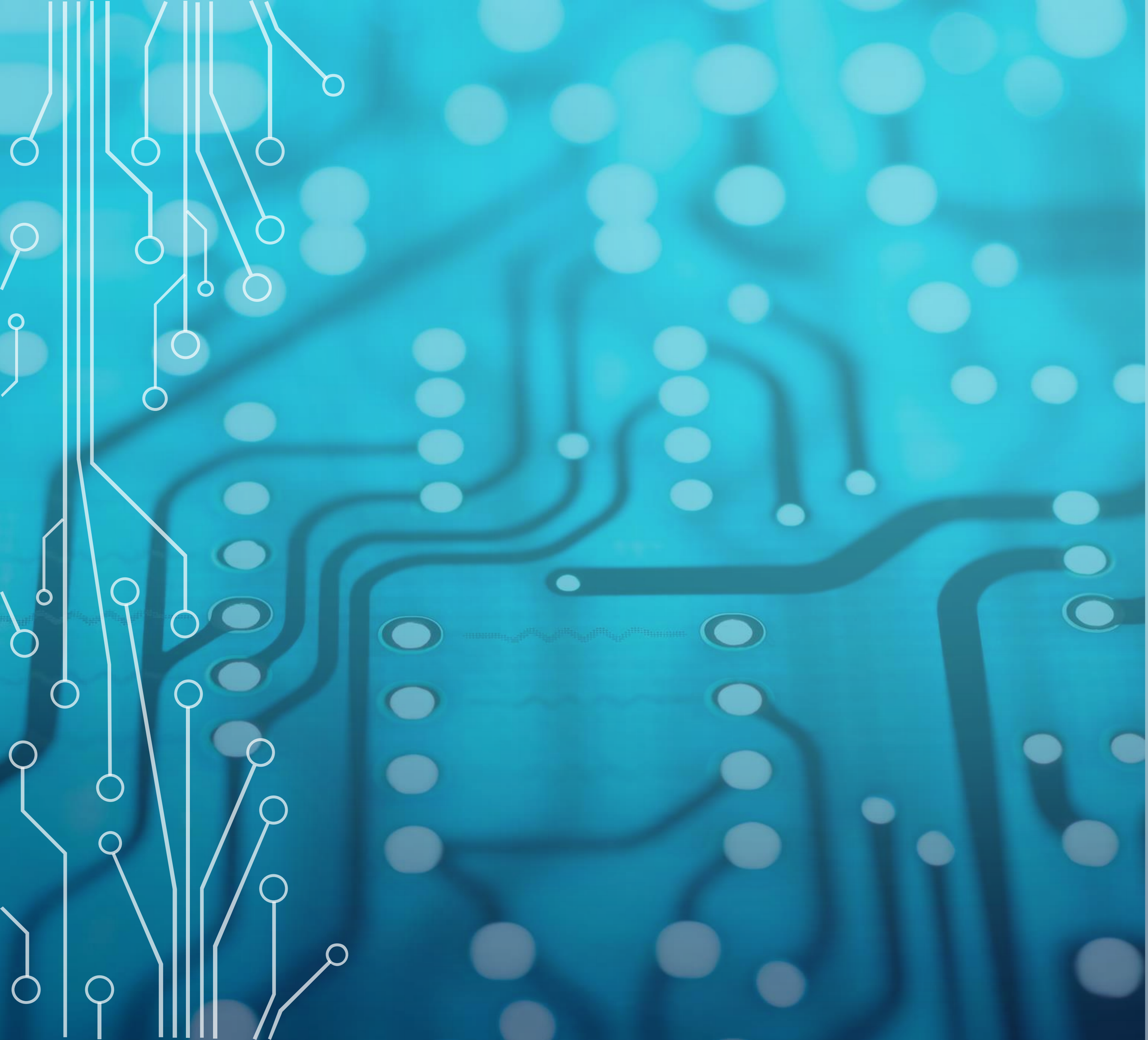
SOME POLICY GRADIENT METHODS

- A2C – Advantage Actor-Critic (synchronous). Multiple actors; the *advantage* is the reward minus the average reward
- DDPG – Deep Deterministic Policy Gradient. Deterministic policies
- TRPO – Trust Region Policy Optimization. Clip max value of updates
- PPO – Proximal Policy Optimization. Simpler clipping that TRPO
- SAC – Soft Actor-Critic. Incorporates entropy of policy to encourage exploration
- TD3 – Twin Delayed Deep Deterministic. Uses tricks from Double DQN applied to DDPG

https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

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
- Policy gradient methods
  - Policy gradient
  - REINFORCE algorithm
  - Actor-critic methods
  - List some policy gradient methods



QUESTIONS

&

DISCUSSION

NEXT SESSION

- In two weeks, Sat. August 14, Ryan will talk about reinforcement learning techniques in AlphaGo