



# Reinforcement Learning

## On-policy Control with Approximation

**U Kang**  
**Seoul National University**



# In This Lecture

- On-policy control with function approximation
- Episodic setting
- Continuous setting



# Overview

- Goal: on-policy control with parametric approximation of the action-value function  $\hat{q}(s, a, w) \approx q_*(s, a)$  where  $w \in R^d$  is a finite-dimensional weight vector
- Semi-gradient Sarsa: the natural extension of semi-gradient TD(0) to action values and to on-policy control
- We also introduce a new “average-reward” formulation of the control problem, with new “differential” value functions



# Outline

- ➔  **Episodic Semi-gradient Control**
- Semi-gradient n-step Sarsa
- Average Reward
- Differential and Semi-gradient n-step Sarsa
- Conclusion



# Episodic Semi-gradient Control

- Episodic semi-gradient one-step Sarsa
  - Extension of the semi-gradient prediction methods to action values
  - We consider the approximate action-value function  $\hat{q} \approx q_\pi$ , that is represented as a functional form with weight vector  $w$
  - Whereas before we considered random training examples of the form  $S_t \rightarrow U_t$ , now we consider examples of the form  $S_t, A_t \rightarrow U_t$
  - The update target  $U_t$  can be any approximation of  $q_\pi(S_t, A_t)$ , including the usual backed-up values such as the full MC return ( $G_t$ ) or any of the n-step Sarsa returns
  - The general gradient-descent update for action-value prediction is
$$w_{t+1} = w_t + \alpha[U_t - \hat{q}(S_t, A_t, w_t)]\nabla\hat{q}(S_t, A_t, w_t)$$
  - E.g., the update for the one-step Sarsa is
$$w_{t+1} = w_t + \alpha[R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, w_t) - \hat{q}(S_t, A_t, w_t)]\nabla\hat{q}(S_t, A_t, w_t)$$
  - For a constant policy, this method converges in the same way that TD(0) does



# Episodic Semi-gradient Control

- To form control methods, we need to couple action-value prediction methods with techniques for policy improvement and action selection
- Suitable techniques applicable to continuous actions, or to actions from large discrete sets, are a topic of ongoing research
- If the action set is discrete and not too large, then we can use the techniques discussed in previous chapters
  - For each possible action  $a$  available in the current state  $S_t$ , we can compute  $\hat{q}(S_t, a, w_t)$  and then find the greedy action  $A_t^* = \operatorname{argmax}_a \hat{q}(S_t, a, w_t)$
  - Policy improvement is then done by changing the estimation policy to a soft approximation of the greedy policy such as the  $\epsilon$ -greedy policy
  - Actions are selected according to this same policy



# Episodic Semi-gradient Control

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

Input: a differentiable action-value function parameterization  $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$

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.,  $\varepsilon$ -greedy)

Loop for each step of episode:

Take action  $A$ , observe  $R, S'$

If  $S'$  is terminal:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R - \hat{q}(S, A, \mathbf{w})] \nabla \hat{q}(S, A, \mathbf{w})$$

Go to next episode

Choose  $A'$  as a function of  $\hat{q}(S', \cdot, \mathbf{w})$  (e.g.,  $\varepsilon$ -greedy)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R + \gamma \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})] \nabla \hat{q}(S, A, \mathbf{w})$$

$S \leftarrow S'$

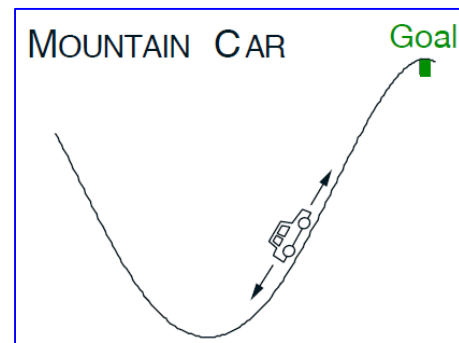
$A \leftarrow A'$

Sutton and Barto,  
Reinforcement  
Learning, 2018



# Example: Mountain Car

- Goal: driving an underpowered car up a steep mountain road
- The difficulty is that gravity is stronger than the car's engine, and even at full throttle the car cannot accelerate up the steep slope
- The only solution is to first move away from the goal and up the opposite slope on the left. Then, by applying full throttle the car can build up enough inertia to carry it up the steep slope even though it is slowing down the whole way
- This is a simple example of a continuous control task where things have to get worse in a sense (farther from the goal) before they can get better
- Many control methodologies have great difficulties with tasks of this kind unless explicitly aided by a human designer



Sutton and Barto,  
Reinforcement  
Learning, 2018





# Example: Mountain Car

- Reward:  $-1$  on all time steps until the car moves past its goal position at the top of the mountain, which ends the episode
- Actions: full throttle forward ( $+1$ ), full throttle reverse ( $-1$ ), and zero throttle ( $0$ )
- The car moves according to a simplified physics. Its position  $x_t$  and velocity  $\dot{x}_t$  are updated by

$$x_{t+1} \doteq \text{bound}[x_t + \dot{x}_{t+1}]$$

$$\dot{x}_{t+1} \doteq \text{bound}[\dot{x}_t + 0.001A_t - 0.0025 \cos(3x_t)]$$

- *bound* operation enforces  $-1.2 \leq x_t \leq 0.5$  and  $-0.07 \leq \dot{x}_{t+1} \leq 0.07$
- When  $x_{t+1}$  reached the left bound,  $\dot{x}_{t+1}$  was reset to zero. When it reached the right bound, the goal was reached and the episode was terminated
- Each episode started from a random position  $x_t \in [-0.6, -0.4)$  and 0 velocity



# Example: Mountain Car

- To convert the two continuous state variables to binary features, we used grid-tilings; we use 8 tilings, with each tile covering 1/8th of the bounded distance in the dimension, and asymmetrical offsets as described
- The feature vectors  $x(s, a)$  created by tile coding were then combined linearly with the parameter vector to approximate the action-value function:

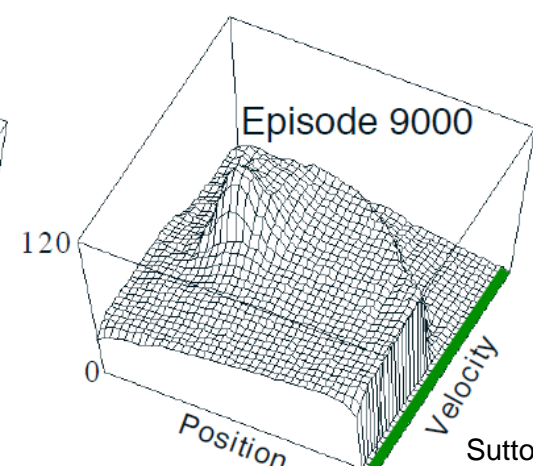
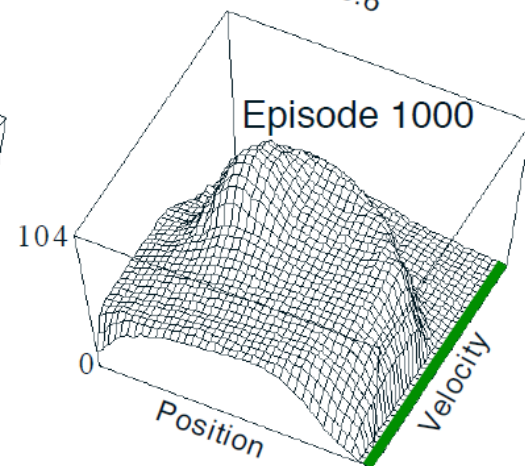
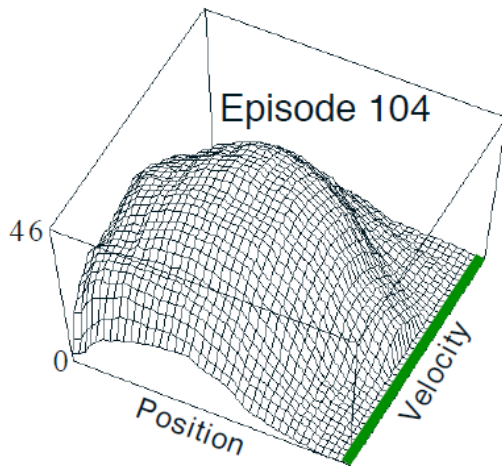
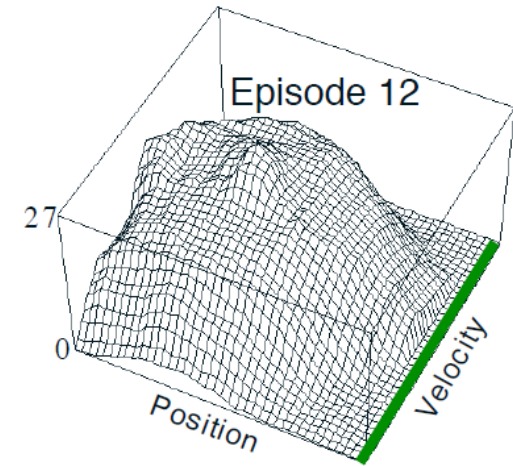
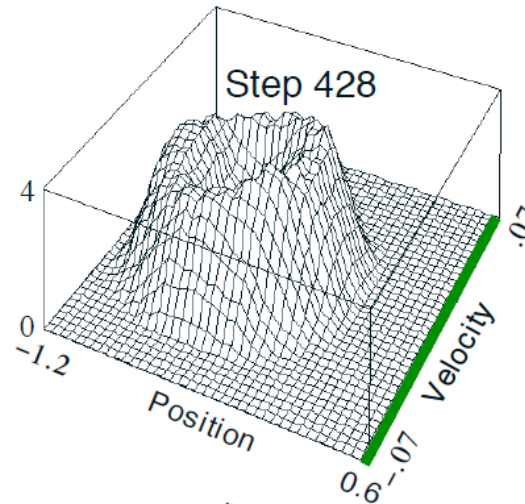
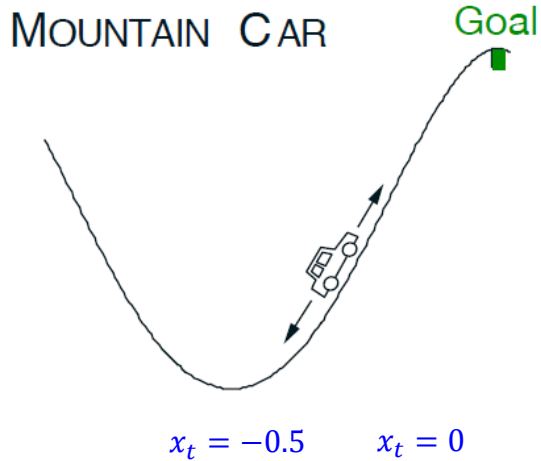
$$\hat{q}(s, a, w) \doteq w^T x(s, a) = \sum_{i=1}^d w_i \cdot x_i(s, a)$$

- for each pair of state  $s$  and action  $a$



# Example: Mountain Car

z axis:  $-1 * (\text{cost-to-go function})$



The optimistic initial action values encourage exploration

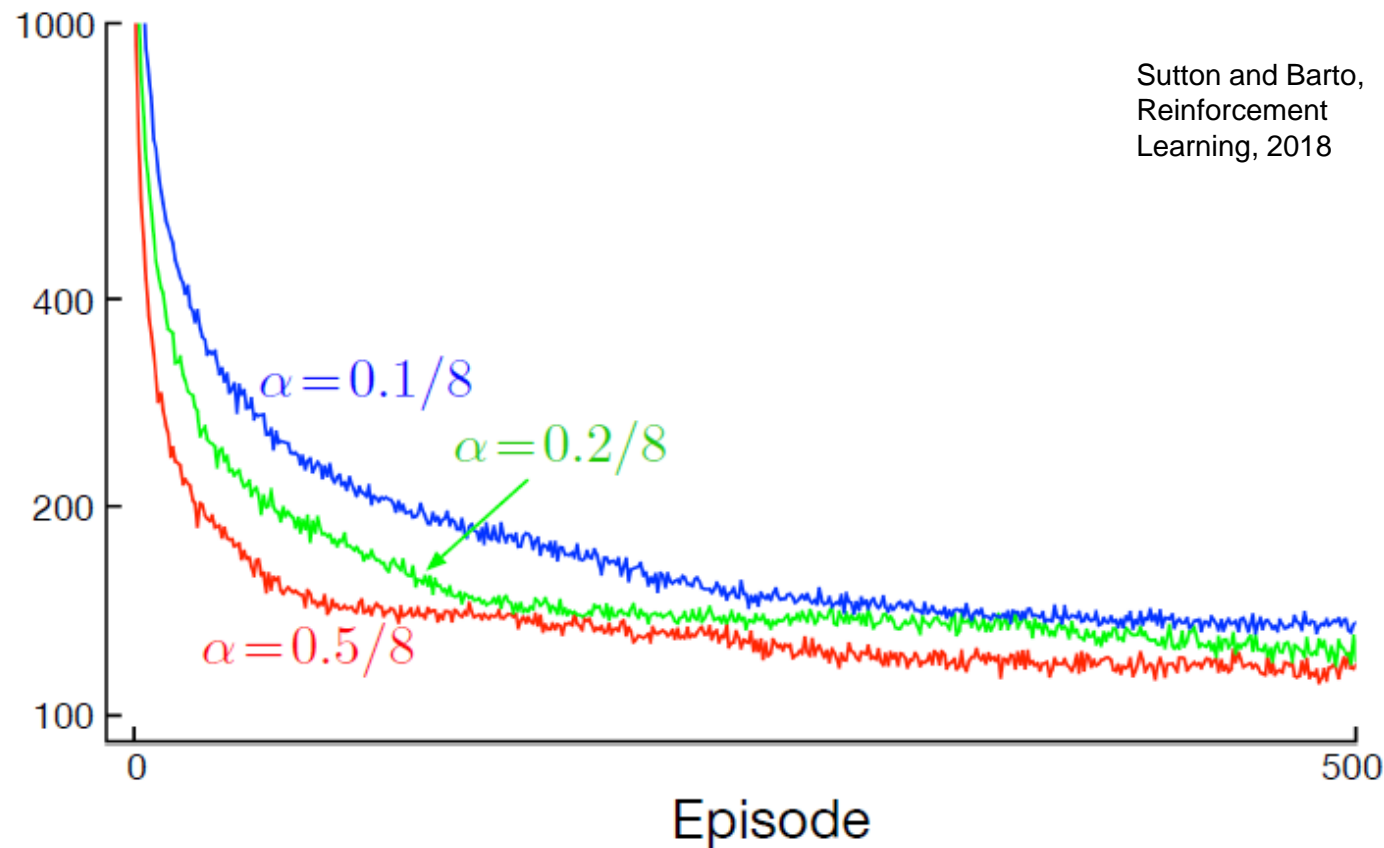
U Kang

Sutton and Barto,  
Reinforcement  
Learning, 2018



# Example: Mountain Car


Mountain Car  
Steps per episode  
log scale  
averaged over 100 runs



Sutton and Barto,  
Reinforcement  
Learning, 2018



# Outline

- Episodic Semi-gradient Control
-   **Semi-gradient n-step Sarsa**
- Average Reward
- Differential and Semi-gradient n-step Sarsa
- Conclusion



# Semi-gradient n-step Sarsa

- We can obtain an n-step version of episodic semi-gradient Sarsa by using an n-step return as the update target in the semi-gradient Sarsa update equation
- The n-step return immediately generalizes from its tabular form to a function approximation form

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}, w_{t+n-1}), \\ t + n < T$$

- with  $G_{t:t+n} = G_t$  if  $t + n \geq T$ , as usual
- The n-step update equation is

$$w_{t+n} \doteq w_{t+n-1} + \alpha [G_{t:t+n} - \hat{q}(S_t, A_t, w_{t+n-1})] \nabla \hat{q}(S_t, A_t, w_{t+n-1}), \\ 0 \leq t < T$$



# Semi-gradient n-step Sarsa

Episodic semi-gradient  $n$ -step Sarsa for estimating  $\hat{q} \approx q_*$  or  $q_\pi$

Input: a differentiable action-value function parameterization  $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$

Input: a policy  $\pi$  (if estimating  $q_\pi$ )

Algorithm parameters: step size  $\alpha > 0$ , small  $\varepsilon > 0$ , a positive integer  $n$

Initialize value-function weights  $\mathbf{w} \in \mathbb{R}^d$  arbitrarily (e.g.,  $\mathbf{w} = \mathbf{0}$ )

All store and access operations ( $S_t$ ,  $A_t$ , and  $R_t$ ) can take their index mod  $n + 1$

Loop for each episode:

    Initialize and store  $S_0 \neq$  terminal

    Select and store an action  $A_0 \sim \pi(\cdot | S_0)$  or  $\varepsilon$ -greedy wrt  $\hat{q}(S_0, \cdot, \mathbf{w})$

$T \leftarrow \infty$

    Loop for  $t = 0, 1, 2, \dots$  :

        | If  $t < T$ , then:

          | Take action  $A_t$

          | Observe and store the next reward as  $R_{t+1}$  and the next state as  $S_{t+1}$

          | If  $S_{t+1}$  is terminal, then:

          |      $T \leftarrow t + 1$

          | else:

          |     Select and store  $A_{t+1} \sim \pi(\cdot | S_{t+1})$  or  $\varepsilon$ -greedy wrt  $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$

          |      $\tau \leftarrow t - n + 1$  ( $\tau$  is the time whose estimate is being updated)

          |     If  $\tau \geq 0$ :

          |          $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$

          |         If  $\tau + n < T$ , then  $G \leftarrow G + \gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w})$

( $G_{\tau:\tau+n}$ )

          |          $\mathbf{w} \leftarrow \mathbf{w} + \alpha [G - \hat{q}(S_\tau, A_\tau, \mathbf{w})] \nabla \hat{q}(S_\tau, A_\tau, \mathbf{w})$

    Until  $\tau = T - 1$

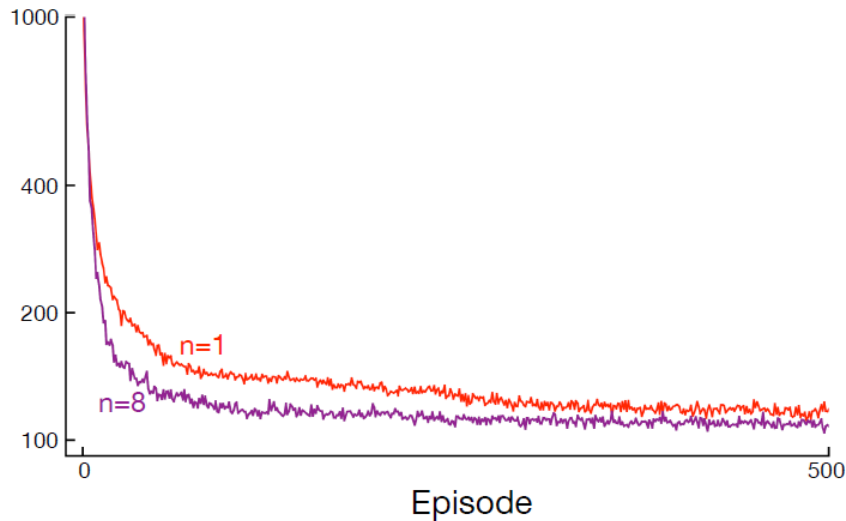


# Semi-gradient n-step Sarsa

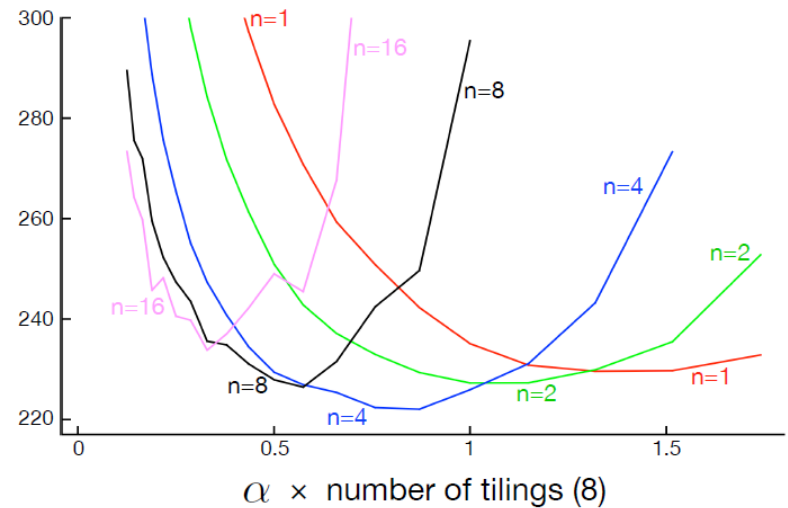
- The performance is best if an intermediate level of bootstrapping is used, corresponding to an  $n$  larger than 1

Sutton and Barto,  
Reinforcement  
Learning, 2018

Mountain Car  
Steps per episode  
log scale  
averaged over 100 runs



Mountain Car  
Steps per episode  
averaged over  
first 50 episodes  
and 100 runs







# Outline

Episodic Semi-gradient Control

Semi-gradient n-step Sarsa

  **Average Reward**

Differential and Semi-gradient n-step Sarsa

Conclusion



# Average Reward: A New Problem Setting for Continuing Tasks

- We now introduce a third classical setting—alongside the episodic and discounted settings—for formulating the goal in MDP
- Like the discounted setting, the *average reward setting* applies to *continuing* problems where the interaction between agent and environment goes on and on forever
- Unlike that setting, however, there is no discounting—the agent cares just as much about delayed rewards as it does about immediate reward
- Instead of discounting, we subtract the average reward from each reward to avoid the return becoming too large



# Average Reward: A New Problem Setting for Continuing Tasks

- In the average-reward setting, we define the average reward  $r(\pi)$  of a policy  $\pi$  as follows:

$$\begin{aligned} r(\pi) &\doteq \lim_{h \rightarrow \infty} \frac{1}{h} \sum_{t=1}^h \mathbb{E}[R_t | S_0, A_{0:t-1} \sim \pi] \\ &= \lim_{t \rightarrow \infty} \mathbb{E}[R_t | S_0, A_{0:t-1} \sim \pi], \\ &= \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) r \end{aligned}$$

- where the expectations are conditioned on the initial state  $S_0$ , and on the subsequent actions,  $A_0, A_1, \dots, A_{t-1}$ , being taken according to  $\pi$
- $\mu_\pi$  is the steady-state distribution,  $\mu_\pi(s) = \lim_{t \rightarrow \infty} \Pr\{S_t = s | A_{0:t-1} \sim \pi\}$ , which is assumed to exist for any  $\pi$  and to be independent of  $S_0$  (ergodicity: in the long run the expectation of being in a state depends only on the policy and the MDP transition prob.)
- Ergodicity guarantees the existence of the limits in the equations above



# Average Reward: A New Problem Setting for Continuing Tasks

- In the average-reward setting, returns are defined in terms of differences between rewards and the average reward:

$$G_t \doteq R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \dots$$

- This is known as the differential return, and the corresponding value functions are known as differential value functions which are defined in the same way and we will use the same notation:  $v_\pi(s) = E_\pi[G_t | S_t = s]$  and  $q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]$  (similarly for  $v_*$  and  $q_*$ )



# Average Reward: A New Problem Setting for Continuing Tasks

- Differential value functions also have Bellman equations, just slightly different from those we have seen earlier. We simply remove all  $\gamma$ s and replace all rewards by the difference between the reward and the true average reward:

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

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

$$v_*(s) = \max_a \sum_{r,s'} p(s', r|s, a) \left[ r - \max_{\pi} r(\pi) + v_*(s') \right], \text{ and}$$

$$q_*(s, a) = \sum_{r,s'} p(s', r|s, a) \left[ r - \max_{\pi} r(\pi) + \max_{a'} q_*(s', a') \right]$$



# Average Reward: A New Problem Setting for Continuing Tasks

- There is also a differential form of the two TD errors:

$$\begin{aligned}\delta_t &\doteq R_{t+1} - \bar{R}_t + \hat{v}(S_{t+1}, w_t) - \hat{v}(S_t, w_t), \\ \delta_t &\doteq R_{t+1} - \bar{R}_t + \hat{q}(S_{t+1}, A_{t+1}, w_t) - \hat{q}(S_t, A_t, w_t)\end{aligned}$$

- where  $\bar{R}_t$  is an estimate at time t of the average reward  $r(\pi)$
- With these alternate definitions, most of our algorithms and many theoretical results carry through to the average-reward setting without change
- E.g., the average reward version of semi-gradient Sarsa is defined similarly, except with the differential version of the TD error

$$w_{t+1} \doteq w_t + \alpha \delta_t \nabla \hat{q}(S_t, A_t, w_t)$$



# Average Reward: A New Problem Setting for Continuing Tasks

Differential semi-gradient Sarsa for estimating  $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization  $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$

Algorithm parameters: step sizes  $\alpha, \beta > 0$

Initialize value-function weights  $\mathbf{w} \in \mathbb{R}^d$  arbitrarily (e.g.,  $\mathbf{w} = \mathbf{0}$ )

Initialize average reward estimate  $\bar{R} \in \mathbb{R}$  arbitrarily (e.g.,  $\bar{R} = 0$ )

Initialize state  $S$ , and action  $A$

Loop for each step:

Take action  $A$ , observe  $R, S'$

Choose  $A'$  as a function of  $\hat{q}(S', \cdot, \mathbf{w})$  (e.g.,  $\varepsilon$ -greedy)

$\delta \leftarrow R - \bar{R} + \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})$

$\bar{R} \leftarrow \bar{R} + \beta \delta$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \nabla \hat{q}(S, A, \mathbf{w})$

$S \leftarrow S'$

$A \leftarrow A'$

Sutton and Barto,  
Reinforcement  
Learning, 2018



# Example: An Access-Control Queuing Task

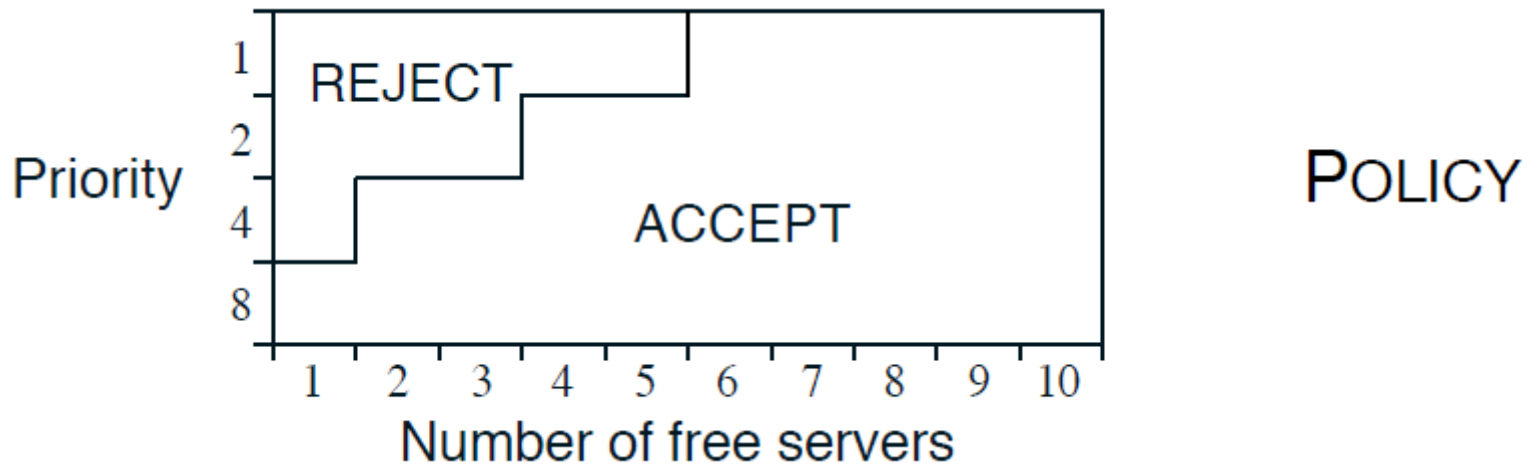
- This is a decision task involving access control to a set of 10 servers
- Customers arrive at a single queue. If given access to a server, the customers pay a reward of 1, 2, 4, or 8 to the server, depending on their priority, with higher-priority customers paying more
- In each time step, the customer at the head of the queue is either accepted (assigned to one of the servers) or rejected (removed from the queue, with a reward of zero). In either case, on the next time step the next customer in the queue is considered
- The queue never empties, and the priorities of the customers in the queue are equally randomly distributed
- Each busy server becomes free with probability  $p = 0.06$  on each time step
- The task is to decide on each step whether to accept or reject the next customer, on the basis of its priority and the number of free servers, so as to maximize long-term reward without discounting






# Example: An Access-Control Queuing Task

- The solution found by differential semi-gradient Sarsa





# Outline

- Episodic Semi-gradient Control
- Semi-gradient n-step Sarsa
- Average Reward
-   **Differential and Semi-gradient n-step Sarsa**
- Conclusion



# Differential and Semi-gradient n-step Sarsa

- In order to generalize to n-step bootstrapping, we need an n-step version of the TD error
- We begin by generalizing the n-step return to its differential form, with FA:

$$G_{t:t+n} \doteq R_{t+1} - \bar{R}_{t+n-1} + \dots + R_{t+n} - \bar{R}_{t+n-1} + \hat{q}(S_{t+n}, A_{t+n}, w_{t+n-1})$$

- where  $\bar{R}$  is an estimate of  $r(\pi)$ ,  $n \geq 1$ , and  $t + n < T$
- If  $t + n \geq T$ , then we define  $G_{t:t+n} = G_t$
- The n-step TD error is then

$$\delta_t \doteq G_{t:t+n} - \hat{q}(S_t, A_t, w)$$

- after which we can apply our usual semi-gradient Sarsa update



# Differential and Semi-gradient $n$ -step Sarsa

Differential semi-gradient  $n$ -step Sarsa for estimating  $\hat{q} \approx q_\pi$  or  $q_*$

Input: a differentiable function  $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$ , a policy  $\pi$

Initialize value-function weights  $\mathbf{w} \in \mathbb{R}^d$  arbitrarily (e.g.,  $\mathbf{w} = \mathbf{0}$ )

Initialize average-reward estimate  $\bar{R} \in \mathbb{R}$  arbitrarily (e.g.,  $\bar{R} = 0$ )

Algorithm parameters: step size  $\alpha, \beta > 0$ , a positive integer  $n$

All store and access operations ( $S_t$ ,  $A_t$ , and  $R_t$ ) can take their index mod  $n + 1$

Initialize and store  $S_0$  and  $A_0$

Loop for each step,  $t = 0, 1, 2, \dots$  :

Take action  $A_t$

Observe and store the next reward as  $R_{t+1}$  and the next state as  $S_{t+1}$

Select and store an action  $A_{t+1} \sim \pi(\cdot | S_{t+1})$ , or  $\varepsilon$ -greedy wrt  $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$

$\tau \leftarrow t - n + 1$  ( $\tau$  is the time whose estimate is being updated)

If  $\tau \geq 0$ :

$$\delta \leftarrow \sum_{i=\tau+1}^{\tau+n} (R_i - \bar{R}) + \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w}) - \hat{q}(S_\tau, A_\tau, \mathbf{w})$$


$$\bar{R} \leftarrow \bar{R} + \beta \delta$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \nabla \hat{q}(S_\tau, A_\tau, \mathbf{w})$$

Sutton and Barto,  
Reinforcement  
Learning, 2018



# Outline

- Episodic Semi-gradient Control
- Semi-gradient n-step Sarsa
- Average Reward
- Differential and Semi-gradient n-step Sarsa
-   **Conclusion**



# Conclusion

- We have extended the ideas of parameterized FA and semi-gradient descent to control
- The extension is immediate for the episodic case, using action values
- For the continuing case we introduce a whole new problem formulation based on average reward



# Questions?