Course 3, Module 3
Control with Approximation

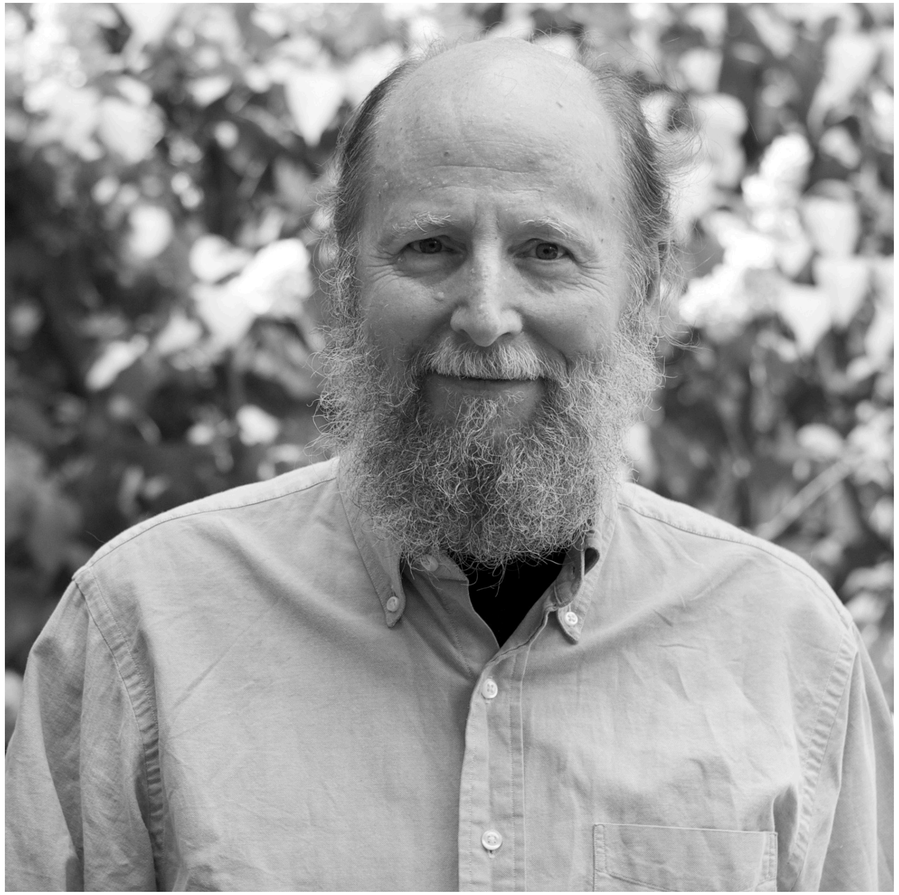
CMPUT 365
Fall 2021

Announcements

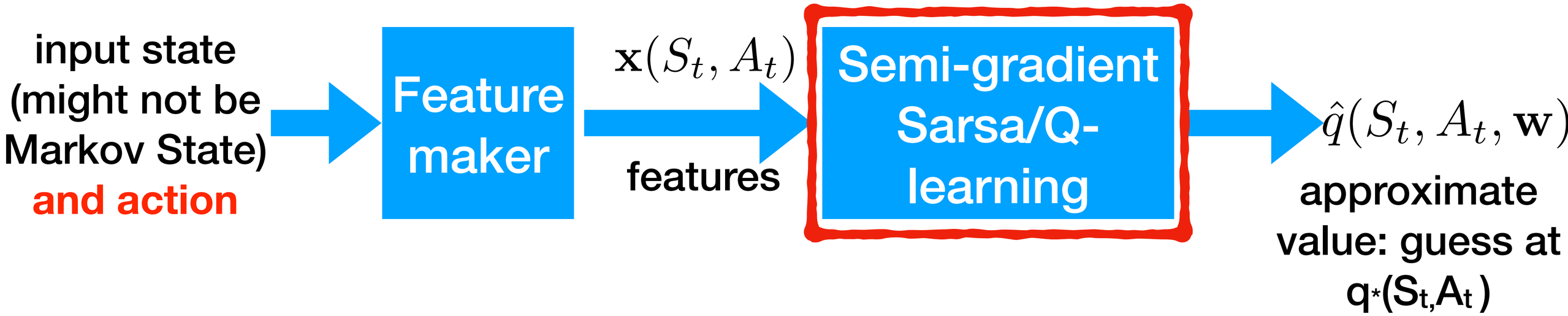
- Don’t forget Mini-essay #2 and Capstone Project. Dates in course schedule gsheet
- No class on Monday! Instead, we will review the Practice Final on another day. I will do a poll to see which day s best

Announcements

- Rich Sutton Lectures Friday!!



Review of Course 3, Module 3
How to do control (learn a good policy) with function approximation


input state
(might not be
Markov State)
and action
Feature
maker
$\mathbf{x}(S_t, A_t)$
features
Semi-gradient
Sarsa/Q-
learning
$\hat{q}(S_t, A_t, \mathbf{w})$
approximate
value: guess at
$q_*(S_t, A_t)$

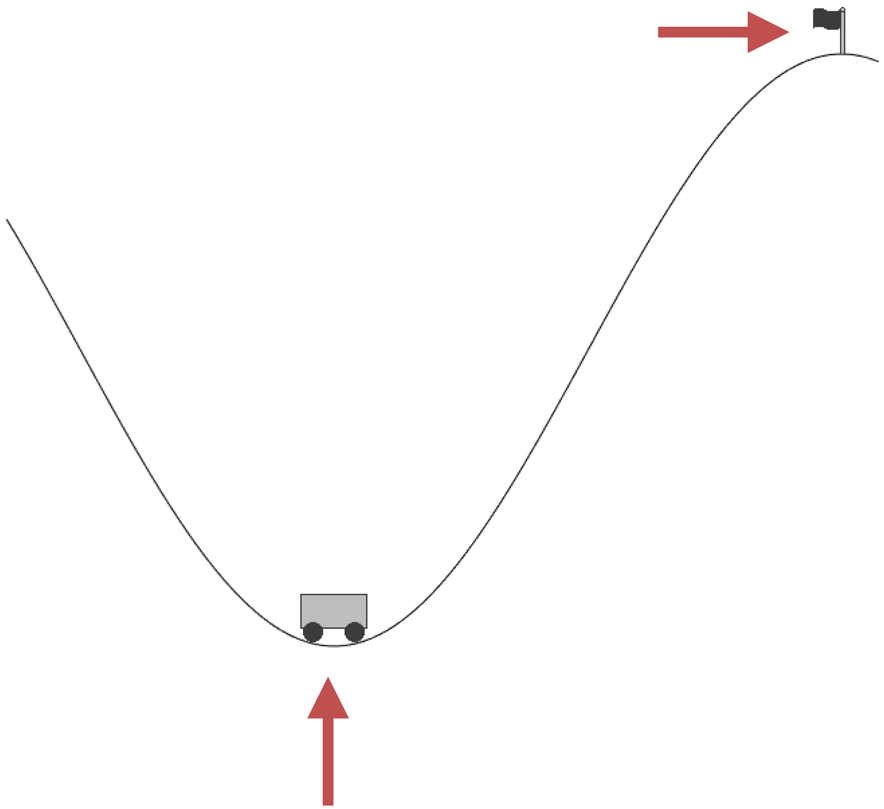
Video 1: Episodic Sarsa with Function Approximation

- We know how to do function approximation with TD; how about using that to learn action-values and a policy. **On-policy TD control** with approximation
- Goals:
  - Understand how to construct **action-dependent features** for approximate action-values >> stacking
  - and explain how to use Sarsa in episodic tasks with function approximation
- In Episodic Sarsa with Tile Coding an example of GPI?

Video 2: Episodic Sarsa in Mountain Car

- **Can we do a large number of states with Semi-gradient Sarsa?** How about an infinite number of state? Yep. We do a classic control task: **Mountain Car**
- Goals:
  - gain experience analyzing the performance of an approximate TD control method
- Name one computational benefit of using Tile Coding for linear function approximation. **Hint**: how do we estimate the value function …

The Mountain Car environment



$$R_{step} = -1$$

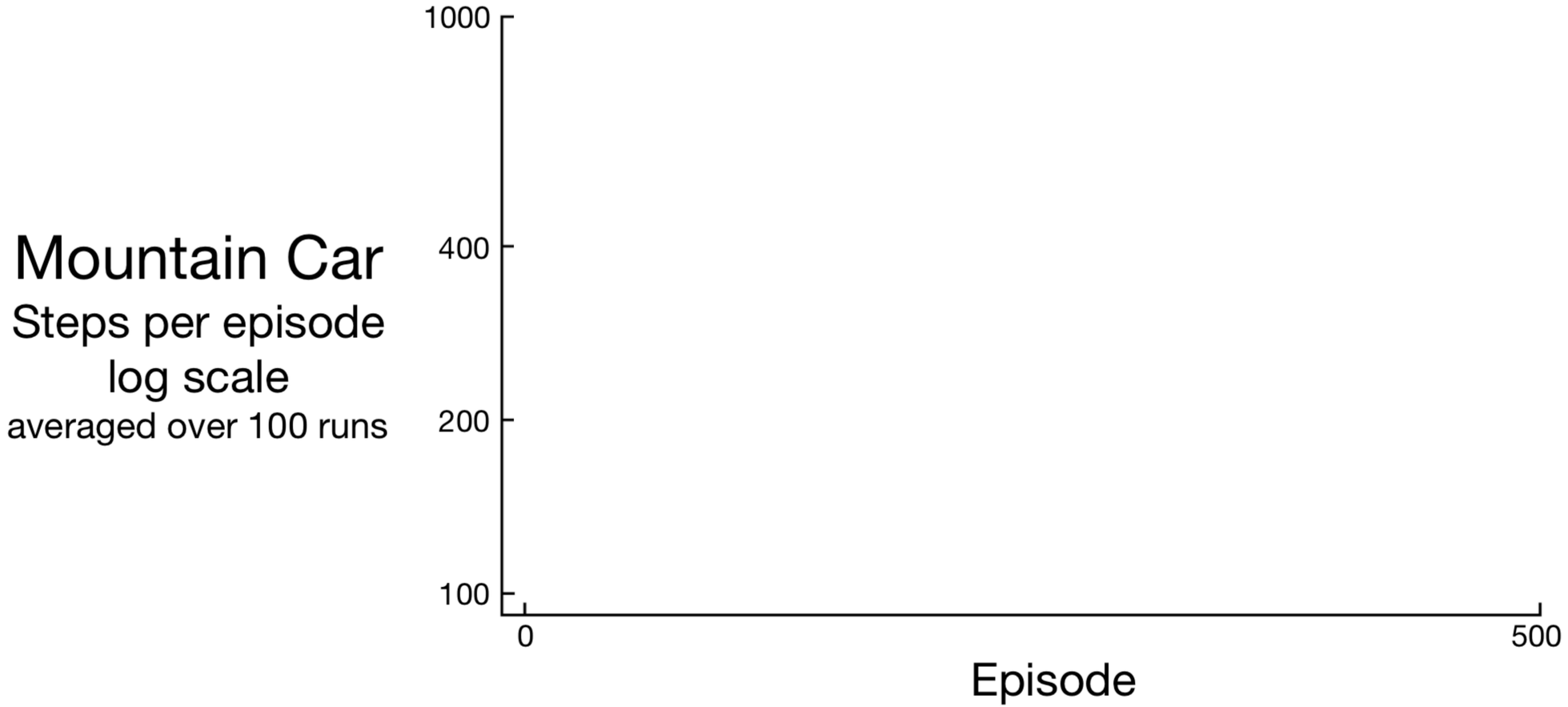
$$\gamma = 1.0$$

State: Car position
Car velocity

Actions: Accelerate right
Accelerate left
Coast (no acceleration)

- What is a simple near-optimal policy for this domain?

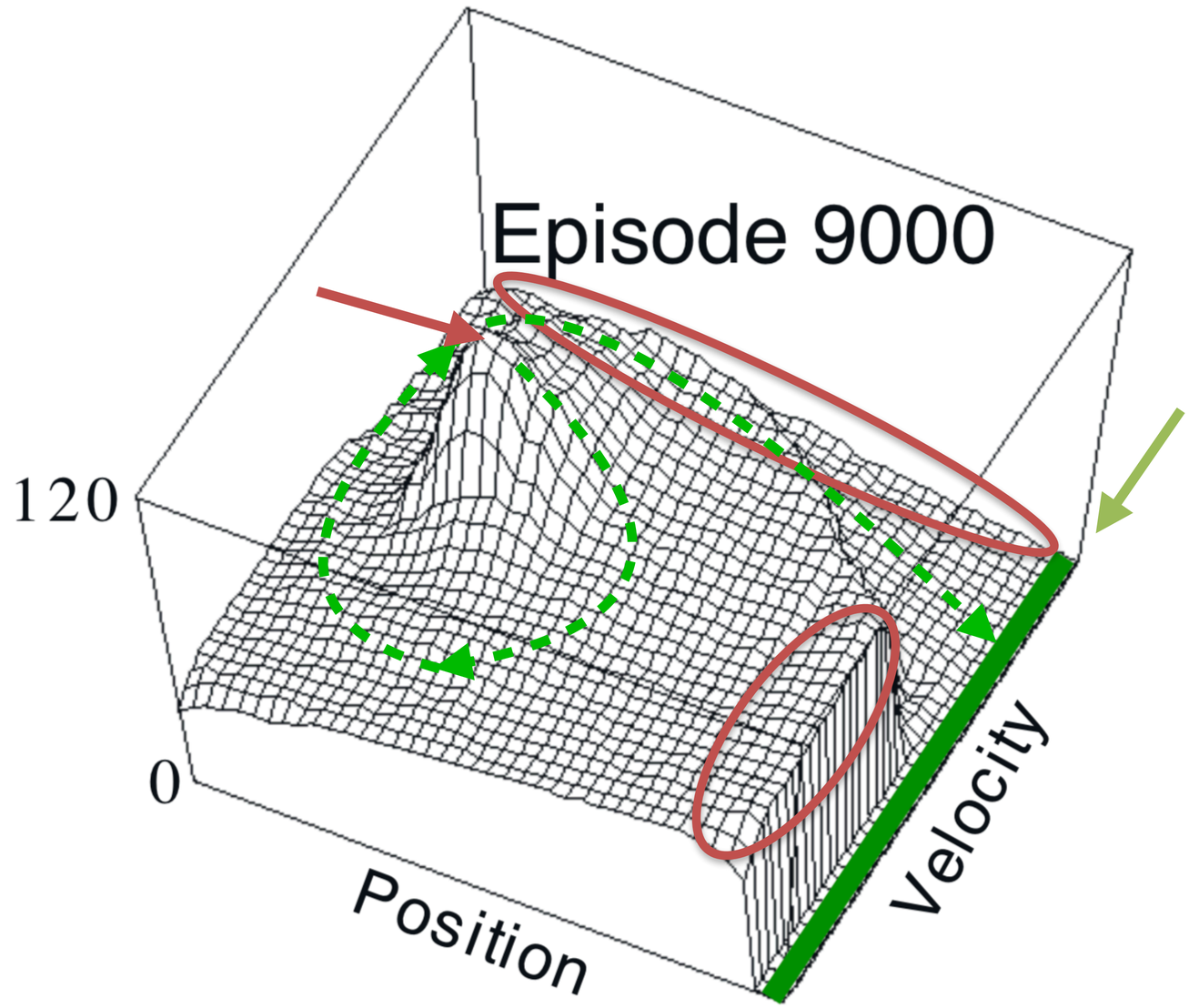
Learning curves



- In Episodic Sarsa in the Mountain car environment, is step size with 0.5/8 the best step size?

Learned values

– $\max_a Q(s, a, \mathbf{w})$



Be a good RL Scientist!

- Notice, even for this tiny problem we tried different alpha. We did **many runs**. We studied learning **speed**; **final performance**; even the value function
    - we have a good idea of how Sarsa works on this problem. It's robust and stable and pretty easy to tune it's parameters
- We want to do such careful analysis every time! Especially when comparing algorithms!
- ML and AI are growing! Lots of people want jobs
- One way to stand out, is to become a really careful empiricist! A master of **good experiments.** It's a rare skill

Video 3: Expected Sarsa with Function Approximation

- If we can do Semi-gradient Sarsa, then its just **small changes** to make Semi-gradient **Expected Sarsa** and Semi-gradient **Q-learning**!
- Goals:
  - Explain the update for Expected Sarsa with function approximation
  - And explain the update for Q-learning with function approximation

- In the tabular case we decided we might prefer ESarsa over Sarsa and Q-learning, and sometimes Sarsa over Q-learning. What about with function approximation?

Self-test

- How do you turn the expected sarsa algorithm to an update of Q-Learning?

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        If S' is terminal:
            $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
            Go to next episode
        Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
        $S \leftarrow S'$
        $A \leftarrow A'$

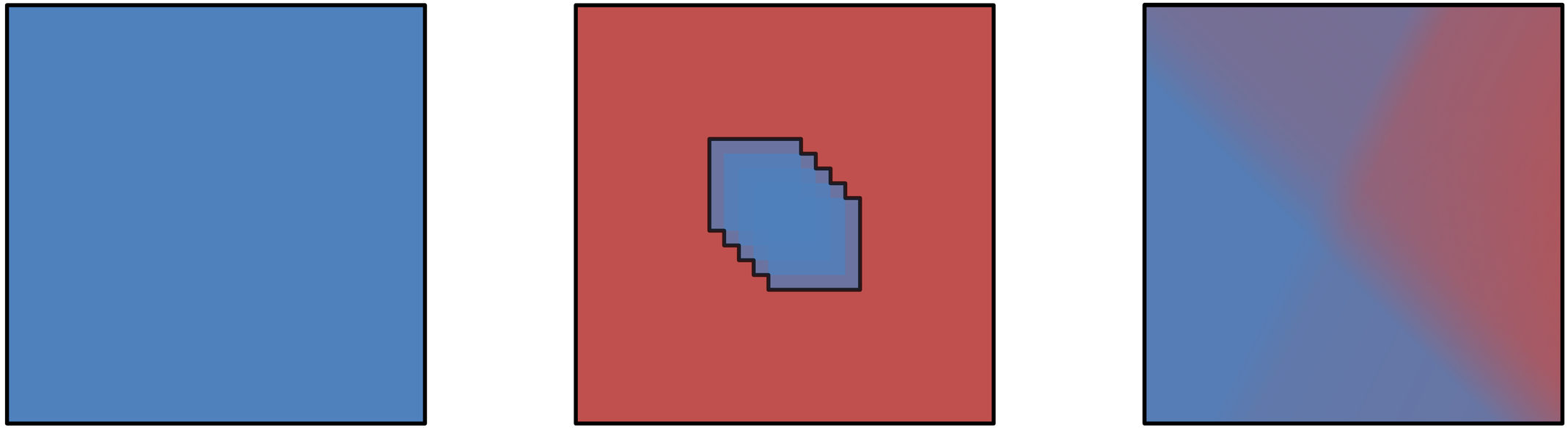
Video 4: Exploration under Function Approximation

- Balancing exploration and exploitation in RL is **hard, even in the tabular case**. Recall that some of the ideas from the Bandit problem could not be easily translated into the tabular RL problem. It is even harder in function approximation. Counting state visits? How do we do optimistic initial values with a tile coder or a NN?

- Goals:

  - Describe how **optimistic initial values** and **epsilon-greedy** can be used with function approximation.

How Optimism Interacts with Generalization



Single feature **Tile coding** **Neural network**

- Do we always want local generalization (like tile coding)?

Video 5: Average Reward: A New Way of Formulating Control Problems

- In some situations **discounting might not be the best choice**. For example, in **continuing tasks with function approximation**. Let's consider another way to formulate the RL task: average reward!

- Goals:

  - Describe the average reward setting

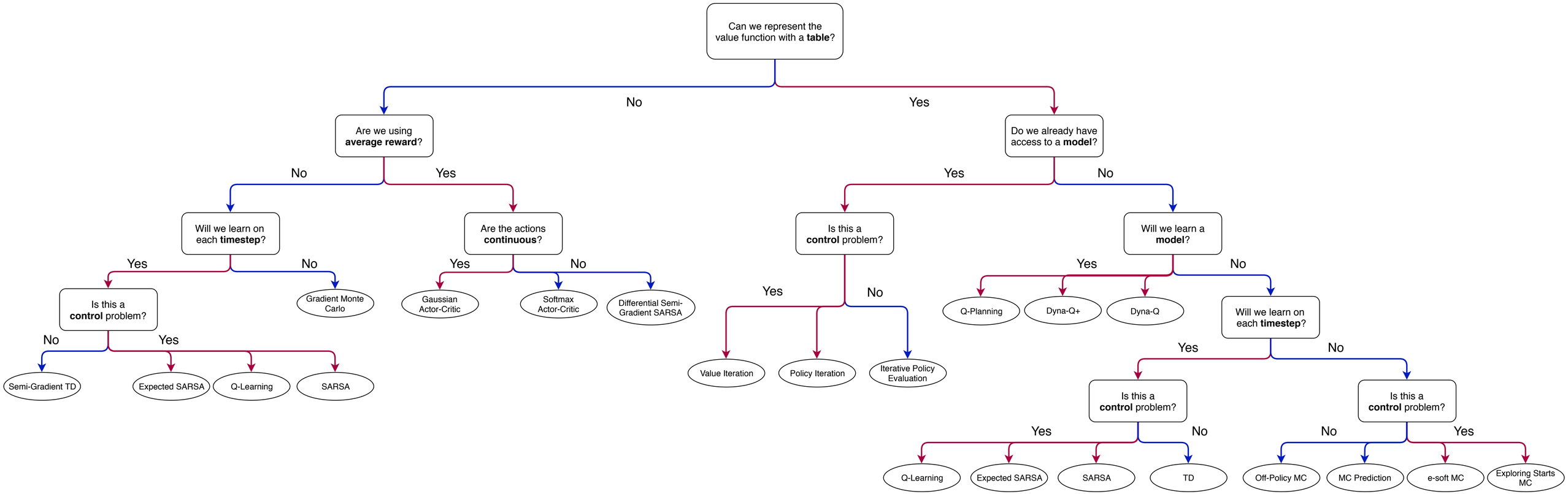
  - Explain when average reward optimal policies are different from policies obtained under discounting

  - And understand differential value functions.

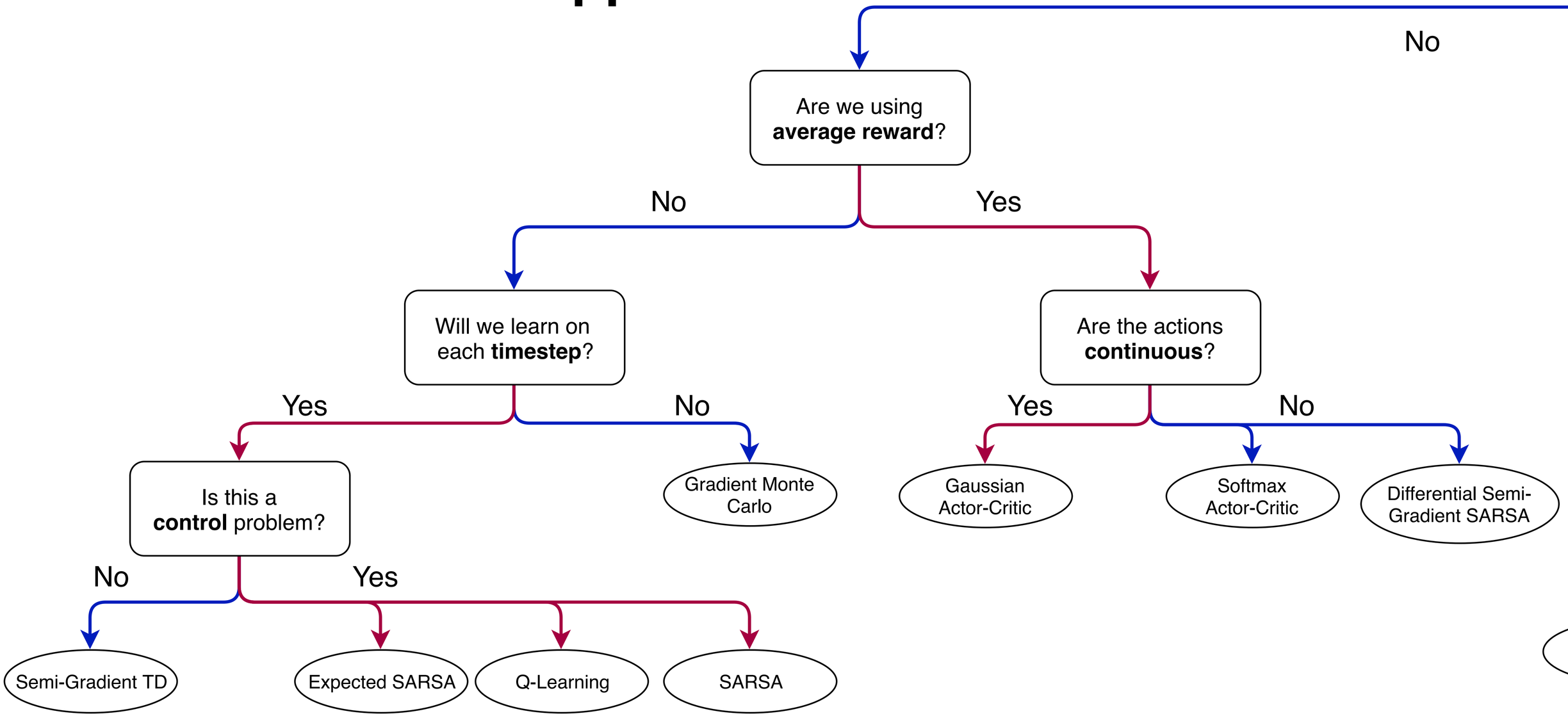
- What is an important hyper-parameter in average reward algorithms analogous to gamma?

Where are we?

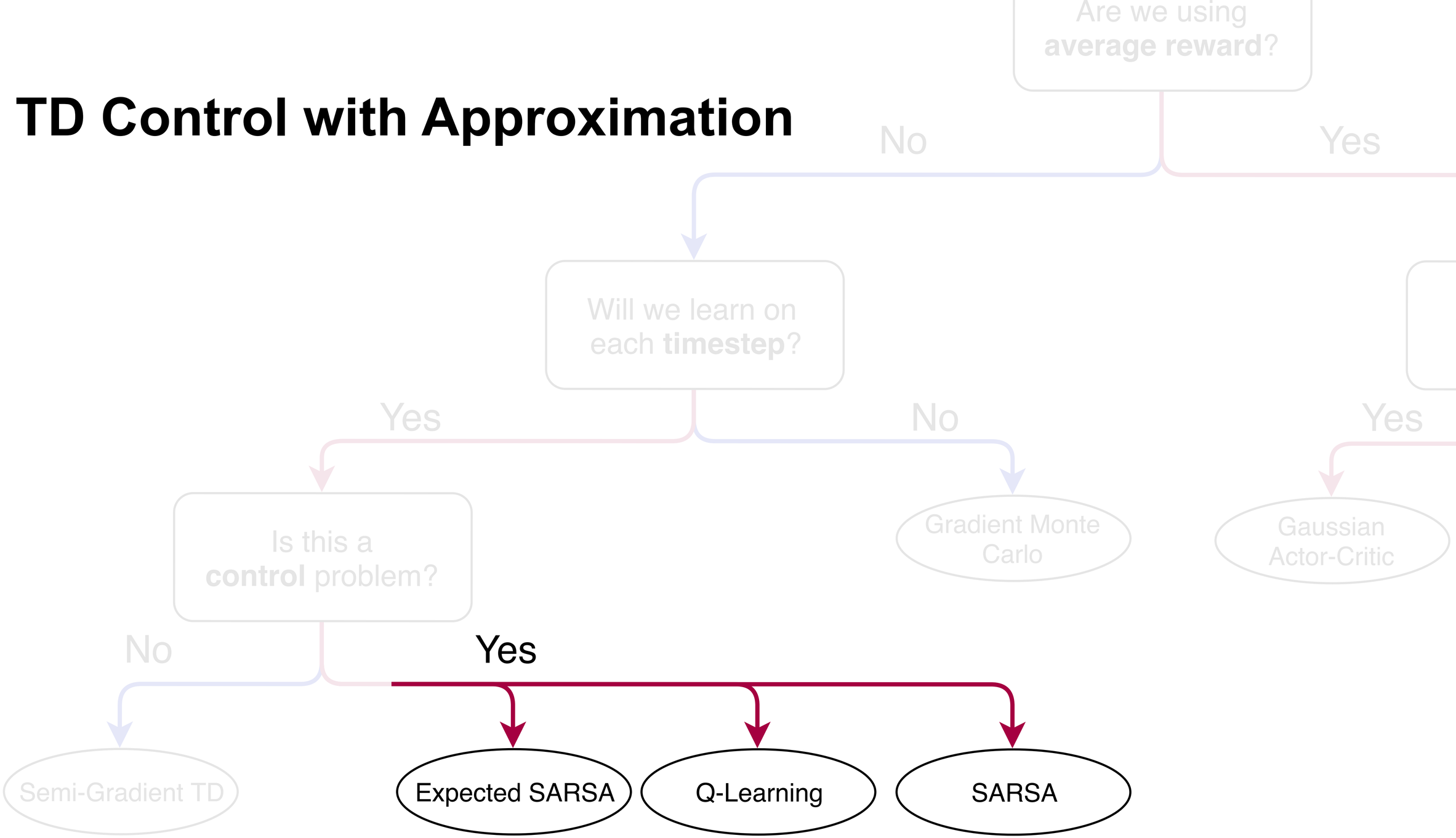
TD Control with Approximation



TD Control with Approximation



TD Control with Approximation



Problem Specification

- Is there any reference about how to design a reward function? For example, we know in Go the reward could be 1 for win, -1 for loss, 0 for tie. How about a universal or general environment?
  - AKA Where do the rewards come from?????
- Since the average reward and differential return method avoid one of the key issues with discounting, is there any reason that we would want to use discounted rewards over differential rewards?

On the futility of discounting

- Setting: continuing problems AND control AND function approximation
- If we optimized discounted value over the on-policy distribution, then the effect would be identical to optimizing undiscounted average reward; the actual value of would have no effect.
- The discounting parameter changes from a problem parameter to a solution method parameter!

Exploration

- Couldn't we do some "dummy" training to initialize an NN with optimistic values? For instance, run through a sufficient number of episodes with high step sizes and an artificially high reward?