Reinforcement Learning: Advanced Off-Policy Learning (ESE 3990)

Instructor:
Antonio Loquercio

Fall 2026

University of Pennsylvania

Categorizing RL agents

Axis 1: How to make decisions

- Value Based
  - No Policy (Implicit)
  - Value/Q Function

Today

- Policy Based
  - Policy
  - No Value/Q Function
- Actor Critic
  - Policy
  - Value Function

Axis 2: Environment Representation

- Model-based
  - Policy and/or Value/Q Function
  - Model
- Model-free
  - Policy and/or Value/Q Function
  - No Model

Off-Policy Learning

- You have a lot of data from human drivers.
- You cannot use this data for policy learning.
- But it would be very useful to avoid catastrophic forgetting.



Last Time: Policy Gradients

- Init θ at random
- Run policy $\pi_\theta(a_t|s_t)$ and collect an episode in the environment. Add data in the replay buffer D (can contain other data)
- For a few epochs:
  - Sample a batch of transitions $\{(s_t, a_t, r_t, s_{t'})^k\}$ from the replay buffer
  - For each transition $(s_t, a_t, r_t, s_{t'})^k$
    - Compute the (bootstrapped) return G_t^λ
    - Compute the advantage $A_t^k = G_t^\lambda - \hat{V}(s_t)$
  - Compute the policy gradient estimate: $\hat{g} \approx \frac{1}{K-1}\sum_k \nabla_\theta \log \pi_\theta\left(a_t^k \middle| s_t^k\right) A_t^k$
  - Update policy parameters: $\theta \leftarrow \theta + \alpha\hat{g}$
  - Update the approximation of the value function

Unsatisfying Solution: Still a lot of Variance

- $\nabla_\theta J(\theta^{\text{new}}) =$

$$\mathbb{E}_{\tau\sim p(\tau;\theta^{old})}\left[\sum_{t=0}^{T-1}\left(\rho_{0:t}\nabla_\theta \log\pi(a_t|s_t,\theta^{\text{new}})\left(\sum_{t'=t}^{T-1}\gamma^{t'-t}\rho_{t:t'}r_{t'} - V^\rho(s_t)\right)\right)\right]$$

- Where:

$$V^\rho(s_t) = \rho_{t:T-1}V^{\theta^{old}}(s_t)$$

- We can use other techniques presented in the off-policy learning class

Today's Agenda

- We will take a different route to off-policy learning
- Specifically, we will focus on Q-learning and its variants
- But first, let's start with some bad news.

Q-Learning: Refresher

- Init $Q(s,a) = 0 \; \forall s, a$
- Collect one episode with *some* policy μ, add it to the replay buffer D
- For K iterations:
  - Sample a batch of transitions $\{(s, a, s', r)\}$ from D
  - For every transition: $Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s,a) \right)$

- In contrast to SARSA, the data can be generated by any policy μ.
- The closer is μ to π^*, the faster I learn.
- No importance sampling!

Q-Learning: Refresher

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

- Proven to converge in the *tabular* case (under some assumptions)
- Easy to get a policy once you have a Q function ($\pi = greedy(Q)$)

- Looks like this solves all our problems, right?

Well…

- The state space (e.g., the sequence of images) is continuous.
- The state space is very large
- We need something to *generalize* what we learned in a state to other similar states.



Value Function Approximation (Refresher)

- ***Idea***. For large MDPs, estimate the value function with *function approximation*.

$$\hat{V}(s, \boldsymbol{w}) \approx V^{\pi}(s)$$

$$\hat{Q}(s, a, \boldsymbol{w}) \approx Q^{\pi}(s)$$

- **Goal**. Generalize from seen states to unseen states
  - You can't access all possible states. So you *hope* that it works on unseen ones.

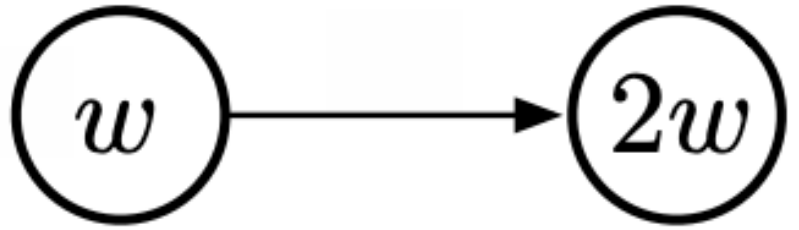
Q-Learning with Function Approximation

- Init $\boldsymbol{w}$
- Collect one episode with *some* policy μ, add it to the replay buffer D
- For K iterations:
  - Sample a batch of transitions $\{(s, a, s', r)\}$ from D
  - For every transition:

$$\Delta w = \alpha \left(r + \gamma \max_{a'} \hat{Q}(s', a', \boldsymbol{w}) - \hat{Q}(s, a, \boldsymbol{w}) \right) \nabla_w \hat{Q}(s, a, \boldsymbol{w})$$

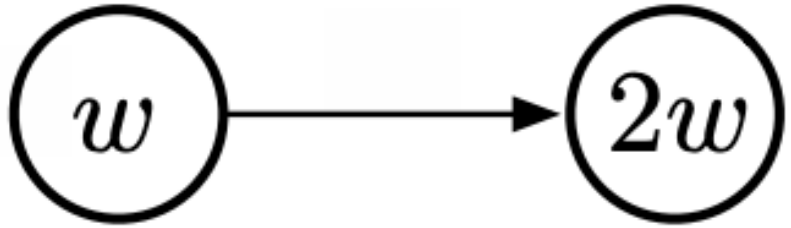
- Seems reasonable, but does it work on a simple example?

An example of divergence



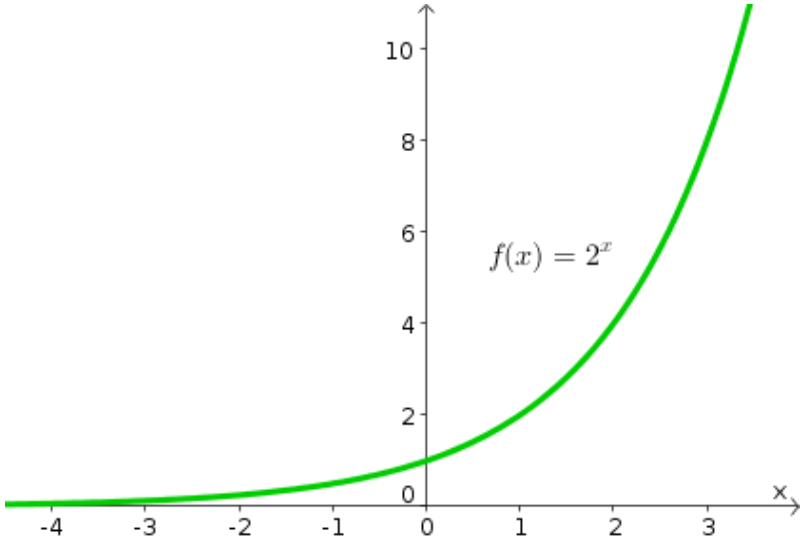
- Only two states. After s_2, deterministically in a terminal state.
- $r(s, a) = 0$
- Initialize with $w = 10$
- Replay buffer contains a transition from a policy μ: $(s_1, a, s_2, 0)$
- Weight update:
- $w_{t+1} = w_t + \alpha \left(r + \gamma \max_{a'} Q(s_2, a', w) - Q(s_1, a, w) \right) \nabla_w Q(s_1, a, w)$
- $w_{t+1} = w_t + \alpha(0 + \gamma 2 w_t - w_t) 1 = \big(1 + \alpha(2\gamma - 1)\big) w_t$

An example of diverge

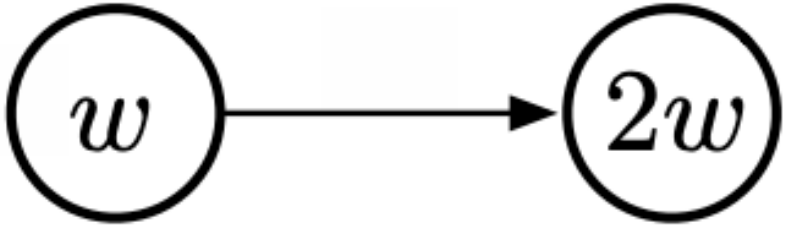


- $w_{t+1} = w_t + \alpha(0 + \gamma 2 w_t - w_t)1 = \big(1 + \alpha(2\gamma - 1)\big)w_t = K w_t$

- If $K > 1$, w_t diverges



What is the problem?

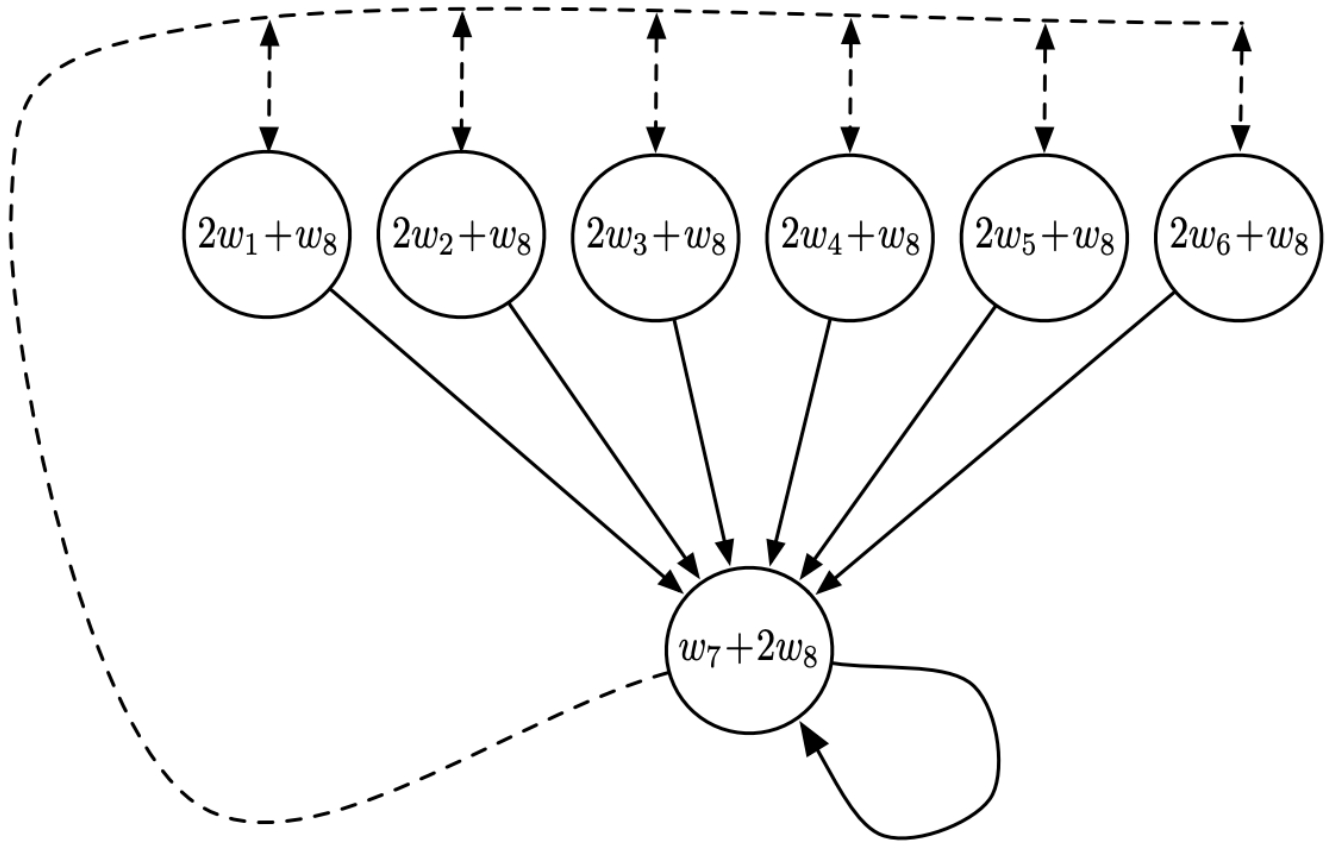


- Replay buffer contains a transition $(s_1, a, s_2, 0)$
- No transitions from s_2 to the terminal state in the buffer.

- While this example is extreme, similar situations can happen when training with off-policy data.

- For instance, when the behavior policy selects an action that the target policy would never select.

A second example of divergence (Baird 95’)



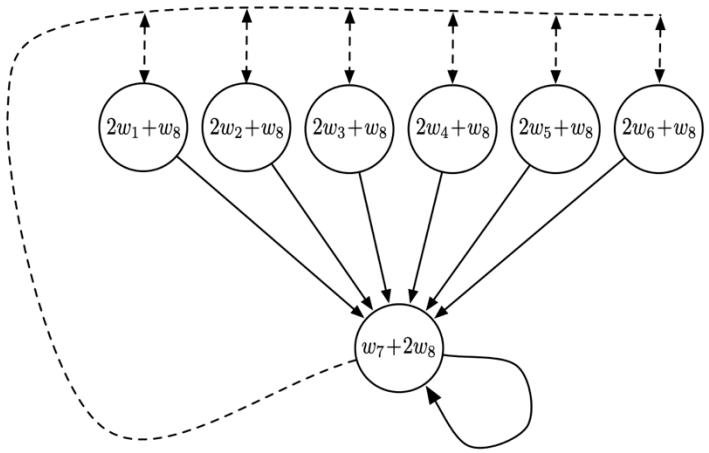
$\pi(\text{solid}|\cdot) = 1$

$b(\text{dashed}|\cdot) = 6/7$

$b(\text{solid}|\cdot) = 1/7$

$\gamma = 0.99$

A second example of divergence (Baird 95')

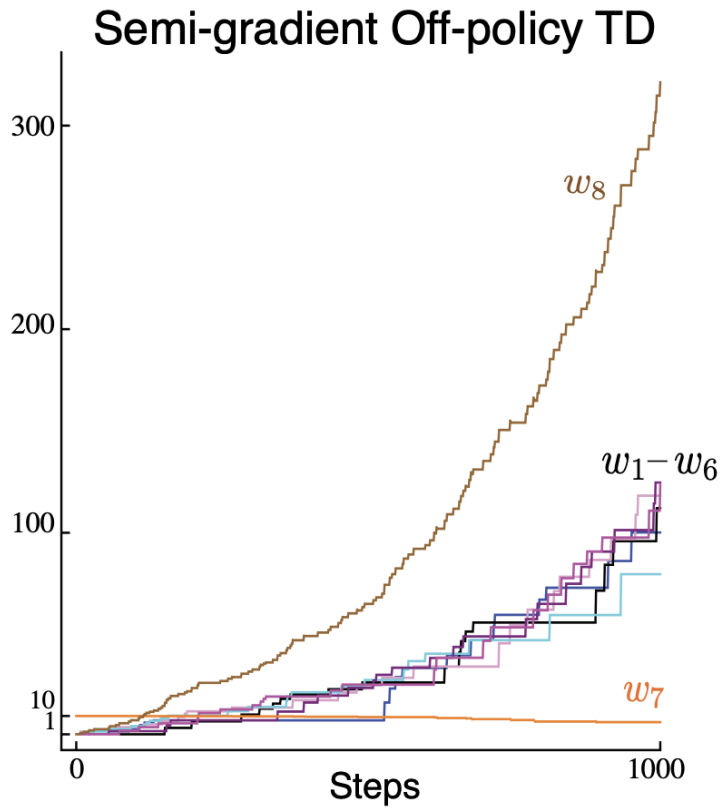


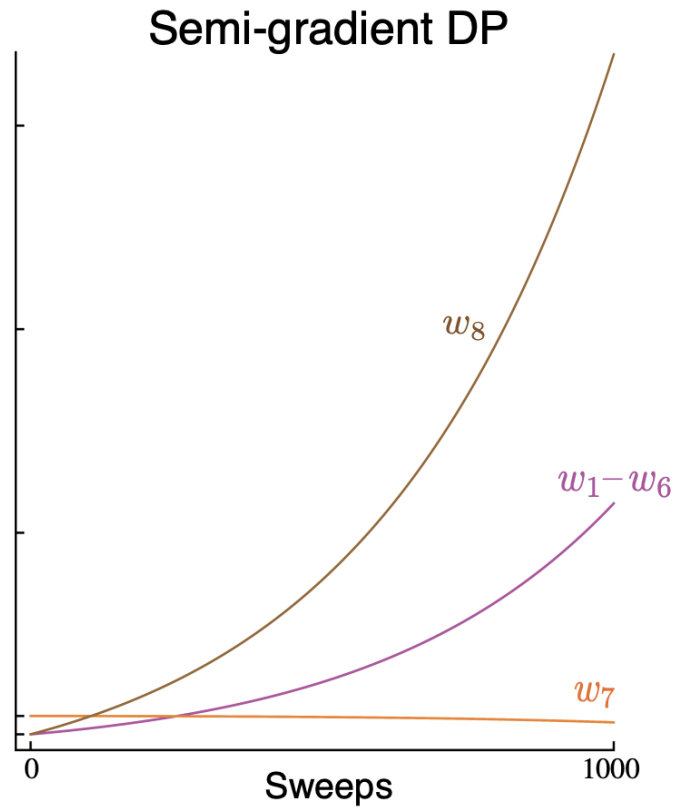
$\pi(\text{solid}|\cdot) = 1$

$b(\text{dashed}|\cdot) = 6/7$

$b(\text{solid}|\cdot) = 1/7$

$\gamma = 0.99$





The Deadly Triad

- **Function Approximation**

$$\hat{Q}(s, a, \boldsymbol{w}) \approx Q^{\pi}(s)$$

- **Bootstrapping**

$$Q(s_t, a_t) = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$$

- **Off-Policy Training**

$$\mu \neq \pi$$

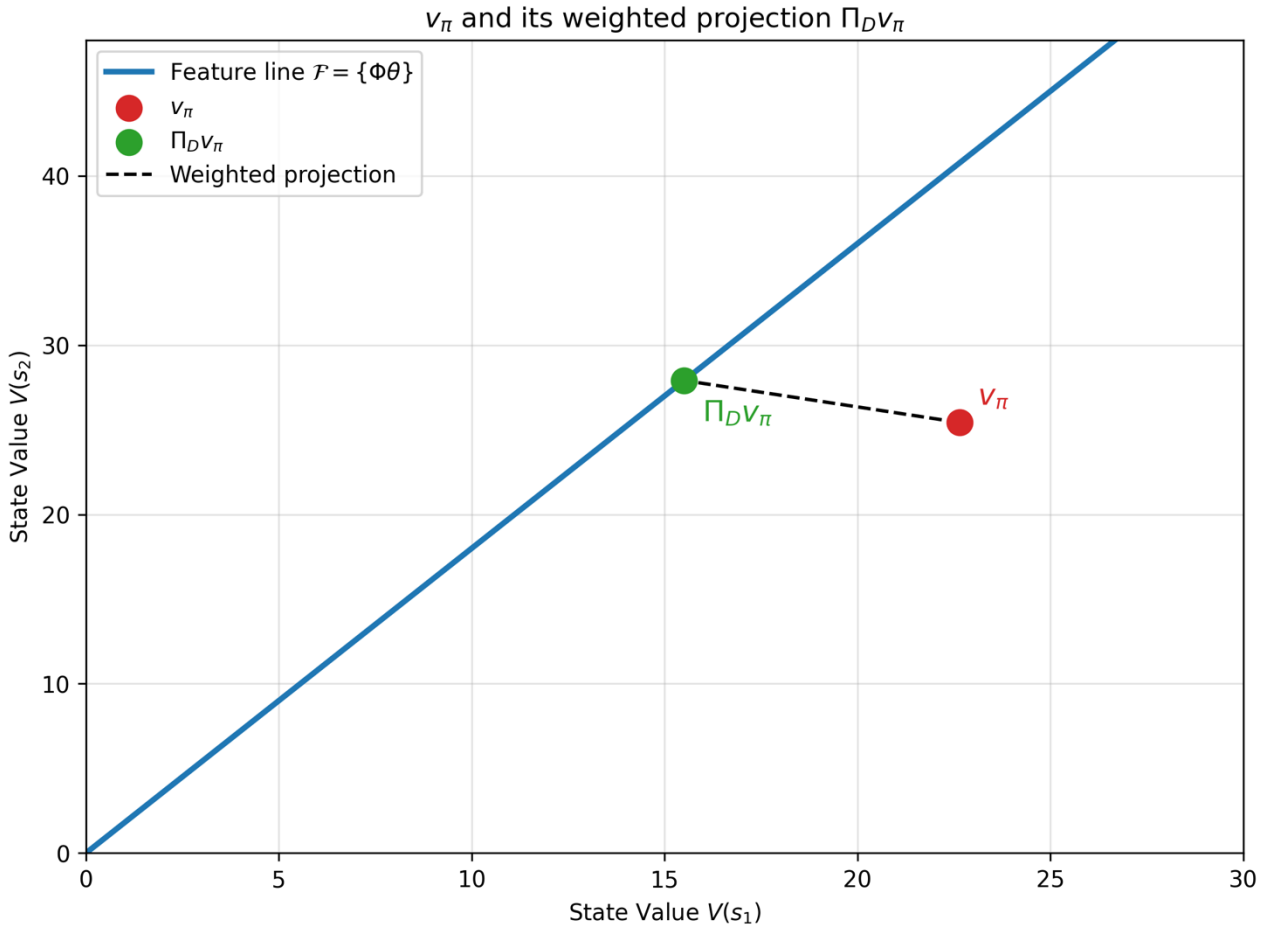
- Algorithms working in this regime are prone to divergence.

Geometric Intuition

- Two states MDP $\{s_1, s_2, T\}$
- True V^π is a point in a plane

- Linear function approximation:
  - $V(s_1, w) = aw$
  - $V(s_2, w) = bw$
  - Where a, b are the scalars representing s_1, s_2
- Possible $V^\pi(\cdot, w)$ is a line

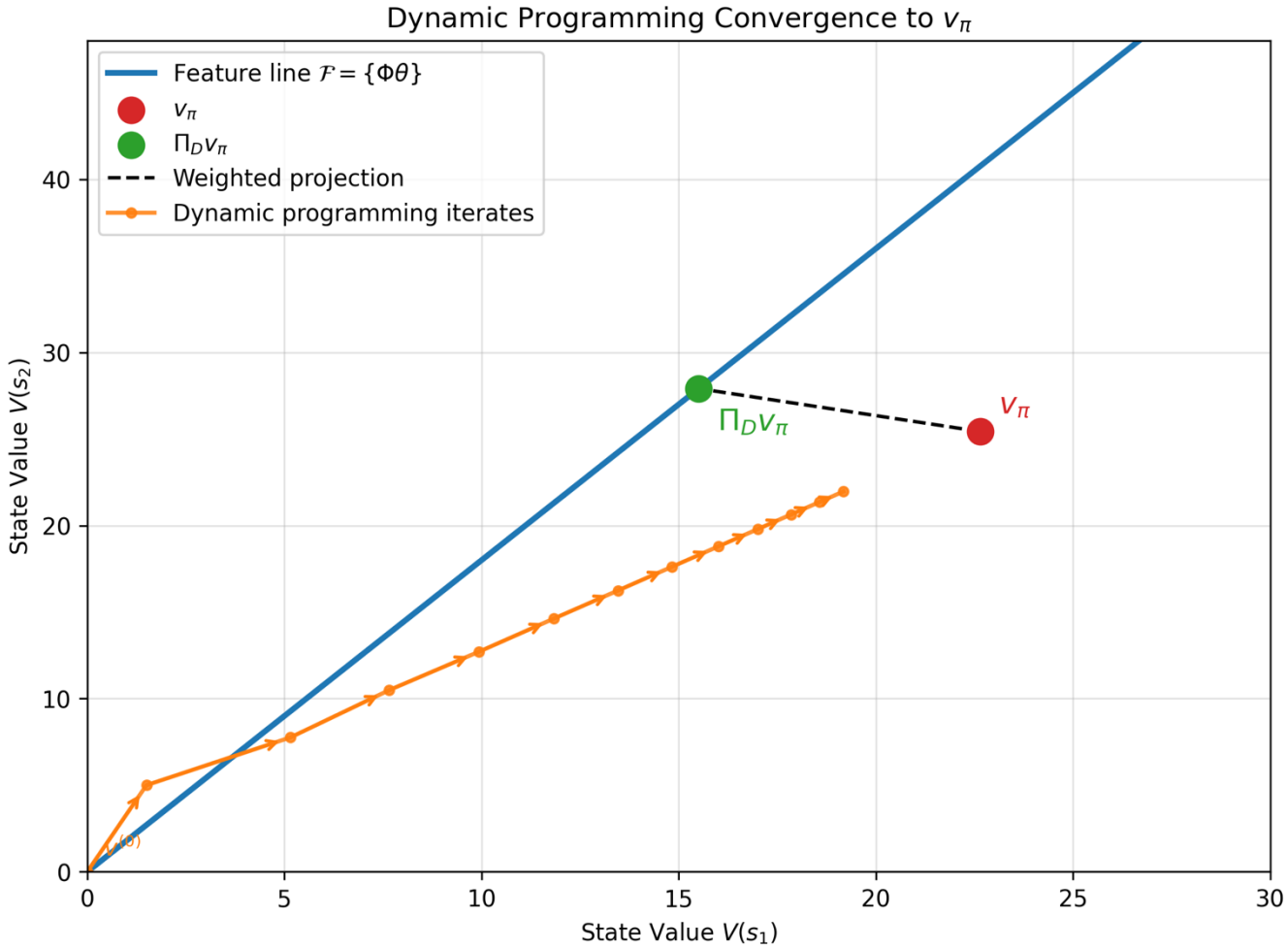
- V^π cannot be fully represented.
  - Green point is the “closest”



Dynamic Programming Updates

- We can use dynamic programming over the full space of value functions.
  - Using, for example, Value Iteration
  - At each iteration, minimize Bellman error.

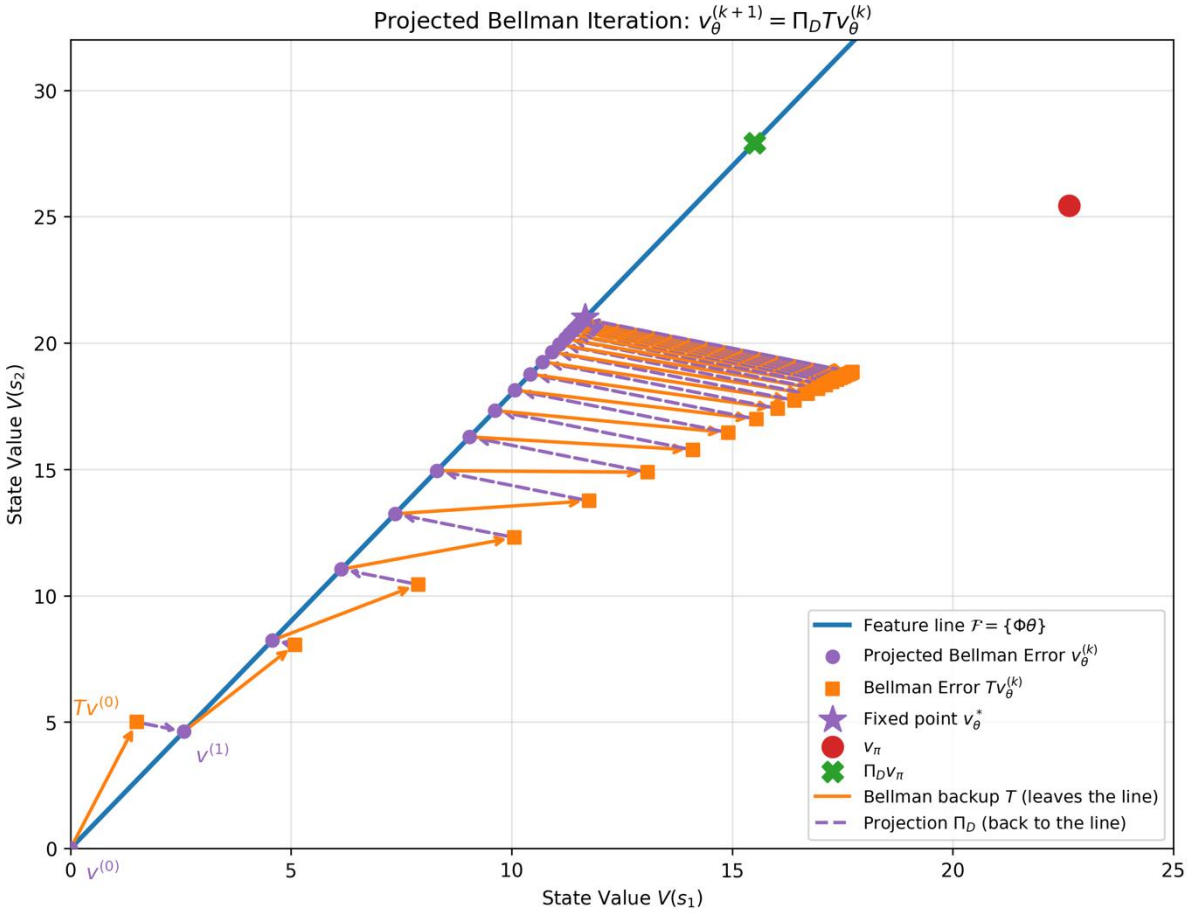
- We will slowly converge to the correct solution.



Value Iteration With Approximation

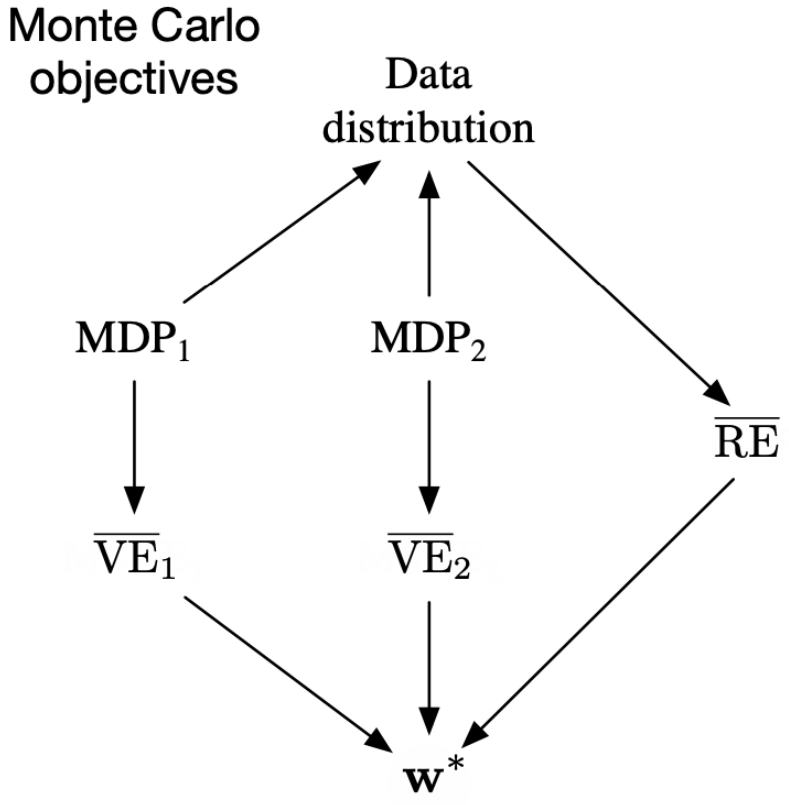
- With approximation, we project the Bellman error back to the function space at every iteration:

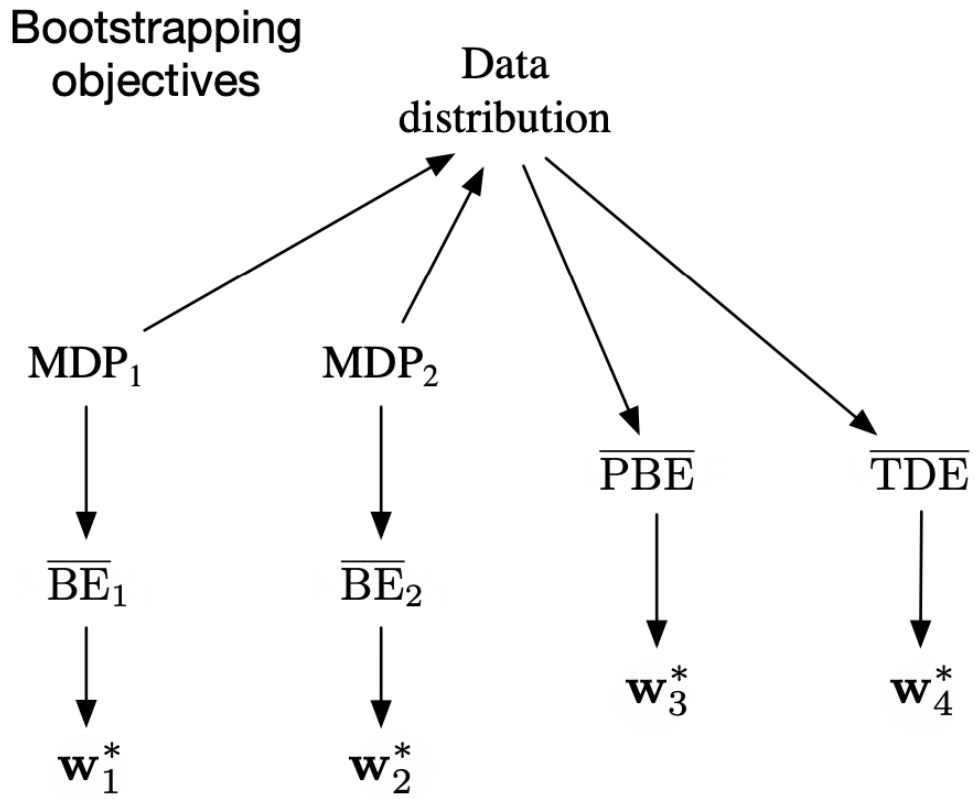
- $\Delta w = \alpha \underbrace{\left(r + \gamma \max_{a'} \hat{Q}(s', a', \boldsymbol{w}) - \hat{Q}(s, a, \boldsymbol{w}) \right)}_{\text{Bellman Error}} \underbrace{\nabla_w \hat{Q}(s, a, \boldsymbol{w})}_{\text{Projection}}$



- The fixed point is not the projection of V^π in the function space.
- Things get worse with off-policy data.

The Bellman Error is Not Learnable





Algebraic Intuition

- The Bellman Operator $\mathcal{B}$ is a contraction  $V \leftarrow \mathcal{B}V$
  - Whose fixed point is the real value (V^{π} or V^{*})

- The projection operator Π is a contraction: $\|\Pi V - \Pi \bar{V}\| \leq \|V - \bar{V}\|$

- But their combination $V \leftarrow \Pi BV$ is not a contraction!
  - It does not converge to ΠV^{π} or ΠV^{*}
  - Gradient descent on the Bellman error might not converge even with infinite data.
  - In practice, it often converges to the wrong thing.

- Same problem when learning Q

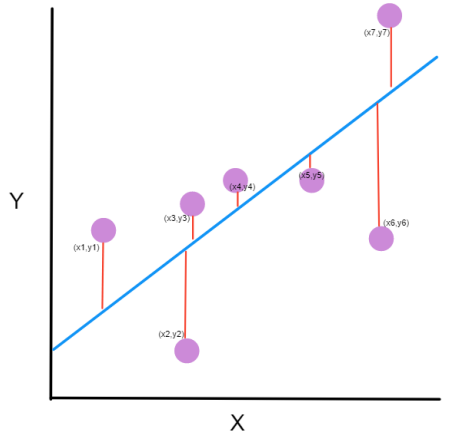
Gradient Descent of Bellman Error on Q

- Given a batch of experience $\{(s, a, s', r)\}$, we want to learn a w that minimizes

$$L(w) = \frac{1}{2}\left(r_t + \gamma \max_{a'} Q(s', a'; w) - Q(s, a; w)\right)^2$$

- How is this different from a traditional regression loss?

$$L(w) = \frac{1}{2}\left(y_i - Q(s, a)\right)^2$$



Gradient Descent of Bellman Error on Q

- The target y_i also depends on the parameter w
- This creates a wild goose chase situation
- Very challenging to stabilize training



Gradient Descent of Bellman Error on Q

$$\Delta w = \alpha \left(r + \gamma \max_{a'} \hat{Q}(s', a', \boldsymbol{w}) - \hat{Q}(s, a, \boldsymbol{w}) \right) \nabla_w \hat{Q}(s, a, \boldsymbol{w})$$

What about $\nabla_w \hat{Q}(s', a', w)$?

- We technically cannot get it since the max operator is not always differentiable.

Summary

- Q-Learning sounds very appealing for learning from off-policy data.
- But unfortunately, we can't prove stability or convergence to the optimal solution in general.
- The problem is compounded when working with large neural nets:
  - Training Q functions often requires significant tuning
  - And even when it converges, it reaches the wrong optimum (also for linear fn)
- Next, we'll see a few popular tricks to tackle this issue

Q-learning with target networks

- $L(w) = \frac{1}{2}\left(r_t + \gamma \max_{a'} Q(s', a'; w) - Q(s, a; w)\right)^2$



- Stabilize training by using different weights for the regression target
- $L(w) = \frac{1}{2}\left(r_t + \gamma \max_{a'} Q(s', a'; \overline{w}) - Q(s, a; w)\right)^2$

- Perfectly fine regression:
  - The target does not depend on the current weight

Q-Learning with Target Networks

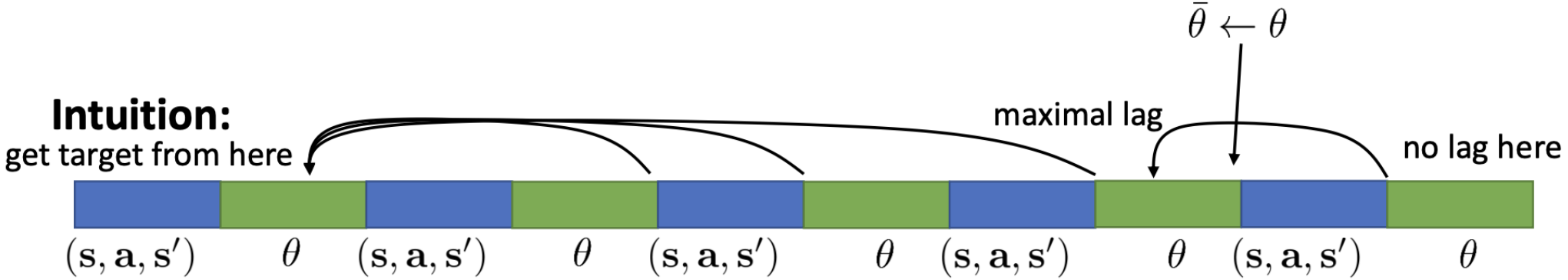
- What to use as weights of the target $\overline{w}$?
- First option: Some old w
  - Init $\boldsymbol{w}$
  - Collect one episode with *some* policy μ, add it to the replay buffer D
  - For K iterations:
    - Sample a batch of transitions $\{(s, a, s', r)\}$ from D
    - For every transition: $\Delta w = \alpha \left(r + \gamma \max_{a'} \hat{Q}(s', a', \overline{\boldsymbol{w}}) - \hat{Q}(s, a, \boldsymbol{w}) \right) \nabla_w \hat{Q}(s, a, \boldsymbol{w})$
  - Every N (>>0) steps: $\overline{w} \leftarrow w$



- Do you see any problem with this?

Q-Learning with Target Networks

- Using an old w creates an uneven balance of the lag between the current and target network



- How to balance the lag more evenly?

Q-Learning with Target Networks

- Second option: A slow-moving average of w_t:
- Algorithm:
  - Init $\boldsymbol{w}$
  - Collect one episode with *some* policy μ, add it to the replay buffer D
  - For K iterations:
    - Sample a batch of transitions $\{(s, a, s', r)\}$ from D
    - For every transition: $\Delta w = \alpha \left(r + \gamma \max_{a'} \hat{Q}(s', a', \bar{\boldsymbol{w}}) - \hat{Q}(s, a, \boldsymbol{w}) \right) \nabla_w \hat{Q}(s, a, \boldsymbol{w})$
    - $\bar{w} \leftarrow m\bar{w} + (1 - m)w \qquad (m = 0.999)$

Maximization Bias in Q-Learning

- Target value $y_i = r(s,a) + \gamma \max_{a'} Q(s', a'; \bar{w})$

- Maximization bias:
  - We design algorithms for estimating the Q function that are (at best) unbiased on average for each action individually:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\hat{Q}(s,a)\right]$$

  - This creates a systematic positive bias when computing targets:

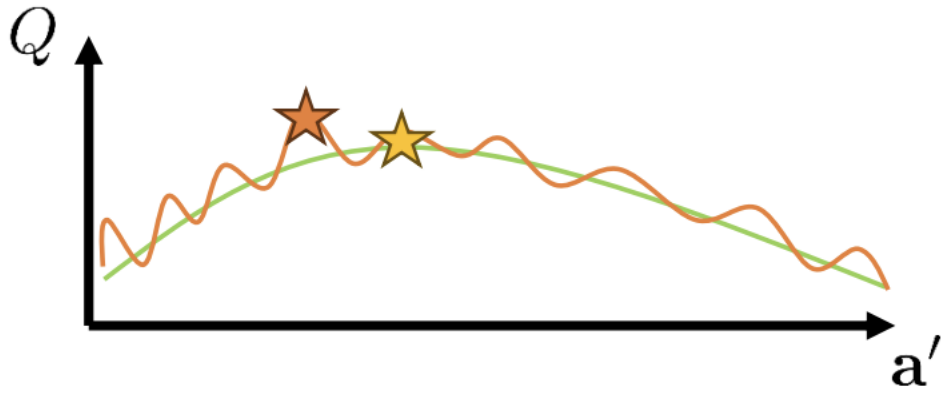
$$\max_a \mathbb{E}\left[\hat{Q}(s,a)\right] \leq \mathbb{E}\left[\max_a \hat{Q}(s,a)\right]$$

What we want

What we are doing above

Maximization Bias in Q-Learning

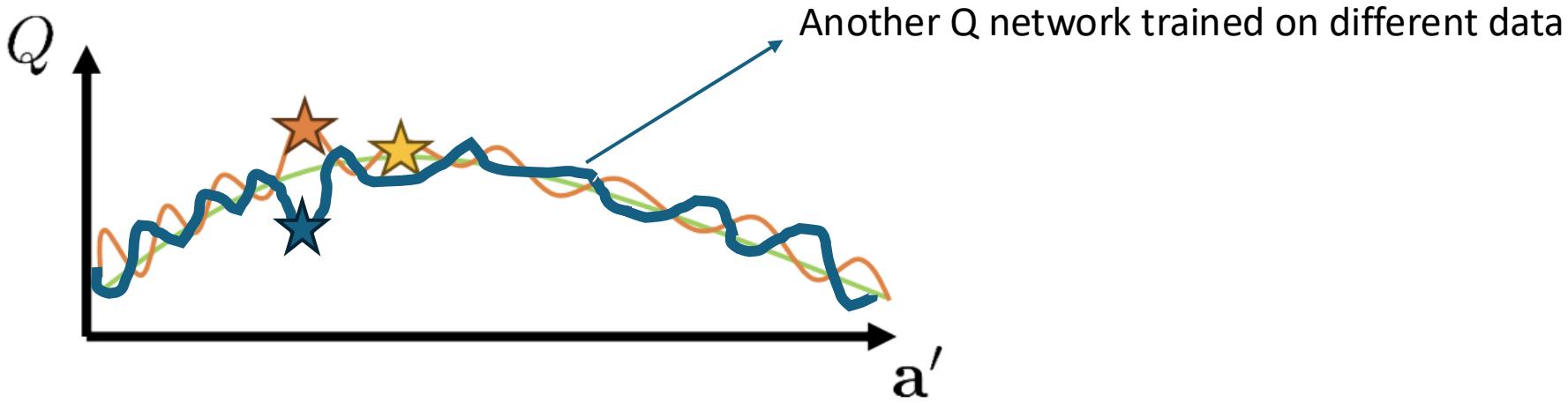
- $\max_a \mathbb{E}\left[\hat{Q}(s,a)\right] \leq \mathbb{E}\left[\max_a \hat{Q}(s,a)\right]$



- Why is this a problem?
  - The max operator leads to consistent overestimation of the Q value of the state
  - The error accumulates through the entire state space, propagating backwards through successive TD updates.

Double Q-Learning

- **Decouple** action selection and Q function evaluation.



- Works *only* if the noise between the two networks is decorrelated.

Double Q-Learning

- How to get two decorrelated Q networks?

- First option (Traditional Double Q-Learning):
  - Train two networks Q_a, Q_b on non-overlapping subsets of the replay buffer.
  - Target for Q_1: $y_i = r(s,a) + \gamma Q_2\left(s', \arg\max_{a'} Q_1(s', a')\right)$
  - Target for Q_2: $y_i = r(s,a) + \gamma Q_1\left(s', \arg\max_{a'} Q_2(s', a')\right)$

Double Q-Learning

- How to get two decorrelated Q networks?

- Second option (Popular in Practice):
  - Use the current and target networks as the two Q functions.
  - Target $y_i = r(s,a) + \gamma Q\left(s', \arg\max_{a'} Q(s', a', w)\,;\, \bar{w}\right)$
  - Why is this technically wrong?

From Q-Learning to Sarsa

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big(r_{t+1} + \gamma Q(s_{t+1}, a') - Q(s_t, a_t)\big)$$

- Q-Learning: Sample the greedy action at the next step

$$\mathrm{a}' = \mathrm{argmax}_{\mathrm{a}} Q(s', a)$$

- SARSA: Sample from the current policy

$$a' = \pi(\cdot\,|s')$$

- Both can be used on off-policy data without importance sampling

From Q-Learning to SARSA

- Fixed point of Q-Learning: Q^*

$$Q^*(s_t, a_t) = r_{t+1} + \gamma \mathbb{E}_{s_{t+1}}[Q^*(s_{t+1}, a')]$$

  - If $a' = \text{argmax}_a Q(s', a)$

- Fixed point of Sarsa: Q^π

$$Q^\pi(s_t, a_t) = r_{t+1} + \gamma \mathbb{E}_{s_{t+1}}[Q^\pi(s_{t+1}, a')]$$

  - If $a' = \pi(\cdot\,|s')$

SARSA with Function Approximation

- Algorithm:
  - Init $\boldsymbol{w}$
  - Collect one episode with *some* policy μ, add it to the replay buffer D
  - For K iterations:
    - Sample a batch of transitions $\{(s, a, s', r)\}$ from D
    - For every transition: $\Delta w = \alpha\left(r + \gamma\hat{Q}(s', a' \sim \pi_\theta(s'), \boldsymbol{w}) - \hat{Q}(s, a, \boldsymbol{w})\right)\nabla_w \hat{Q}(s, a, \boldsymbol{w})$
  - Get a new policy from the current Q^π
    - $\pi_\theta(\cdot) = \epsilon$-greedy$(Q^\pi)$
- Do we need a target network with parameters $\overline{w}$?
- Do we need the double network trick to avoid the maximization bias?

Double SARSA with Target Networks

- Algorithm:
  - Init $\boldsymbol{w}$
  - Collect one episode with *some* policy μ, add it to the replay buffer D
  - For K iterations:
    - Sample a batch of transitions $\{(s, a, s', r)\}$ from D
    - For every transition: $\Delta w = \alpha\left(r + \gamma\hat{Q}(s', a' \sim \pi_\theta(s'), \bar{\boldsymbol{w}}) - \hat{Q}(s, a, \boldsymbol{w})\right)\nabla_w \hat{Q}(s, a, \boldsymbol{w})$
    - $\bar{w} \leftarrow m\bar{w} + (1 - m)w \qquad (m = 0.999)$
  - Get a new policy from the current Q^π
    - $\pi(\cdot) = \epsilon$-greedy(Q^π)

Continuous Action

- How to find a maximum if the actions are continuous?
  - $\pi(a_t|s_t) = \begin{cases} 1 \; if \; a_t = \arg\max\limits_{a} Q_\theta(s_t, a_t) \\ 0 \; otherwise \end{cases}$
  - Target value $y_i = r(s,a) + \gamma Q\left(s', \arg\max\limits_{a'} Q(s', a', w)\,; \bar{w}\right)$
- We need something that is fast enough yet accurate

Finding a Maximum With Stochastic Optimization

- Simplest solution: *sampling-based* optimization.
  - Select N action candidates $(a_1, \dots, a_N)$ from some distribution (uniform)
  - $\max_a Q(s, a) \approx \max\{Q(s, a_1), \dots, Q(s, a_N)\}$

- Works surprisingly well for low-dim actions (up to 40).
  - Efficiently parallelizable
  - But sometimes not very accurate
  - Sometimes called *sample and rank* or *rejection sampling*

- A more accurate solution can be found by changing the sampling distribution (e.g., cross-entropy methods or CMA-ES).

Learning a "Maximizer" Policy

- **Idea:** Train a network $\pi_\theta(s)$ such that $\pi_\theta(s) \approx \arg\max_a Q(s, a, w)$

- Update the parameters θ in the direction that locally maximizes $V(s)$:

$$\hat{g} = \frac{dQ(s, \pi_\theta(s), w)}{d\theta} = \frac{dQ(s, \pi_\theta(s), w)}{d\,\pi_\theta(s)} \frac{d\pi_\theta(s)}{d\theta}$$

$$\theta^{new} \leftarrow \theta^{old} + \alpha \hat{g}$$

- Nice connection between policy gradients and Q-learning methods!

Learning a “Maximizer” Policy

- Use the “maximizer” policy π_θ to select actions
- Old Target value $y_i = r(s,a) + \gamma Q\left(s', \arg\max_{a'} Q(s', a', w)\,; \bar{w}\right)$
- New Target value $y_i = r(s,a) + \gamma Q(s', \pi_\theta(s'); \bar{w})$
- To avoid the maximization bias, use a target network $\pi_{\bar{\theta}}(s)$

Deep Deterministic Policy Gradients (DDPG)

- Init w, θ
- Collect one episode with *the policy* π_θ, add it to the replay buffer D
  - For K iterations:
    - Sample a batch of transitions $\{(s, a, s', r)\}$ from D
    - Update Critic: $\Delta w = \alpha \left(r + \gamma \hat{Q}(s', a' \sim \pi_{\bar{\theta}}(s'), \bar{\boldsymbol{w}}) - \hat{Q}(s, a, \boldsymbol{w}) \right) \nabla_w \hat{Q}(s, a, \boldsymbol{w})$
    - Update Actor $\Delta\theta = \beta \left(\frac{dQ(s, \pi_\theta(s), w)}{d\pi_\theta(s)} \frac{d\pi_\theta(s)}{d\theta} \right)$
    - $\bar{w} \leftarrow m\bar{w} + (1 - m)w \qquad (m = 0.999)$
    - $\bar{\theta} \leftarrow m\bar{\theta} + (1 - m)\theta \qquad (m = 0.999)$

Summary

- Q-Learning/SARSA-style methods are great for off-policy learning.
- But they are unstable and hard to tune when used in tandem with function approximation.
- There are a few tricks we can use to improve convergence.
- Next time, we'll see additional ideas to make off-policy work in practice in continuous domains.