Reinforcement Learning

CS 59300: RL1

September 16, 2025

Joseph Campbell

Department of Computer Science

PURDUE UNIVERSITY®

Today's lecture

1. Assignment 1

2. Continuous action spaces for Q-learning

3. Overestimation bias in Q-learning

Some content inspired by David Silver's UCL RL course and Katerina Fragkiadaki's CMU 10-403

Assignment 1

Recap: Q-learning: off-policy TD learning

In Sarsa...

$$Q(s,a) = Q(s,a) + \alpha\,(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s,a))$$

In Q-learning...

$$Q(s,a) = Q(s,a) + \alpha\,(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a') - Q(s,a))$$

Remember Bellman optimality equations?

$$Q^*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a \in \mathcal{A}} Q^*(s',a')]$$

Recap: Deep Q-Networks

Simply replace our original estimate of Q with our approximation

$$\hat{Q}(s,a,\mathbf{w}) = \hat{Q}(s,a,\mathbf{w}) + \alpha\left(r_{t+1} + \gamma \max_{a'\in\mathcal{A}} \hat{Q}(s_{t+1},a',\mathbf{w}) - \hat{Q}(s,a,\mathbf{w})\right)$$

$$\Delta\mathbf{w} = \overset{\text{Step size}}{\alpha}\left(\overbrace{r_{t+1} + \gamma \max_{a'\in\mathcal{A}} \hat{Q}(s_{t+1},a',\mathbf{w}) - \hat{Q}(s,a,\mathbf{w})}^{\text{Magnitude of error}}\right)\overbrace{\nabla_{\mathbf{w}}\,\hat{Q}(s,a,\mathbf{w})}^{\text{Direction of error}}$$

Initialize replay memory D to capacity N

Initialize action-value function $\hat{Q}$ with random weights θ and $Q^{\mathrm{Tar.}}$ with $\theta_2 = \theta$

for episode = 1…M do

Initialize sequence $s_1 = \{x_1\}$ and pre. seq. $\phi_1 = \phi(s_1)$

for t=1…T do

With probability ϵ select a random action a_t

otherwise $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

Execute action a_t and observe r_t and x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and pre. $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sample random minibatch from D

Set $y_j = r_j$ if episode ends else $r_j + \gamma \max_{a'} Q^{\mathrm{Tar.}}\{ (\phi_{j+1}, a'; \theta_2)$

Perform a gradient step on $(y_j - \hat{Q}(\phi_j, a_j; \theta))^2$

Every C steps set $Q^{\mathrm{Tar.}} = \hat{Q}$

Continuous action spaces for Q-learning

So far we have discussed discrete actions...

$$Q(s,a) = Q(s,a) + \alpha\,(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a') - Q(s,a))$$

Requires examining each action and finding the highest Q-value

Does this work in a continuous action space?

- *Hint: Can we enumerate all actions?*

Max as inner optimization

The ”simple” method for finding $\max_{a' \in \mathcal{A}}$ is to perform optimization

Cross-Entropy Method

- Start with a randomly initialized normal distribution
- Sample actions from it
- Select top-K actions sorted by $Q(s, a)$
- Fit distribution to top-K samples
- Repeat

Max as inner optimization

The ”simple” method for finding $\max_{a' \in \mathcal{A}}$ is to perform optimization

Cros

- Sta
- San
- Sel
- Fit c
- Repeat

We refer to this as ”Cross-Entropy Method” because we fit the new distribution to the top-K samples.

In other words, we minimize the cross-entropy to the new sampling distribution.

QT-Opt

Goal: use stochastic optimization to find target $Q_T(s_{t+1}, a')$

for episode = 1…M do
  for t=1…T do
    Perform CEM to find $a_t = \max_a \hat{Q}\ (s_t, a; \theta)$
    With probability ϵ select random action else a_t
    Execute action a_t and observe r_t and s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D and sample random minibatch
    Set $y_j = r_j$ if episode ends else $r_j + \gamma \max_{a'} \hat{Q}\ \{\ (s_{j+1}, a'; \theta)$
    Perform a gradient step on $(y_j - \hat{Q}(\phi_j, a_j; \theta))^2$

Kalashnikov et al, *QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation*, 2018.

https://youtu.be/W4joe3zzglU?si=9milzRXNGDnbhz3m

What are the limitations of this method?

What are the limitations of this method?

Extremely computationally expensive

- Every time we take an action we must perform inner optimization

CEM is not guaranteed to find the best action

- Only *approximate* solution, meaning targets become biased

CEM doesn't work well in high-dimensional action spaces

A more sophisticated solution

We can derive Q-values using a different equation!

Advantage

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

The advantage represents how good action a is relative to π

- $A_\pi(s, a) > 0$: a is better than what I would get with π
- $A_\pi(s, a) < 0$: a is worse

A more sophisticated solution

By re-ordering the equation, we get

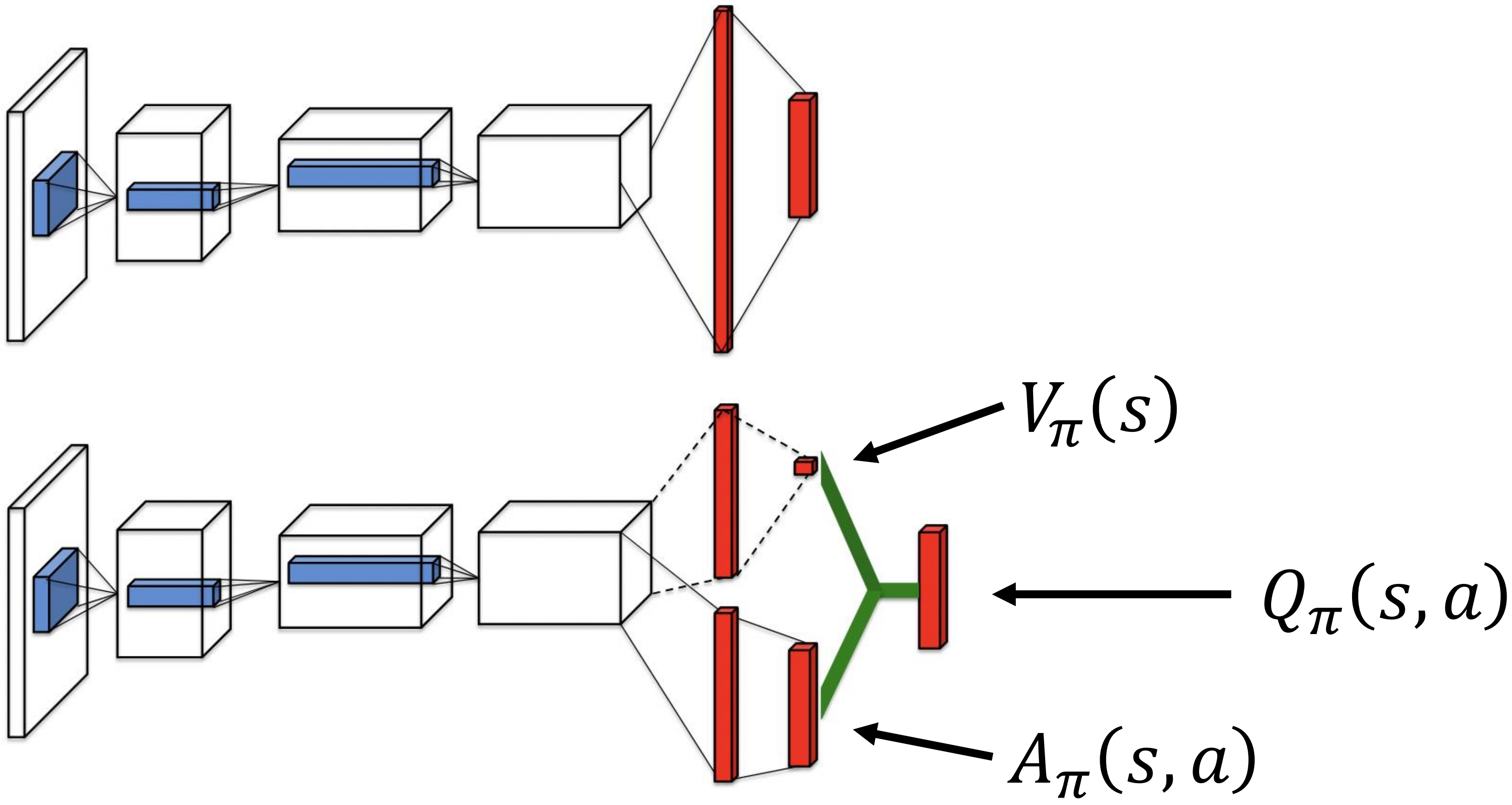
$$Q_\pi(s,a) = A_\pi(s,a) + V_\pi(s)$$

Important observation: instead of directly predicting Q-values, what if we predict both advantages and state-values?

Dueling DQN

DQN

Dueling DQN



Wang et al, *Dueling Network Architectures for Deep Reinforcement Learning*, 2016.

Decomposed Q-Networks

Intuition: learn which states are good without considering the effect of actions.

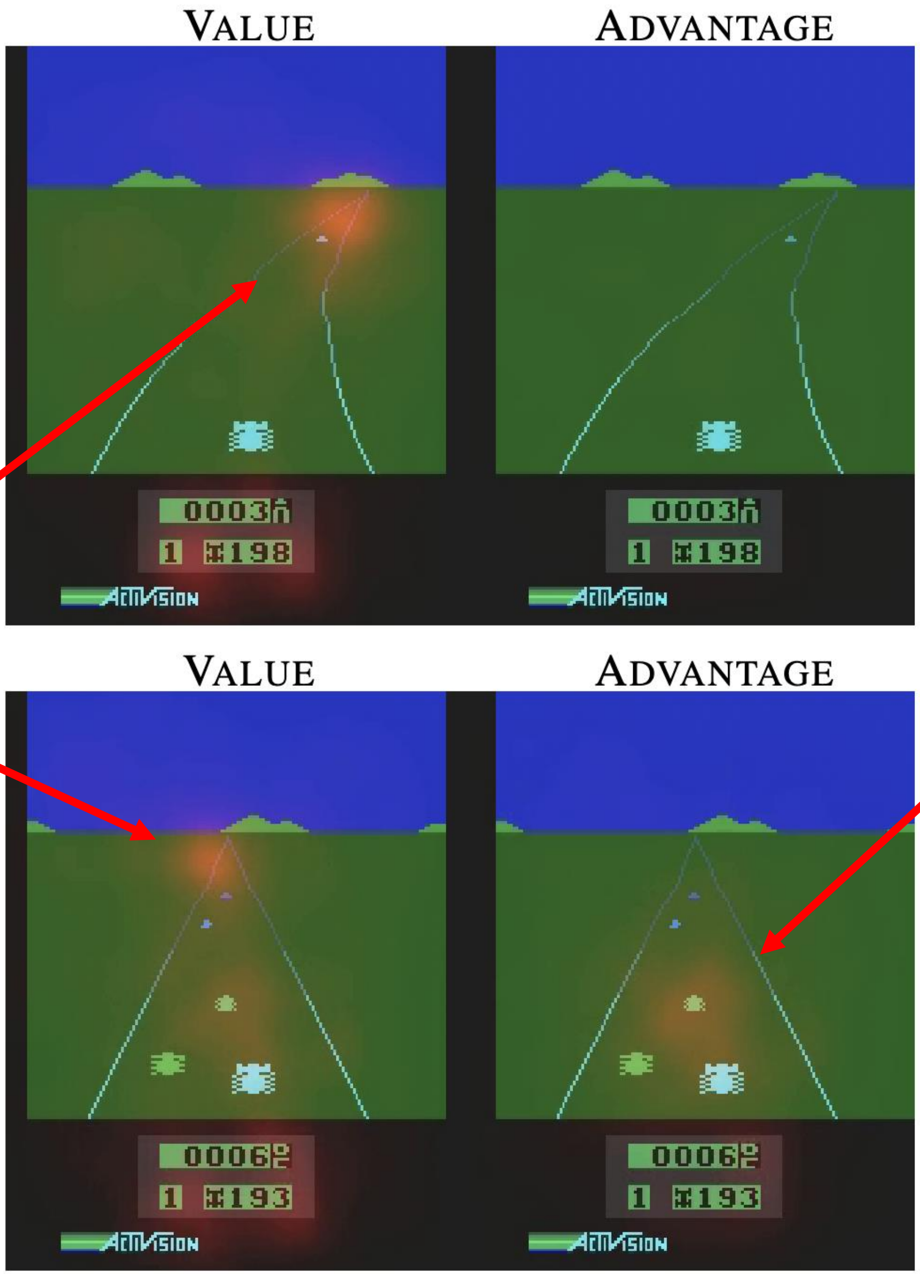
Benefits:

- Shared layers mean Q-value updates also update state-values
- Differences between Q-values for a given state are often small

Particularly useful for states in which actions have little-to-no effect!

- Creates an inductive bias which simplifies learning.

Value estimates pay attention to road.



Advantage estimates pay attention to cars.

How does this apply to continuous actions?

Start with dueling DQN, and further decompose the advantage

$$A_\pi(s,a) = -\frac{1}{2}\big(a - \mu(s)\big)^T P(s)(a - \mu(s))$$

Positive-definite square matrix.
Obtained via Cholesky decomposition: $L(s)L(s)^T$

Assumption: quadratic dynamics and linear rewards

- The advantage is parameterized as a quadratic function

Decomposed advantage

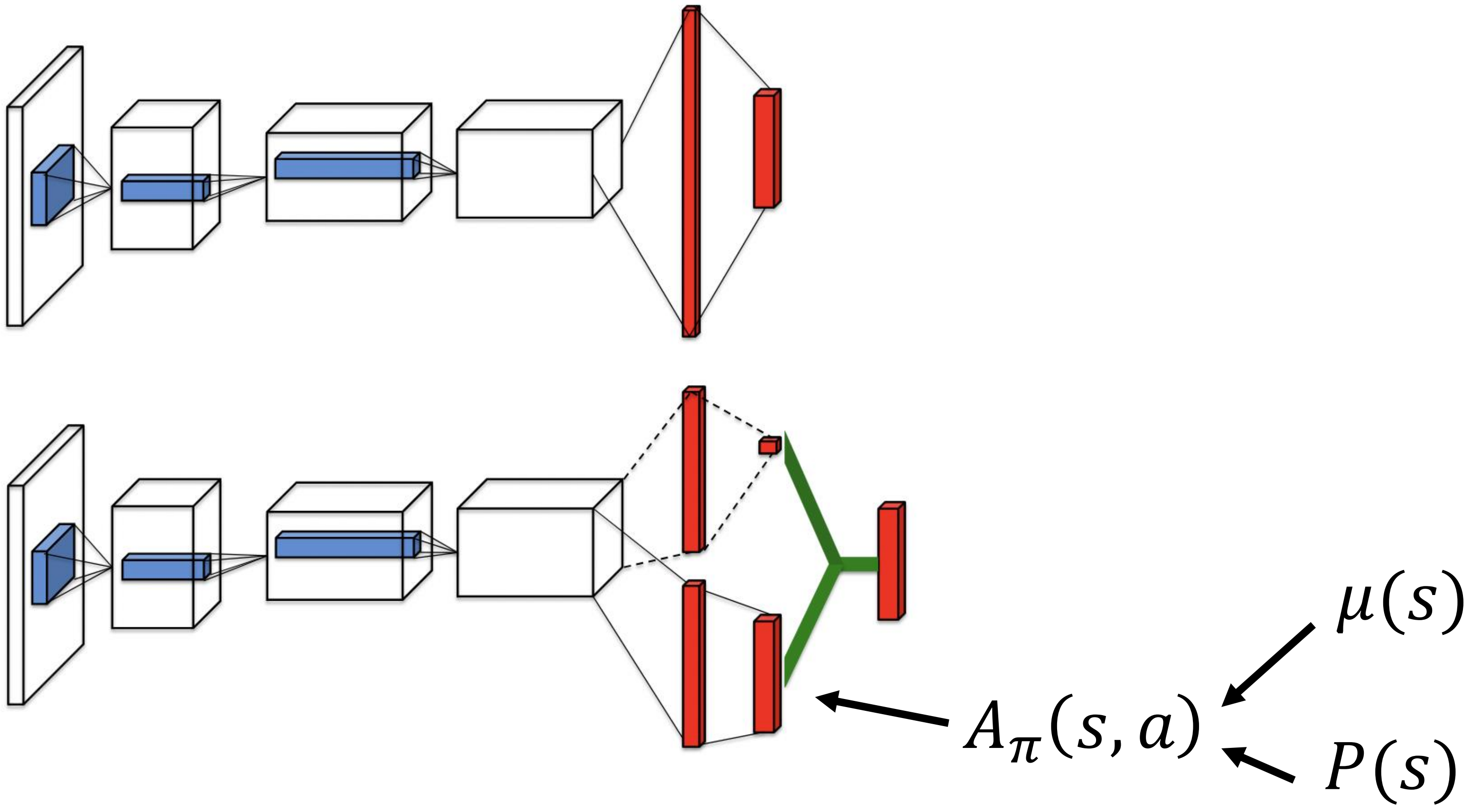
Since Q is quadratic in a…

$$\max_{a \in \mathcal{A}} \hat{Q}\ (s_t, a) = \mu(s)$$

Quadratic formulation of advantage ensures convexity

- Normal distribution with mean μ and covariance P

Decomposed advantage



Overestimation bias in Q-learning