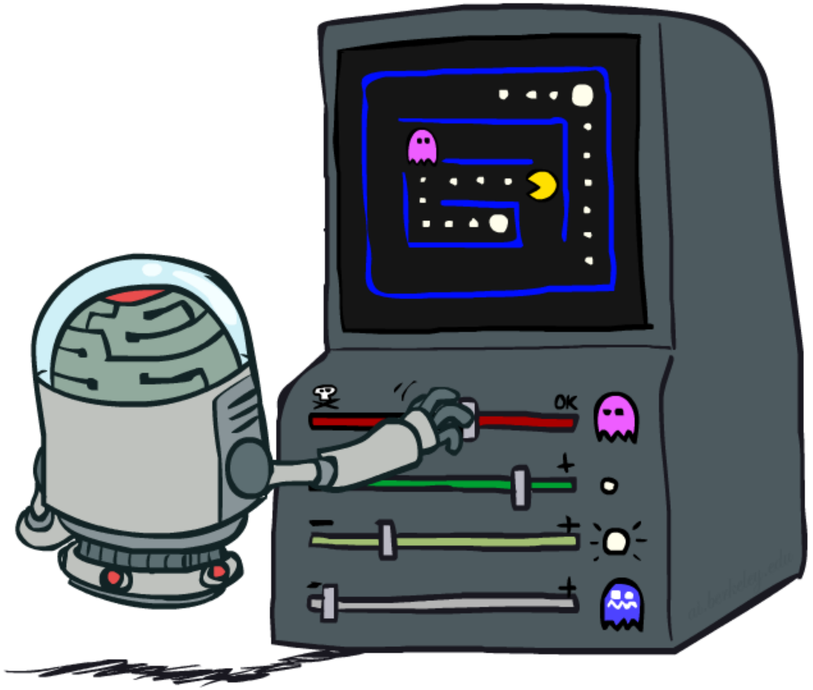
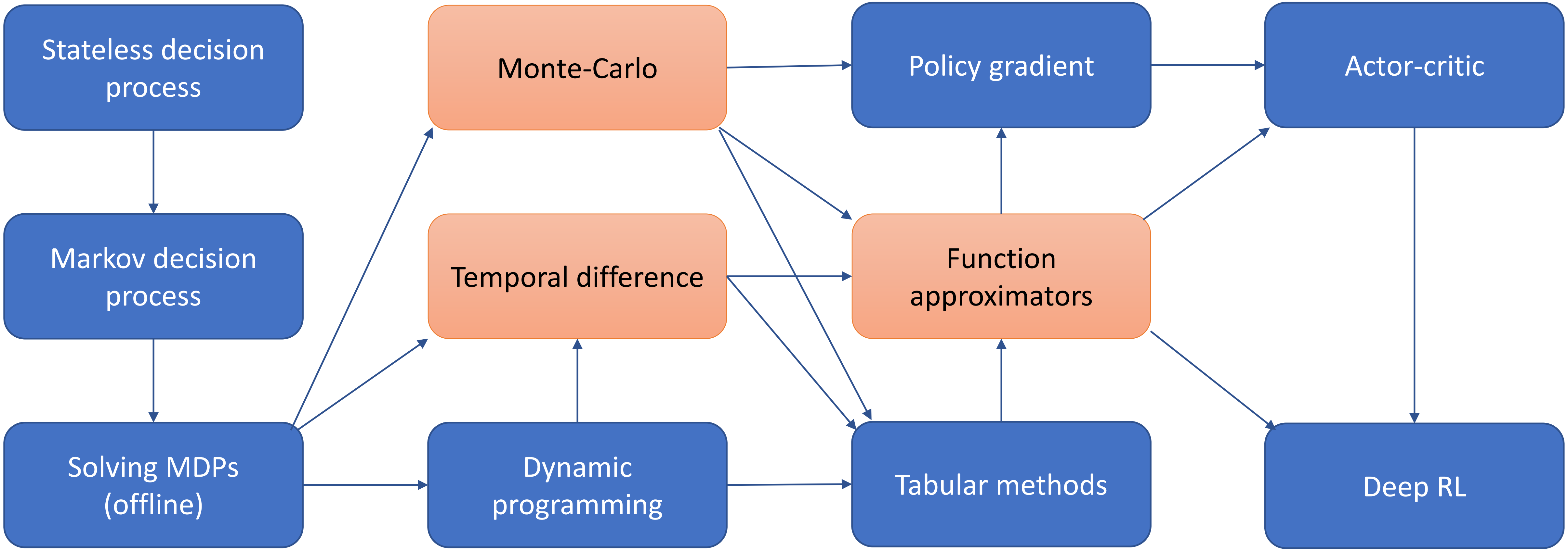
CSCE-642 Reinforcement Learning
Ch10,11: Control with Approximation



Instructor: Guni Sharon

CSCE-689, Reinforcement Learning



Function Approximation in RL

- So far:
  - Approximate state (or action) values for a given policy (on policy)
  - Deep neural nets as function approximators
- Today:
  - Approximate state (or action) values for a given policy then use approximation to adjust the policy
    - On-policy control with approximation
  - Approximate a value function for a target policy, π, while learning and acting on a different policy
    - Off-policy control with approximation

On policy, episodic Semi-gradient control

- Simple extension over the the semi-gradient prediction method from Ch9 (“on-policy prediction with approximation”)
- We want to use an approximation function to derive a policy
  - Approximate Q-values instead of state values
  - Choose actions for state S as a function of $\hat{Q}(S,\cdot;\theta)$ e.g., $\varepsilon - greedy$, softmax

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

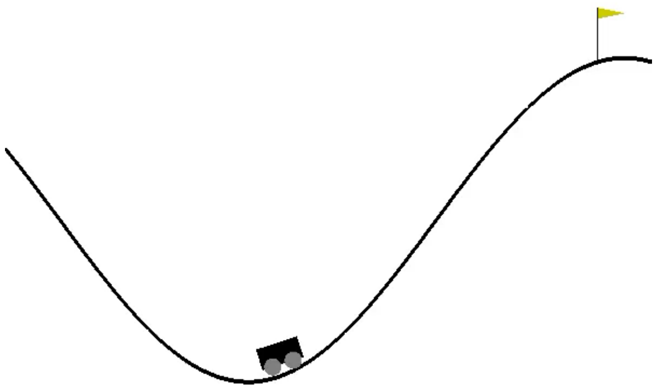
Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
    $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
    Repeat (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)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
        $S \leftarrow S'$
        $A \leftarrow A'$

Gradient assuming squared loss of the TD-error

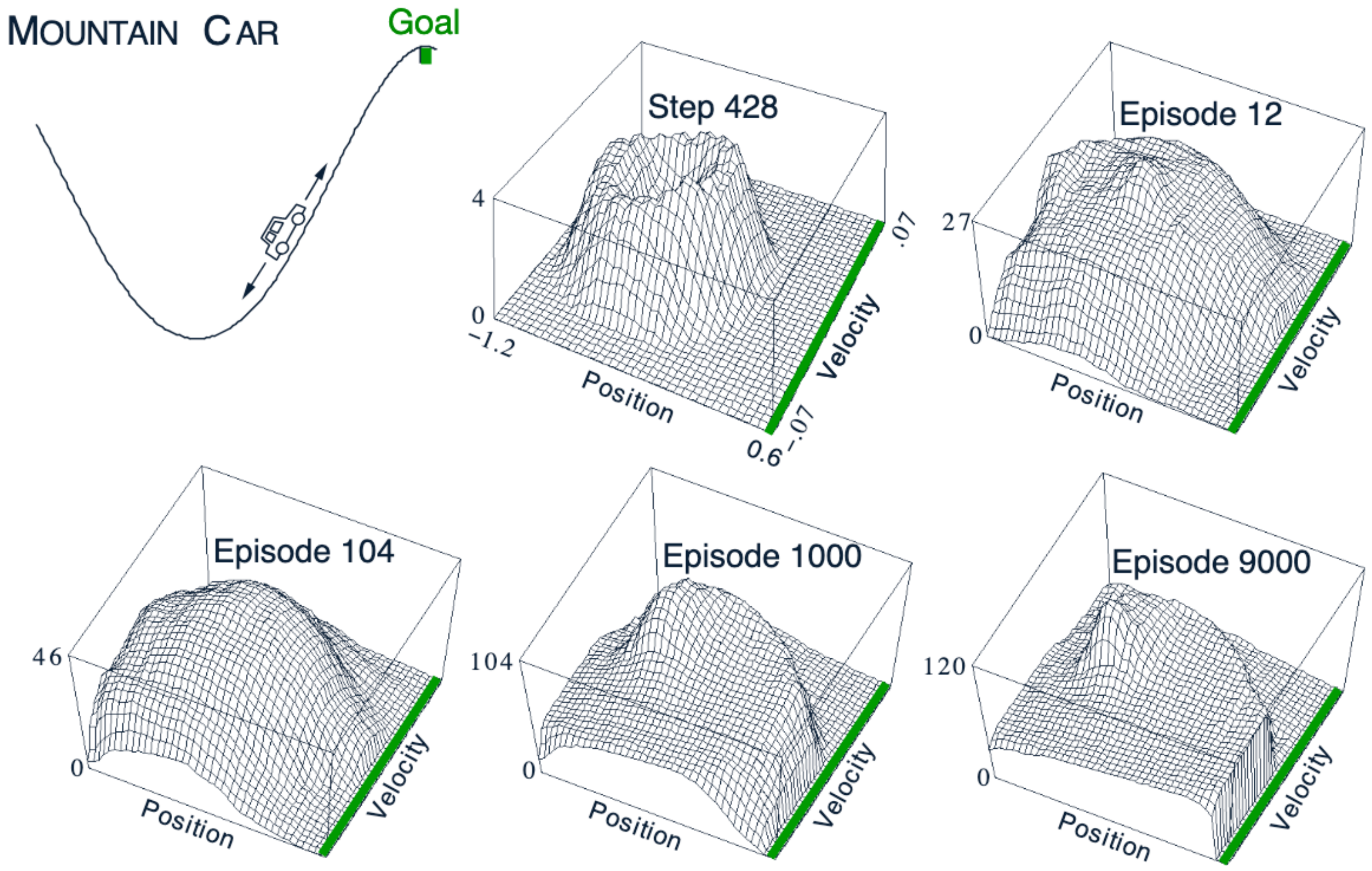
Mountain car example

- Gravity is stronger than the car’s engine
  - Even at full throttle the car cannot accelerate up the steep slope
- Solution: 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
- Things have to get worse in a sense (further from the goal) before they can get better
- Reward: -1 on all time steps until the car moves past its goal position
- Actions: full throttle forward, full throttle reverse, and zero throttle



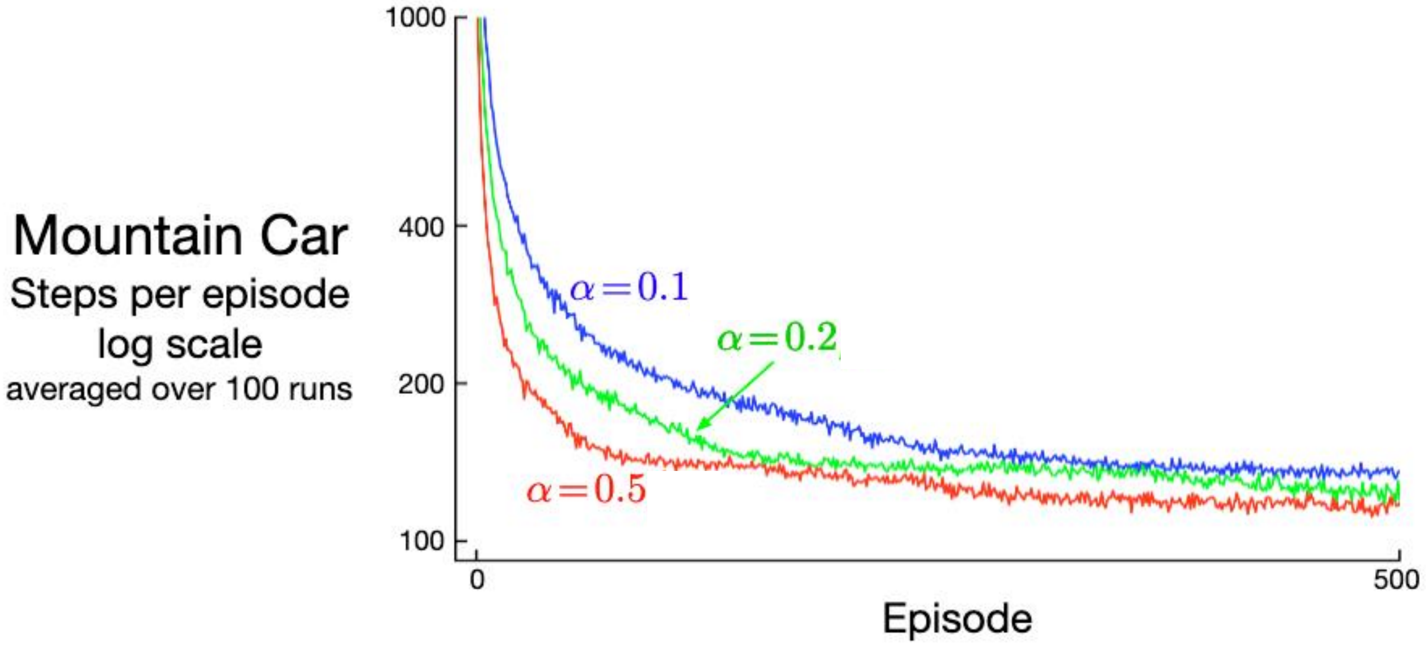
Semi –gradient SARSA with coarse coding approximation

- Plots report negative of the approximated state value
  - $-\max_a \hat{q}(s, a; \theta)$
- Q-values were initialized to 0
- First episode takes many steps to complete
- Throttle actions = {left, right, none}



Semi –gradient SARSA with approximation

- The policy converges faster than the Q-values
- How can we speed up the Q-values propagation?



Semi-gradient n-step SARSA with approximation

- We can use the n-step return as the target value to speed up on-policy Q learning propagation

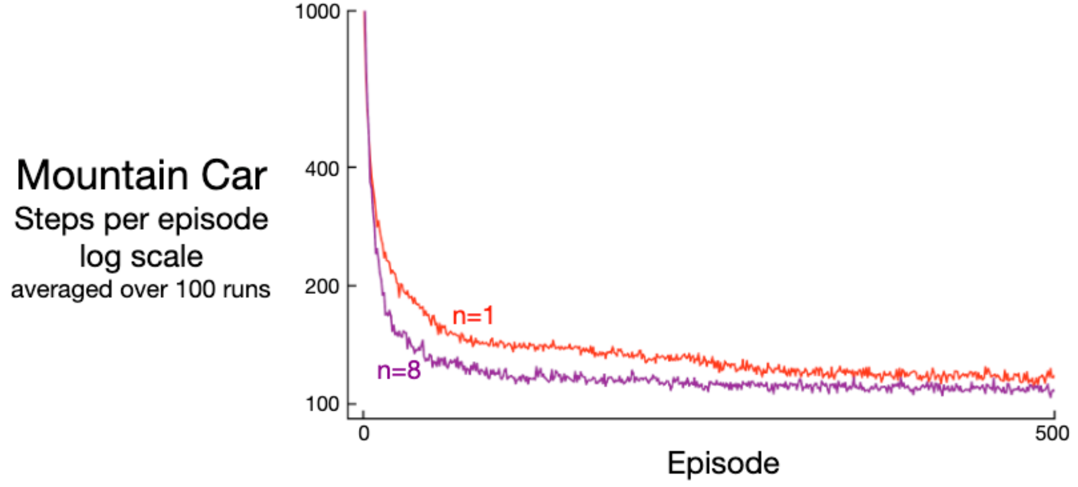
Episodic semi-gradient n-step Sarsa for estimating $\hat{q} \approx q_*$, or $\hat{q} \approx q_\pi$

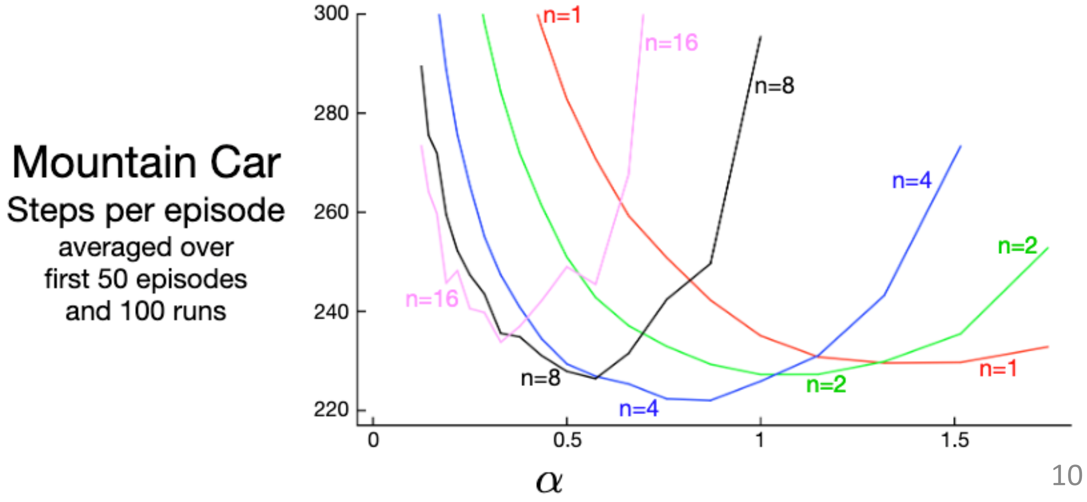
Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, possibly π
Initialize value-function weight vector $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Parameters: step size $\alpha > 0$, small $\varepsilon > 0$, a positive integer n
All store and access operations (S_t, A_t, and R_t) can take their index mod n

Repeat (for each episode):
    Initialize and store $S_0 \neq$ terminal
    Select and store an action $A_0 \sim \pi(\cdot|S_0)$ or ε-greedy wrt $\hat{q}(S_0, \cdot, \mathbf{w})$
    $T \leftarrow \infty$
    For $t = 0, 1, 2, \ldots$:
    | 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 ε-greedy wrt $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$
    | $\tau \leftarrow t - n + 1$ (τ 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 $G \leftarrow G + \gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w})$ $(G_{\tau:\tau+n})$
    | $\boxed{\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[G - \hat{q}(S_\tau, A_\tau, \mathbf{w})\right] \nabla \hat{q}(S_\tau, A_\tau, \mathbf{w})}$
    Until $\tau = T - 1$

n-step SARSA with approximation

- One-step vs multi-step performance of n-step semi-gradient Sarsa on the Mountain Car task
- Effect of α and n on early performance of n-step semi-gradient Sarsa
- An intermediate level of bootstrapping (n= 4) performed best





Off-policy control with approximation

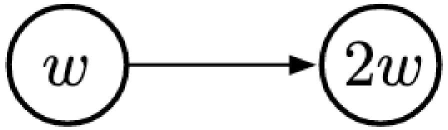
- In off-policy learning we seek to learn a value function for a target policy π, given transitions sampled from a different, behavior policy, b
- In the prediction case, both policies are static and given, and we seek to learn either state values $\hat{v} \approx v_\pi$ or action values $\hat{q} \approx q_\pi$
- In the control case, action values are learned, and both policies typically change during learning
- π being the greedy policy with respect to $\hat{q}$, and b being more exploratory such as the ε-greedy policy with respect to $\hat{q}$

Importance sampling

- Importance sampling (see "Monte-Carlo.pptx" slide 28)
  - $\rho_{t:t} = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
  - $\rho_{t:T} = \prod_{k=t}^{T-1} \frac{\pi(A_K|S_K)}{b(A_K|S_K)}$
- Easily extended to the approximation case
  - **Monte Carlo**: $\hat{q}(S_t, A_t) \leftarrow \hat{q}(S_t, A_t) + \rho_{t:T}[G_b - \hat{q}(S_t, A_t)]\nabla\hat{q}(S_t, A_t; \theta)$
  - **TD(0) + Approx**: $\theta_{t+1} = \theta_t + \alpha\rho_{t:t}\big(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}; \theta) - \hat{q}(S_t, A_t; \theta)\big)\nabla\hat{q}(S_t, A_t; \theta)$
  - **n-step + Approx** : $\theta_{t+n} = \theta_{t+n-1} + \alpha\rho_{t:t+n}\big(G_{t:t+n} - \hat{q}(S_t, A_t; \theta)\big)\nabla\hat{q}(S_t, A_t; \theta_{t+n-1})$

Off-policy divergence

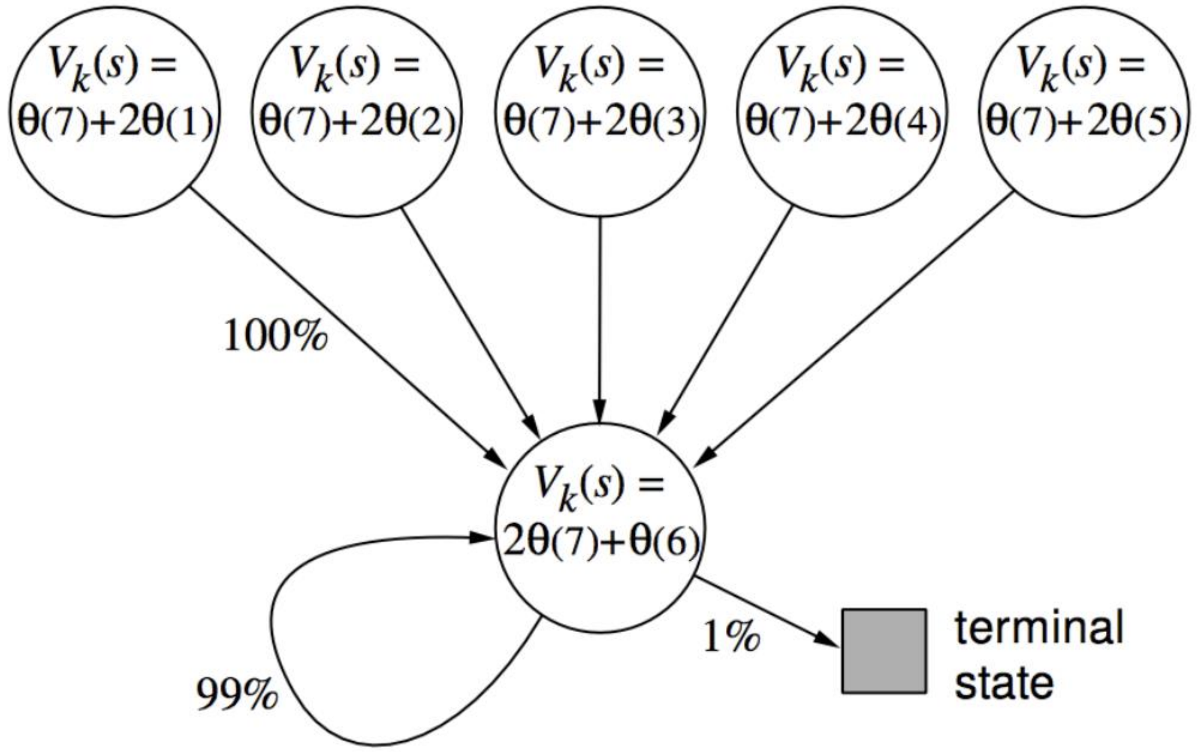
- With function approximation, the distribution of updates does not match the on-policy distribution
  - Updating one state might affect the value of many others
  - Semi-gradient and other simple algorithms are unstable and often diverge
- E.g., consider two states and a value approximator with one tunable parameter



- Updating the value for the left state would also increase the value of the right state, which will increase the value in the left state…

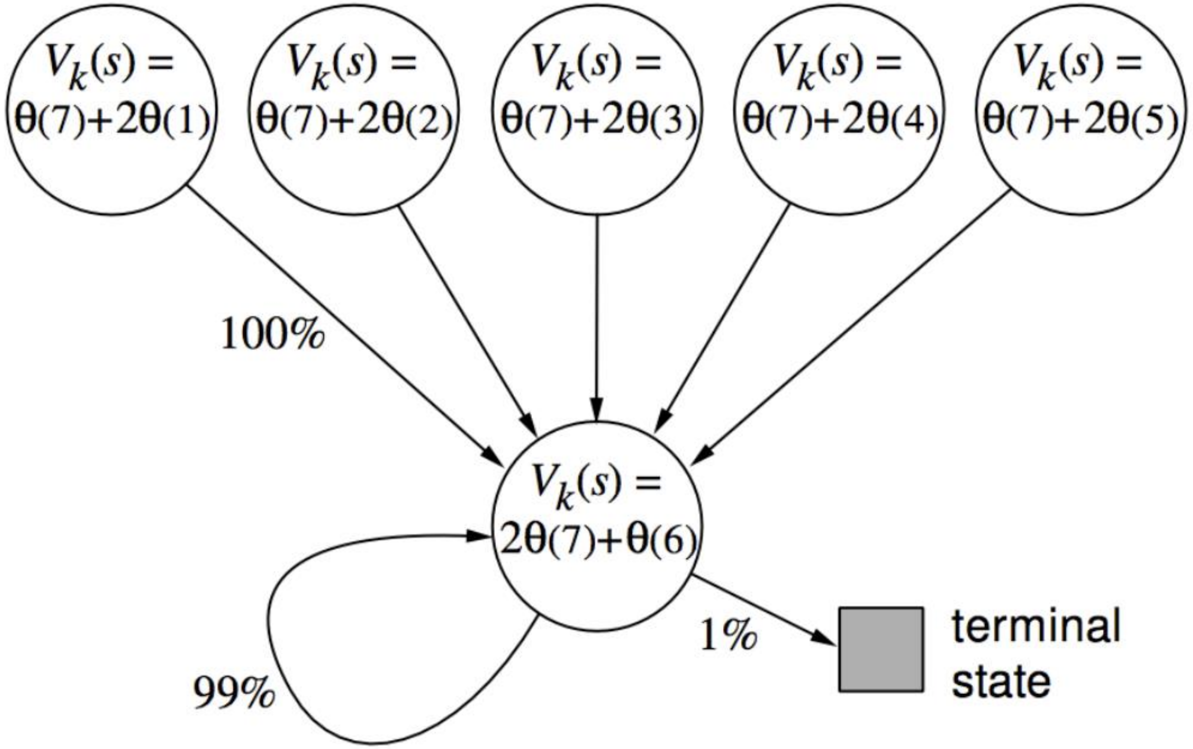
Baird's counterexample

- A scenario where the values of the function approximator won't converge
- Linear value approximator for each state
- E.g., the estimated value of the leftmost state corresponds to the inner product of the tunable parameters vector $[\theta_1, \theta_2, \dots, \theta_7]$ and $[2,0,0,0,0,0,1]$



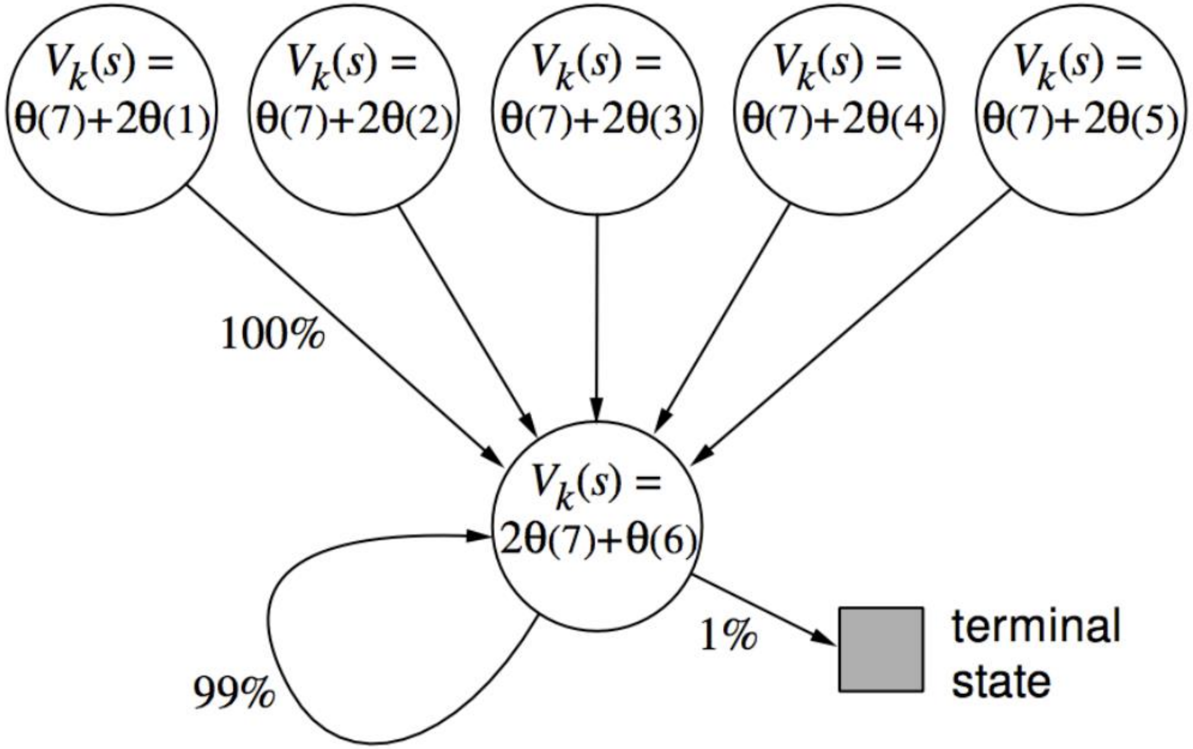
Baird's counterexample

- Reward=0 on all transitions, so $V(s) = 0$, for all s, which can be exactly approximated if, for instance, $\Theta = [0, \dots, 0]$
- However, semi-gradient TD(0) causes weights to diverge to infinity



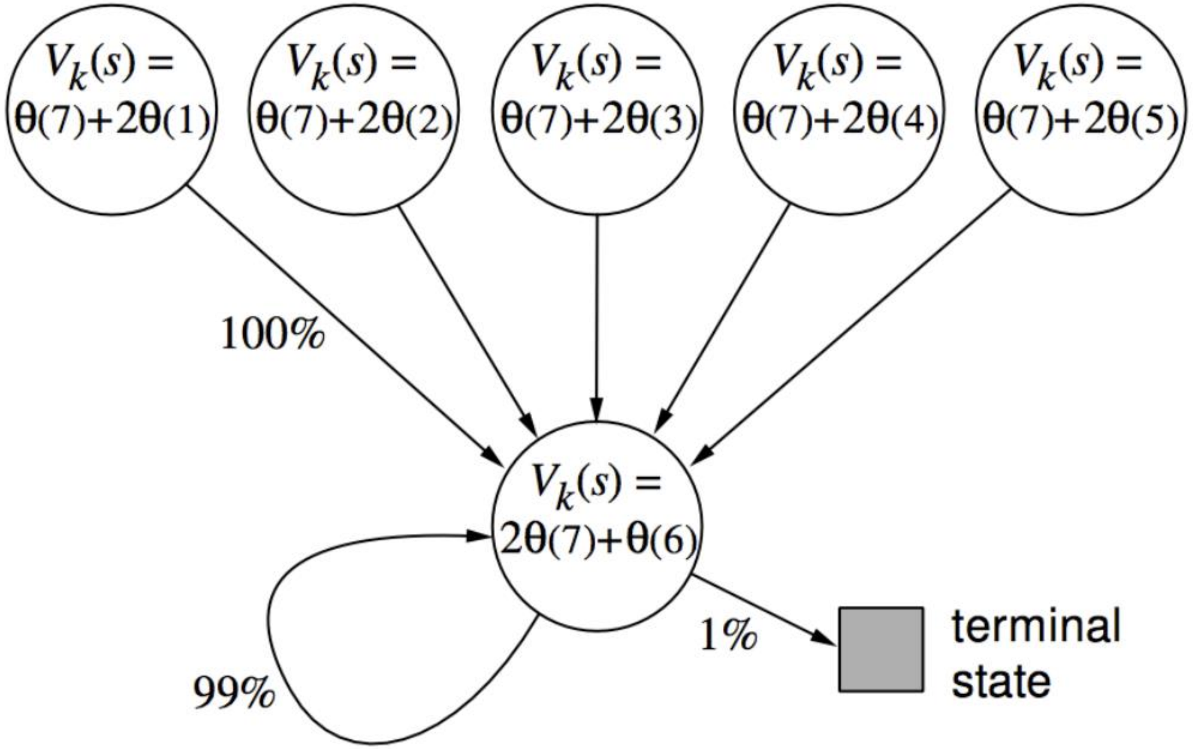
Baird’s counterexample

- Init: $\Theta = (1,1,1,1,1,10,1)$, assume: $\alpha = 0.1, \gamma = 0.99$
- $\hat{V}_0(bottom) = 2 \cdot 1 + 10 = 12$
- $\hat{V}_0(s \neq bottom) = 1 + 2 \cdot 1 = 3$
- $S_0 = left, R_1 = 0, S_1 = bottom,$ T
- $\theta_7 = \theta_7 + \alpha\left(R_1 + \gamma\hat{V}(bottom) - \hat{V}(left)\right)\nabla_{\theta_7}\hat{V}(left)$
- $= 1 + 0.89 \cdot 1 = 1.89$
- $\theta_1 = 1 + 0.89 \cdot 2 = 2.78$

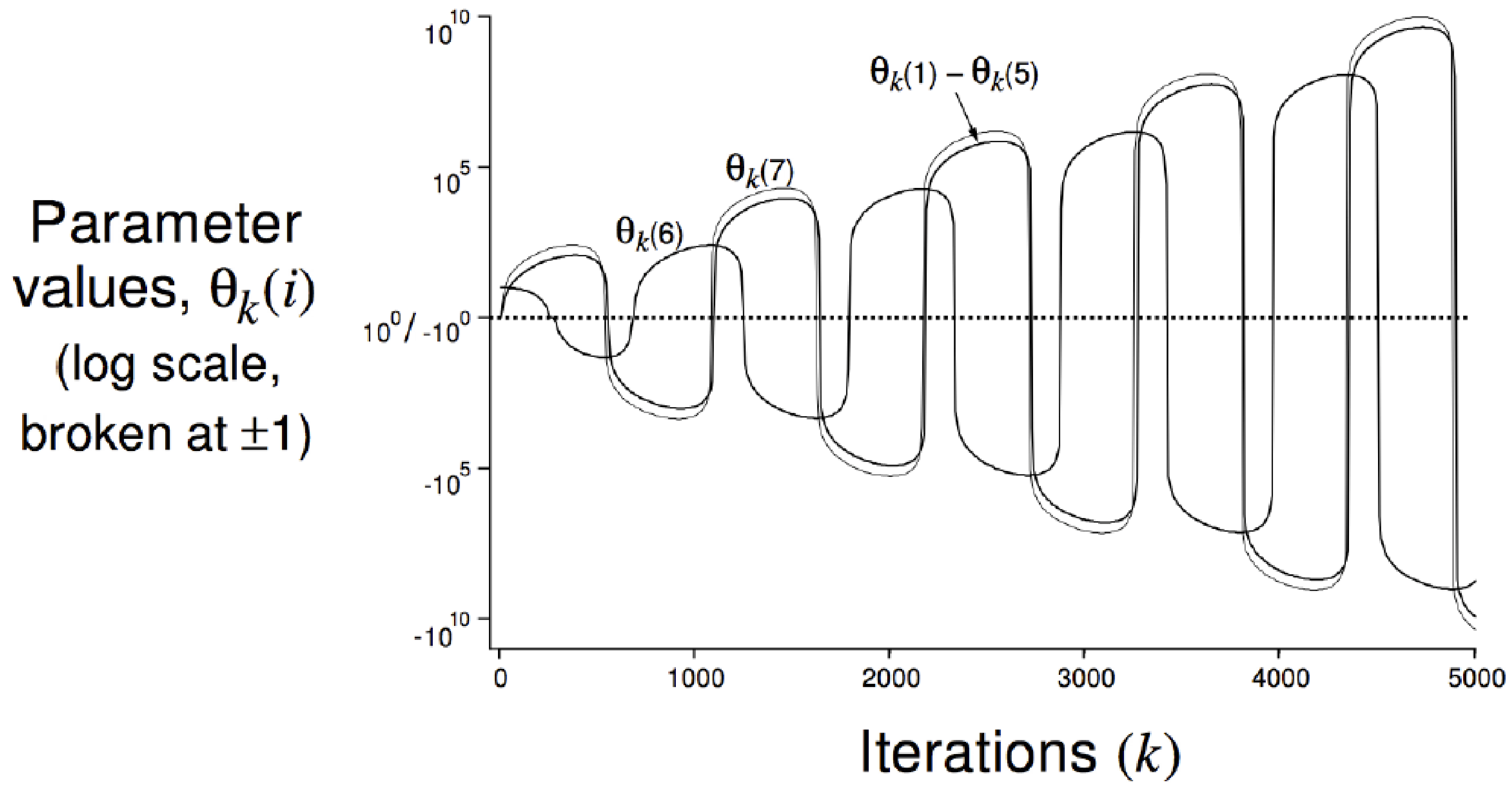


Baird's counterexample

- $\Theta_0 = (1,1,1,1,1,10,1)$
- $\Theta_1 = (2.78,1,1,1,1,10,1.89)$
- $\hat{V}_1(bottom) = 2 \cdot 1.89 + 10 = \cancel{12}\ 13.78$
- $\hat{V}_1(left) = 1.89 + 2 \cdot 2.78 = \cancel{3}\ 7.45$
- $\hat{V}_1(s \neq bottom, left) = \cancel{3}\ 3.89$



Baird's counterexample



Convergence guarantees - prediction

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

For now think of this as n-step TD

Convergence guarantees - control

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗

(✓) chatters around the optimal value function

The deadly triad

- Danger of instability and divergence arises whenever we combine all of the following three elements
- **Function approximation**: A powerful, scalable way of generalizing from a state space much larger than the memory and computational resources
- **Bootstrapping**: Update targets that include existing estimates (as in dynamic programming or TD methods) rather than relying exclusively on actual rewards and complete returns (as in MC methods)
- **Off-policy training**: Training on a distribution of transitions other than that produced by the target policy.

* Divergence can always occur if the learning rate is set too high

Avoiding instability and divergence

Avoid the use of a function approximator

- We need methods that scale to large problems and to great expressive power
- State aggregation or nonparametric methods whose complexity grows with data are too weak or too expensive
- Least-squares methods such as LSTD are of quadratic complexity and are therefore too expensive for large problems

Avoiding instability and divergence

Avoid bootstrapping

- Monte Carlo (non-bootstrapping) methods require storing all transitions leading to the final return and weights are updated only once the final return is obtained
- With bootstrapping, data can be dealt with when and where it is generated, then need never be used again
- The savings in communication and memory made possible by bootstrapping are great
- Bootstrapping often results in faster learning because it allows learning to take advantage of the state property
- Often bootstrapping greatly increases efficiency. It is an ability that we would very much like to keep in our toolkit

Avoiding instability and divergence

Avoid off-policy learning

- On-policy methods are often adequate. For model-free reinforcement learning, one can simply use SARSA rather than Q-learning
- Off-policy methods free behavior from the target policy. This could be considered an appealing convenience but not a necessity
- However, off-policy learning is essential to other anticipated use cases, cases that we have not yet mentioned but are important to the larger goal of creating a powerful intelligent agent
- In these use cases, the agent learns not just a single value function and single policy, but large numbers of them in parallel
- Moreover, in many control problems, the policy (actor) evolves over time. Once the policy changes, all our previous experience becomes off-policy.

Improving the approximation

- Usually, the merits of off-policy bootstrapping with function approximation outweigh the dangers of instability and divergence
- Such methods will be the focus of the rest of this course
  - They are the state-of-the-art
- We will concentrate on avoiding instability and divergence
- Remember that our approximation is updated through interactions with the environment
  - Sample transitions (S, A, R, S')
  - Estimate state or action value
  - Update approximation function towards the new estimation

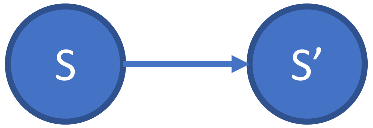
Improving the approximation

- SGD based on sampled TD error in approximated value
  - $\hat{V}(s) = \hat{V}(s) + \alpha\left(sample(V(s)) - \hat{V}(s)\right)\nabla\hat{V}(s)$
  - Stop at fixed point, i.e., $\hat{V}(s) = \mathrm{E}\left[sample\left(V(s)\right)\right]$
- What can go wrong?
  - Bias in sampled value
  - High variance of sampled value
- Bias: $\mathrm{E}\left[sample\left(V(s)\right)\right] \neq V(s)$
  - What's the problem with bias?
- What's the problem with high variance in $sample\left(V(s)\right)$?
  - Noisy gradients, advancing in the wrong direction

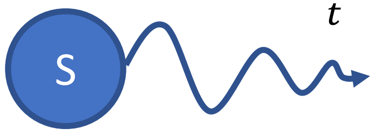
Improving the approximation

- What can go wrong?
  - Bias in sampled value
  - High variance of sampled value
- Consider the sampling rules of TD(0) and MC
- Which approach introduces higher bias and which higher variance?
  - TD(0) adds bias to the sampled $V(s)$ that stems from the error of the approximation function at s'
  - MC suffers no bias yet usually suffers from high variance due to the high variance in the sampled trajectory

$$sampled\ V(s) = R + \hat{V}(S')$$



$$sampled\ V(s) = \sum_t R_t$$



Improving the approximation

- TD(0) adds bias to the estimated $v(s)$ that stems from the error of the approximation function at S'
- MC suffers no bias yet usually suffers from high variance due to the high variance in the sampled trajectory
- Can we rely on future reward (reduce bias) while weighing them according to their probability?
- Far future reward are less likely as a result we should rely less on them and reduce variance
- We will discuss eligibility traces in the next class

What did we learn?

- We can rely on an approximation function for estimating both state/action values and for defining a policy (control with approximation)
- No convergence guarantees (as opposed to the tabular setting)
- Some guarantees are available for on-policy learning with linear approximator
- Once we use bootstrapping approaches for off-policy learning with non-linear approximation no guarantees can be given regarding stability and divergence
- So, should we forgo such methods?
  - No! in practice they work best
  - * requires hyperparameter tuning

What next?

- **Lecture**: Ch 12-Eligibility Traces
- **Assignments**:
  - Tabular Q-Learning, by Oct. 7, EOD
  - SARSA, by Monday Oct. 7, EOD
  - Q-Learning with Approximation, by Oct. 14, EOD
- **Quiz (on Canvas)**:
  - Deep Neural Nets, by Tuesday Oct. 9, EOD
- **Project**:
  - Literature survey, by Monday, November 4 EOD