Offline RL

CISC 7404 - Decision Making

Steven Morad

University of Macau

Admin 2
Review 3
Offline RL 4
Behavioral Cloning with Rewards 6
The Deadly Triad 7
Constraining Q 9
Conclusion 10

Admin

Admin

Two more lectures after today

- Make sure you start on final projects!

Homework 1 grading is done

- Results on moodle

Homework 2 grading deadline next Wednesday

Admin

Quiz 1 scores uploaded

- Mean grade 57/100, modes at 25 and 85
- Someone forget their name
  - If you have no score, come see me
- 2 students with 100% on quiz 1 and quiz 2 can skip final quiz

Mean course grade is 92%

- Lower mean grade after final quiz (second quiz dropped for most)

Final quiz on 17 April (next week), format subject to change

- Question fundamental RL (V/Q/PG)
- Question actor-critic
- Question on new material (imitation/offline RL/etc)

Review

Offline RL

Offline RL

Review: In on-policy RL, each iteration we collect a **new** dataset using our policy

In off-policy RL, each iteration we **update** our dataset using our policy

In imitation learning, we are given a fixed **expert** dataset

Question: Did you find imitation learning interesting? Why?

Offline RL

Imitation learning from a fixed dataset is attractive for many reasons

- Fixed dataset results in much simpler code
  - No need to collect new data, step environment, etc
- Easier and more stable to train (supervised learning)

But imitation learning (IL) also has disadvantages

- Only imitates, does not think or plan
  - Can only do as good as expert
  - Humans are usually bad "experts"
  - We want to do **better** than humans

Question: Can we learn policies from fixed datasets that do better than the experts?

Offline RL

In **offline RL** or **batch RL**, we learn without exploration

Like imitation learning, learn from a fixed dataset

- Simpler implementation
- Can learn from large existing datasets without collecting it youurself

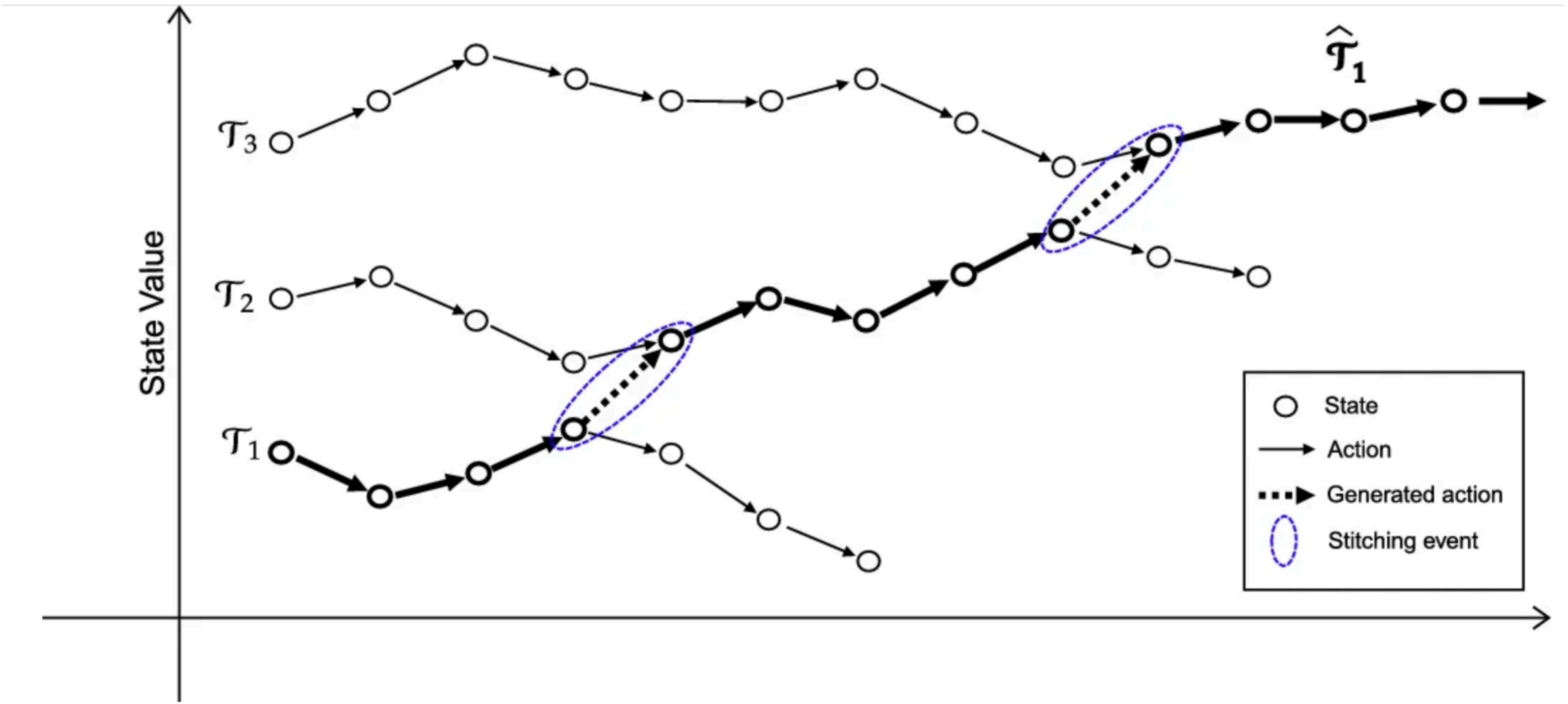
Unlike imitation learning, offline RL can do **better** than the expert

- Can learn an optimal policy from a random “expert”!
- How is this possible without exploration?

In imitation learning, we learn to imitate dataset trajectories

In offline RL, we learn to **stitch** together subtrajectories into optimal trajectories

Offline RL



$\mathcal{T}_3$
$\hat{\mathcal{T}}_1$
$\mathcal{T}_2$
$\mathcal{T}_1$
State Value
State
Action
Generated action
Stitching event

Offline RL

Offline RL is a very new field

If we can make offline RL work reliably, it can be very powerful

- Learn superhuman driving from existing Xiaomi dataset
- Learn superhuman surgery from existing surgery videos
- Learn language model finetuning from existing internet text

I will also present my work on offline RL at ICRA next month

https://sites.google.com/view/llm-marl/home

Offline RL

Definition: An **offline MDP** consists of:

- MDP with unknown $\mathcal{R}$, Tr
- Dataset $\boldsymbol{X}$ of episodes
  - Following a policy $\pi(a \mid s; \theta_\beta)$

$$(S, A, \mathcal{R}, \mathrm{Tr}, \gamma, \boldsymbol{X})$$

$$\mathcal{R}(s_{t+1}) = ? \qquad \mathrm{Tr}(s_{t+1} \mid s_t, a_t) = ?$$

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{E}_{[1]} & \boldsymbol{E}_{[2]} & \ldots \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} s_0 & a_0 & d_0 & r_1 \\ s_1 & a_1 & d_1 & r_2 \\ \vdots & \vdots & \vdots & \end{bmatrix} & \begin{bmatrix} s_0 & a_0 & d_0 & r_1 \\ s_1 & a_1 & d_1 & r_2 \\ \vdots & \vdots & \vdots & \end{bmatrix} & \ldots \end{bmatrix}$$

Offline RL

Goal: learn a policy that maximizes the return

$$\arg\max_{\theta_\pi} \mathbb{E}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0; \theta_\pi]$$

There are two ways to approach offline RL

- Improve behavior cloning with rewards
- Off-policy RL without exploration

Let us begin with behavior cloning first

Behavioral Cloning with Rewards

Behavioral Cloning with Rewards

Recall the behavior cloning objective

Want to minimize difference between learned and expert policy

$$\arg\min_{\theta_\pi} \sum_{s \in \boldsymbol{X}} \mathrm{KL}\left(\pi\left(a \mid s; \theta_\beta\right), \pi\left(a \mid s; \theta_\pi\right)\right)$$

From this, we derive the cross entropy loss

$$\arg\min_{\theta_\pi} \sum_{s \in \boldsymbol{X}} \sum_{a \in A} -\pi\left(a \mid s; \theta_\beta\right) \log \pi\left(a \mid s; \theta_\pi\right)$$

Behavioral Cloning with Rewards

$$\underset{\theta_\pi}{\arg\min} \sum_{s \in \boldsymbol{X}} \sum_{a \in A} -\pi(a \mid s; \theta_\beta) \log \pi(a \mid s; \theta_\pi)$$

Consider the following situation:

- Two possible actions $A = \{a_+, a_-\}$
- Single state s_0 in the dataset, visited **twice**
- First time, expert takes action a_+ in s
- Second time, expert takes action a_- in s

Expert must behave better in one state than the other!

$$\underset{\theta_\pi}{\arg\min} \sum_{a \in \{a_+, a_-\}} -\pi(a \mid s_0; \theta_\beta) \log \pi(a \mid s_0; \theta_\pi)$$

Behavioral Cloning with Rewards

$$\pi(a_+ \mid s_0; \theta_\beta) = 0.5$$

$$\pi(a_- \mid s_0; \theta_\beta) = 0.5$$

Question: Which action is better behavior?

Behavioral Cloning with Rewards

$$\pi(a_+ \mid s_0; \theta_\beta) = 0.5 \qquad \pi(a_- \mid s_0; \theta_\beta) = 0.5$$

Question: Two actions in same state, what policy θ_π does BC learn?

Answer: Randomly choose $a \in \{a_+, a_-\}$

Question: Is this a good idea?

Behavioral Cloning with Rewards

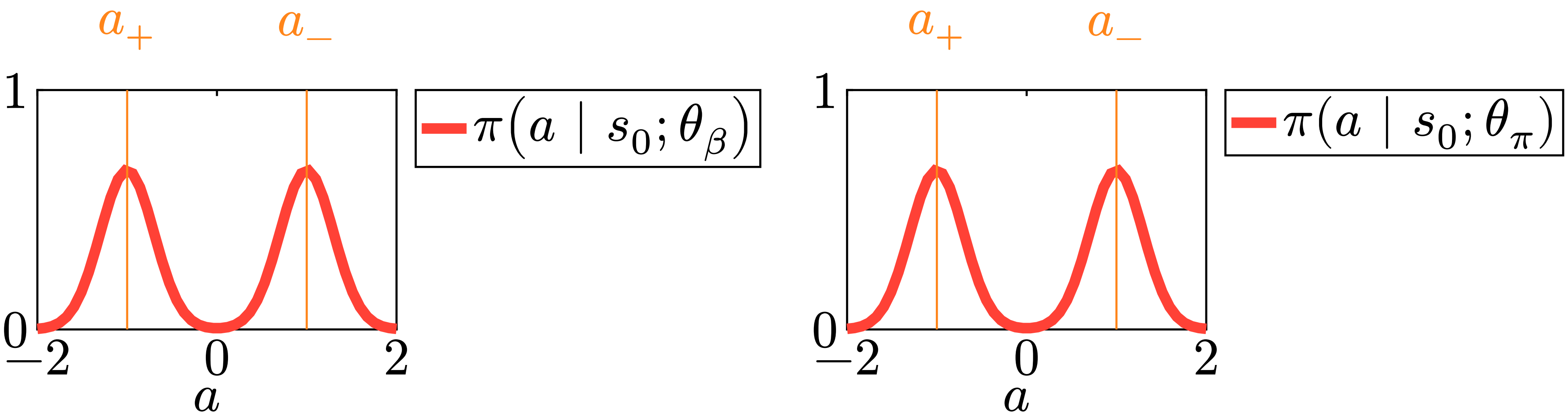
$$\pi(a_+ \mid s_0; \theta_\beta) = 0.5 \qquad \pi(a_- \mid s_0; \theta_\beta) = 0.5$$

Question: We know which action is better, how can we measure this?

Answer: Reward! In BC, no reward. In offline RL, we have reward!

Behavioral Cloning with Rewards

Expert has equal probability for both good and bad actions

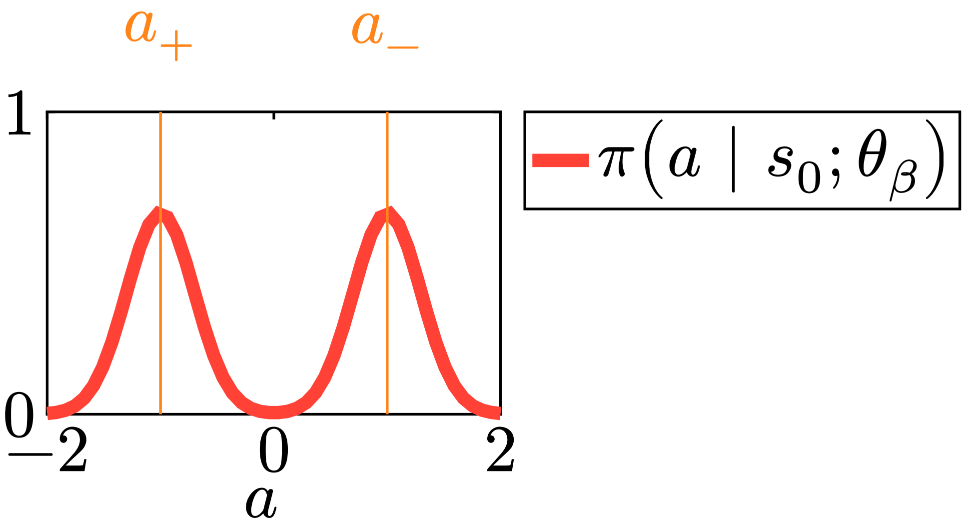


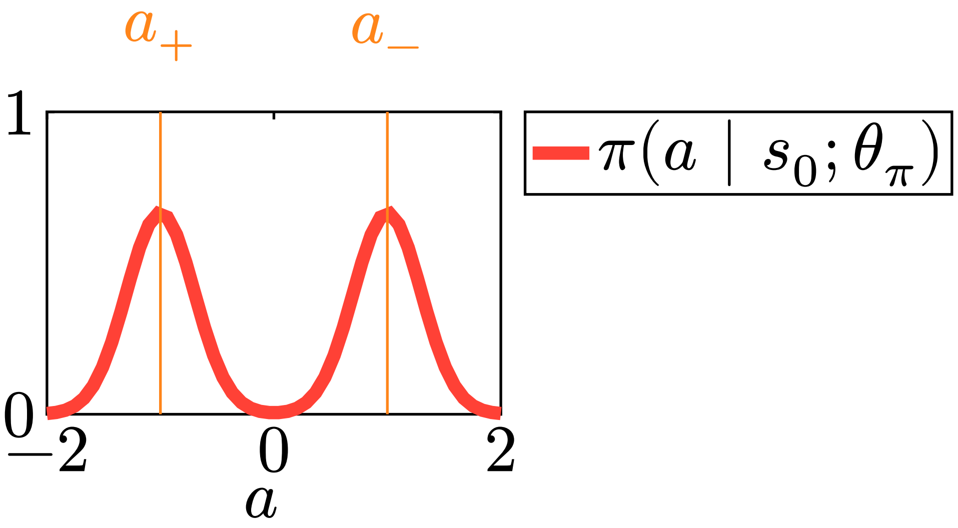
With IL, we can only imitate the expert

With offline RL, we have empirical rewards/returns, we can do better!

$$\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] \qquad \hat{\mathbb{E}}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0; \theta_\beta]$$

Behavioral Cloning with Rewards



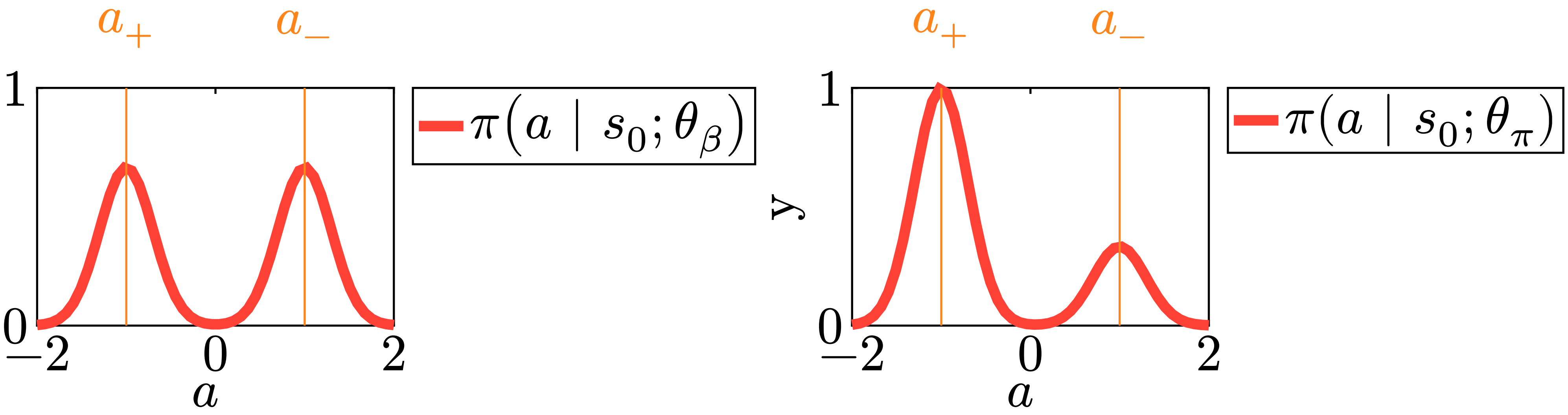


$$\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0; \theta_\beta] \qquad \hat{\mathbb{E}}[\mathcal{G}(\boldsymbol{\tau}) \mid s_0; \theta_\beta]$$

Question: How should we change $\pi(a_+ \mid s_0; \theta_\pi)$ and $\pi(a_- \mid s_0; \theta_\pi)$?

Answer: Increase probability of a_+, decrease probability of a_-

Behavioral Cloning with Rewards



We want to reweight action probabilities based on reward or return

Behavioral Cloning with Rewards

$$\arg\min_{\theta_\pi} \sum_{a \in \{a_+, a_-\}} -\pi(a \mid s_0; \theta_\beta) \log \pi(a \mid s_0; \theta_\pi)$$

Increase probability of a_+ and decrease probability of a_- using reward

Question: How? Hint:

$$\arg\min_{\theta_\pi} -\pi(a_+ \mid s_0; \theta_\beta) \log \pi(a_+ \mid s_0; \theta_\pi) - \pi(a_- \mid s_0; \theta_\beta) \log \pi(a_- \mid s_0; \theta_\pi)$$

Answer: Reweight each action in the objective using the reward

$$\arg\min_{\theta_\pi} -0.9\pi(a_+ \mid s_0; \theta_\beta) \log \pi(a_+ \mid s_0; \theta_\pi) - 0.1\pi(a_- \mid s_0; \theta_\beta) \log \pi(a_- \mid s_0; \theta_\pi)$$

Behavioral Cloning with Rewards

$$\arg\min_{\theta_\pi} -0.9\pi(a_+ \mid s_0; \theta_\beta) \log \pi(a_+ \mid s_0; \theta_\pi) - 0.1\pi(a_- \mid s_0; \theta_\beta) \log \pi(a_- \mid s_0; \theta_\pi)$$

More generally, use weights w

$$\arg\min_{\theta_\pi} -w_+\pi(a_+ \mid s_0; \theta_\beta) \log \pi(a_+ \mid s_0; \theta_\pi) - w_-\pi(a_- \mid s_0; \theta_\beta) \log \pi(a_- \mid s_0; \theta_\pi)$$

Question: What can we use for w_+, w_-?

$$\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0; \theta_\beta]$$

Behavioral Cloning with Rewards

Definition: In **Expectation Maximization Reinforcement Learning** (EMRL, Hinton), we reweight the behavior cloning objective using the reward

$$\theta_\pi = \arg\min_{\theta_\pi} \left[\underbrace{\sum_{s_0 \in \boldsymbol{X}} \sum_{a \in A} -\pi(a \mid s_0; \theta_\beta) \log \pi(a \mid s_0; \theta_\pi)}_{\text{BC objective}} \cdot \underbrace{\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a]}_{\text{Weighting}} \right]$$

Behavioral Cloning with Rewards

$$\theta_\pi = \arg\min_{\theta_\pi} \left[\sum_{s_0 \in \boldsymbol{X}} \sum_{a \in A} -\pi(a \mid s_0; \theta_\beta) \log \pi(a \mid s_0; \theta_\pi) \cdot \hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a] \right]$$

Consider the simplified example, now with rewards r

$$\arg\min_{\theta_\pi} -{\color{red}r_+} \pi(a_+ \mid s_0; \theta_\beta) \log \pi(a_+ \mid s_0; \theta_\pi) - {\color{red}r_-} \pi(a_- \mid s_0; \theta_\beta) \log \pi(a_- \mid s_0; \theta_\pi)$$

Question: Are there any problems with EMRL?

Hint: What if the reward is negative?

Reweighting only makes sense with positive weights

EMRL only works with positive rewards!

Behavioral Cloning with Rewards

$$\arg\min_{\theta_\pi} -r_+ \pi(a_+ \mid s_0; \theta_\beta) \log \pi(a_+ \mid s_0; \theta_\pi) - r_- \pi(a_- \mid s_0; \theta_\beta) \log \pi(a_- \mid s_0; \theta_\pi)$$

$$\mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] \in [-\infty, \infty]$$

We want our algorithm to work with positive and negative rewards

Need a mapping between rewards and weights $f : [-\infty, \infty] \mapsto [0, \infty]$

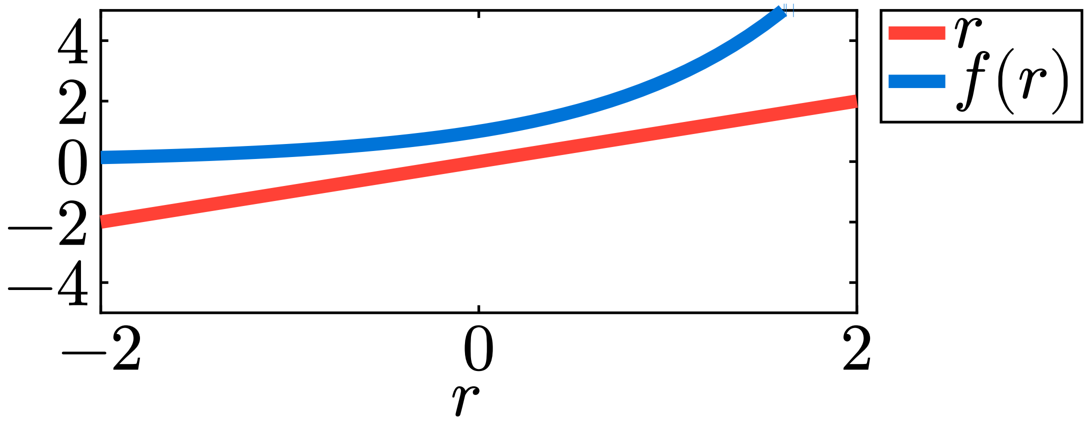
Question: Any other properties for f? Hint: Does $f(r) = |r|$ work?

Answer: f must be **monotonic** to ensure we still maximize rewards!

$$r_+ > r_- \Rightarrow f(r_+) > f(r_-)$$

Behavioral Cloning with Rewards

Question: What functions $f : [-\infty, \infty] \mapsto [0, \infty]$ are monotonic?



$$f(r) = e^r$$

Behavioral Cloning with Rewards

Definition: Reward Weighted Regression (RWR) reweights the behavior cloning objective using the exponentiated reward

$$\theta_\pi = \arg\min_{\theta_\pi} \sum_{s_0 \in \boldsymbol{X}} \sum_{a \in A} -\pi(a \mid s_0; \theta_\beta) \log \pi(a \mid s_0; \theta_\pi) \cdot \exp\left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a]\right)$$

Consider an infinitely large and diverse dataset containing all s, a

Then, weights are proportional to Boltzmann distribution (softmax)

$$\sum_{a_0 \in A} \exp\left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0]\right) \propto \sum_{a_i \in A} \frac{\exp\left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_i]\right)}{\sum_{a_0 \in A} \exp\left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0]\right)}$$

Behavioral Cloning with Rewards

$$\theta_\pi = \arg\min_{\theta_\pi} \sum_{s_0 \in \boldsymbol{X}} \sum_{a \in A} -\pi(a \mid s_0; \theta_\beta) \log \pi(a \mid s_0; \theta_\pi) \cdot \exp\left(\hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a]\right)$$

RWR find θ_π that maximizes the reward

Question: Do we maximize the reward in RL?

Answer: No, we maximize the return!

$$\theta_\pi = \arg\min_{\theta_\pi} \sum_{s_0 \in \boldsymbol{X}} \sum_{a \in A} -\pi(a \mid s_0; \theta_\beta) \log \pi(a \mid s_0; \theta_\pi) \cdot \underbrace{\exp\left(\hat{\mathbb{E}}\left[\mathcal{G}(\boldsymbol{\tau}) \mid s_0; \theta_\beta\right]\right)}_{\text{Replace reward with return}}$$

Behavioral Cloning with Rewards

$$\theta_\pi = \arg\min_{\theta_\pi} \sum_{s_0 \in \boldsymbol{X}} \sum_{a \in A} -\pi(a \mid s_0; \theta_\beta) \log \pi(a \mid s_0; \theta_\pi) \cdot \exp\left(\hat{\mathbb{E}}\left[\mathcal{G}(\boldsymbol{\tau}) \mid s_0; \theta_\beta\right]\right)$$

This works in theory, but does not work well in practice

Question: Why does this fail in practice?

Answer:

- Need infinite rewards to approximate Monte Carlo return
- Returns can be big or small, causing overflows $\exp(\mathcal{G}(\boldsymbol{\tau})) \to 0, \infty$

Question: Similar problems with actor critic, what was the solution?

- Introduce TD value function (remove infinite sum)
- Introduce advantage (normalize return)

Behavioral Cloning with Rewards

$$\theta_\pi = \arg\min_{\theta_\pi} \sum_{s_0 \in \boldsymbol{X}} \sum_{a \in A} -\pi(a \mid s_0; \theta_\beta) \log \pi(a \mid s_0; \theta_\pi) \cdot \exp\left(\hat{\mathbb{E}}\left[\mathcal{G}(\boldsymbol{\tau}) \mid s_0; \theta_\beta\right]\right)$$

$$\theta_\pi = \arg\min_{\theta_\pi} \sum_{s_0 \in \boldsymbol{X}} \sum_{a \in A} -\pi(a \mid s_0; \theta_\beta) \log \pi(a \mid s_0; \theta_\pi) \cdot \underbrace{\exp(A(s, a, \theta_\beta))}_{\text{Use advantage instead}}$$

$$A(s, a, \theta_\beta) = Q(s, a, \theta_\beta) - V(s, \theta_\beta)$$

$$A(s_t, s_{t+1}, \theta_\beta) = -V(s_t, \theta_\beta) + r_t + \gamma V(s_{t+1}, \theta_\beta)$$

Behavioral Cloning with Rewards

Definition: Monotonic Advantage Re-Weighted Imitation Learning (MARWIL) reweights the behavior cloning objective based on the advantage

Step 1: Learn a value function for θ_β

$$\theta_V = \arg\min_{\theta_V} \left(V(s_0, \theta_\beta, \theta_V) - y \right)^2$$

$$y = \hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \gamma V(s_1, \theta_\beta, \theta_V)$$

Behavioral Cloning with Rewards

Step 2: Learn policy using weighted behavioral cloning

$$A(s_t, s_{t+1}, \theta_\beta, \theta_V) = -V(s_t, \theta_\beta, \theta_V) + \hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \gamma V(s_{t+1}, \theta_\beta, \theta_V)$$

$$\theta_\pi = \arg\min_{\theta_\pi} \sum_{s \in \boldsymbol{X}} \sum_{a \in A} \underbrace{-\pi(a \mid s; \theta_\beta) \log \pi(a \mid s; \theta_\pi)}_{\text{BC objective}} \cdot \underbrace{\exp(A(s_t, s_{t+1}, \theta_\beta, \theta_V))}_{\text{Advantage reweighting}}$$

Behavioral Cloning with Rewards

Question: Is MARWIL RL or supervised learning?

Answer: Both

- Learn policy using supervised learning (SL)
- But weights require learning value function (RL)

Question: We used TD objective, can we also use MC objective?

$$V(s_0, \theta_\beta, \theta_V) = \hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0; \theta_\beta] + \gamma V(s_1, \theta_\beta, \theta_V)$$

$$V(s_0, \theta_\beta, \theta_V) = \sum_{t=0}^{\infty} \gamma^t \hat{\mathbb{E}}[\mathcal{R}(s_{t+1}) \mid s_0; \theta_\beta]$$

Answer: Yes, because we learn V for θ_β not θ_π

Behavioral Cloning with Rewards

Add improvements to MARWIL to derive other offline RL algorithms

- Advantage Weighted Regression (AWR)
- Advantage Weighted Actor Critic (AWAC)
- Critic Regularized Regression (CRR)
- Implicit Q learning (IQL)
- Maximum a Posteriori Optimization (MPO)

The Deadly Triad

The Deadly Triad

There are two standard approaches to offline RL

1. Reweight the BC loss using rewards/returns
2. Improve off-policy algorithms to work without exploration

Finished first, now let us visit the second

The Deadly Triad

Goal: Derive offline RL algorithm from an off-policy algorithm

Question: Why off-policy instead of on-policy algorithm?

On-policy requires collecting data with θ_π

- But we cannot do this! Dataset is collected with θ_β
- $\theta_\beta \neq \theta_\pi$, cannot use on-policy method

Off-policy can learn from any trajectories

- Trajectory collected following θ_β
- Can use θ_β trajectory to update θ_π

The Deadly Triad

Ok, let us choose an off-policy algorithm to use

Question: Which off-policy RL algorithms do we know?

- Q learning
- DDPG (continuous Q learning)
- Off-policy gradient (does not work well, ignore for now)

Question: Temporal Difference or Monte Carlo Q learning?

- MC is on-policy
- Only TD Q learning is off-policy

The Deadly Triad

Recall the standard Q learning algorithm

```
while not terminated:
    transition = env.step(action)
    buffer.append(transition)
    train_data = buffer.sample()
    J = grad(td_loss)(theta_Q, theta_T, Q, train_data)
    theta_Q = opt.update(theta_Q, J)
```

Question: What can we do to make this offline? Without exploration?

Answer:

- Put dataset into replay buffer
- Get rid of env code

The Deadly Triad

```python
for x in X:
    buffer.append(x) # Add dataset to replay buffer
# Comment out exploration code
# while not terminated:
    # transition = env.step(action)
    # buffer.append(transition)
for epoch in num_epochs:
    train_data = buffer.sample()
    J = grad(td_loss)(theta_Q, theta_T, Q, train_data)
    theta_Q = opt.update(theta_Q, J)
```

The Deadly Triad

```python
for x in X:
    buffer.append(x) # Add dataset to replay buffer

for epoch in num_epochs:
    train_data = buffer.sample()
    J = grad(td_loss)(theta_Q, theta_T, Q, train_data)
    theta_Q = opt.update(theta_Q, J)
```

Question: Will this work?

Answer: It can! But only for very simple problems

For harder/interesting problems, loss quickly becomes `NaN`

Let us investigate why

The Deadly Triad

On assignment 2, many of you found issues with deep Q learning

1. Q value would often become very large
2. Loss would become very large
3. Loss/parameters become `NaN`

This was not your fault, it is a known problem in RL!

Call it the **deadly triad**, because it is caused by combining three factors:

1. Function approximation (deep neural network)
2. Recursive bootstrapping (TD, $Q(s, a) = r + \gamma \max Q(s, a)$)
3. Off-policy learning/limited exploration

Let us further investigate the deadly triad

The Deadly Triad

Imagine a toy MDP

$$S = \{s\} \quad A = \{1, 2\} \quad Q(s, a, \theta_Q) = \theta_Q \cdot a \quad \mathcal{R}(s) = 0 \quad \gamma = 1$$

Can update θ_Q using simple TD update

$$\theta_Q = \theta_Q - \underbrace{\left[Q(s, a, \theta_Q) - \left(r + \gamma \max_{a' \in A} Q(s', a', \theta_Q)\right)\right]}_{\text{Q error, can consider } \nabla \mathcal{L}}$$

$$\theta_Q = \theta_Q - \left[Q(s, a, \theta_Q) - \max_{a' \in A} Q(s', a', \theta_Q)\right]$$

Importantly, to simulate off-policy/offline RL, we have limited dataset

- We only have $s, a = 1$ in our dataset, not $s, a = 2$

Let us perform some updates to θ_Q and see what happens

The Deadly Triad

Initialize $\theta_Q = 1$, update for $a = 1$

$$\theta_Q = \theta_Q - \left[Q(s, a, \theta_Q) - \max_{a' \in A} Q(s, a', \theta_Q) \right]$$

$$\theta_Q = \theta_Q - [Q(s, 1, \theta_Q) - (\max\{Q(s, 1, \theta_Q), Q(s, 2, \theta_Q)\})]$$

$$\theta_Q = \theta_Q - [Q(s, 1, \theta_Q) - (\max\{\theta_Q \cdot 1, \theta_Q \cdot 2\})]$$

$$\theta_Q = \theta_Q - [Q(s, 1, \theta_Q) - (\max\{1 \cdot 1, 1 \cdot 2\})]$$

$$\theta_Q = 1 - [1 - 2]$$

$$\theta_Q = 2$$

The Deadly Triad

Repeat with $\theta_Q = 2$, update for $a = 1$

$$\theta_Q = \theta_Q - \left[Q(s, a, \theta_Q) - \max_{a' \in A} Q(s, a', \theta_Q) \right]$$

$$\theta_Q = \theta_Q - [Q(s, 1, \theta_Q) - (\max\{Q(s, 1, \theta_Q), Q(s, 2, \theta_Q)\})]$$

$$\theta_Q = \theta_Q - [Q(s, 1, \theta_Q) - (\max\{\theta_Q \cdot 1, \theta_Q \cdot 2\})]$$

$$\theta_Q = \theta_Q - [Q(s, 1, \theta_Q) - (\max\{2 \cdot 1, 2 \cdot 2\})]$$

$$\theta_Q = 2 - [2 - 4]$$

$$\theta_Q = 4$$

The Deadly Triad

Repeat with $\theta_Q = 4$, update for $a = 1$

$$\theta_Q = \theta_Q - \left[Q(s, a, \theta_Q) - \max_{a' \in A} Q(s, a', \theta_Q) \right]$$

$$\theta_Q = \theta_Q - [Q(s, 1, \theta_Q) - (\max\{Q(s, 1, \theta_Q), Q(s, 2, \theta_Q)\})]$$

$$\theta_Q = \theta_Q - [Q(s, 1, \theta_Q) - (\max\{\theta_Q \cdot 1, \theta_Q \cdot 2\})]$$

$$\theta_Q = \theta_Q - [Q(s, 1, \theta_Q) - (\max\{4 \cdot 1, 4 \cdot 2\})]$$

$$\theta_Q = 4 - [4 - 8]$$

$$\theta_Q = 8$$

The Deadly Triad

Each time we update, θ_Q increases, even if $\mathcal{R}(s) = 0$

Larger θ_Q means larger $Q(s, a, \theta_Q)$ for both $a = 1, a = 2$

Eventually $\theta_Q \to \infty$ $\qquad$ $Q(s, a, \theta_Q) \to \infty$

Question: Why does θ_Q keep increasing?

1. (Function approximation) We share parameters for $a = 1$ and $a = 2$
   - Updating θ_Q for $a = 1$ also updates for $a = 2$
2. (Recursive bootstrap) updating θ_Q for $a = 1$ uses $\max_{a \in \{1,2\}}$ in label
3. (Off-policy) Trains on old data, does not contain $a = 2$
   - Eventually, greedy policy should visit $a = 2$

Offline RL guarantees case 3, because we will never explore

The Deadly Triad

1. (Function approximation) We share parameters for $a = 1$ and $a = 2$
   - Updating θ_Q for $a = 1$ also updates for $a = 2$
2. (Recursive bootstrap) updating θ_Q for $a = 1$ uses $\max_{a \in \{1,2\}}$ in label
3. (Off-policy) Trains on old data, does not contain $a = 2$
   - Eventually, greedy policy should visit $a = 2$

If we can prevent any of these, we can learn a Q function offline

1. Do not use neural network	Need nn for large state space
2. Use MC (non-recursive) instead of TD (recursive)	MC is on-policy only, must use TD
3. Visit all possible states/actions	Not possible with fixed dataset

Constraining Q

Constraining Q

So far, offline Q learning seems impossible

Root problem is the out-of-distribution (OOD) TD update

$$\max_{a \in A} Q(s, a); \quad (s, a) \notin \boldsymbol{X}$$

We **overextrapolate** for state-action pairs missing from dataset

Question: What can we do about this?

Answer: Ignore Q for missing state-action pairs

Constraining Q

We want to ignore $Q(s, a)$ where $s, a \notin \boldsymbol{X}$

For a finite discrete state space, this is easy!

Only consider actions we have in our dataset $\overline{A}(s)$

$$\overline{A}(s) = \{a \mid (s, a) \in \boldsymbol{X}\}$$

$$Q(s_0, a_0, \theta_Q) = \mathbb{E}[\mathcal{R}(s_1) \mid s_0, a_0] + \gamma \max_{a \in \overline{A}} Q(s_1, a, \theta_Q)$$

We ignore actions missing from the dataset!

- This will fix deadly triad via (2. Recursive bootstrap)

Question: What if the state space is continuous? Will this work?

Constraining Q

Continuous state space means we cannot check $(s, a) \in \boldsymbol{X}$

- Every state s in $\boldsymbol{X}$ will be different
  - Each state will only have one action
  - $\max_{a \in \overline{A}}$ returns the action in the dataset
  - We will learn the behavior policy, not optimal policy

Solution: Continuous relaxation of $\overline{A}$

$$\overline{A}(s) = \{a \mid \pi(a \mid s; \theta_\beta) > \rho\}$$

Only consider actions that our behavior policy might take

We can learn θ_β using behavioral cloning

Constraining Q

Definition: Batch Constrained Q Learning (BCQ) learns the behavior policy, only considers behavior policy actions in the TD update

Step 1: Learn the behavior policy through BC

$$\theta_\pi = \arg\min_{\theta_\pi} \sum_{s \in \boldsymbol{X}} \sum_{a \in A} -\pi(a \mid s; \theta_\beta) \log \pi(a \mid s; \theta_\pi)$$

Step 2: Learn the Q function

$$\theta_{Q,i+1} = \arg\min_{\theta_{Q,i}} \left(Q(s_0, a_0, \theta_\pi, \theta_{Q,i}) - y\right)^2$$

$$y = \hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \gamma \max_{a \in \overline{A}} Q(s_1, a, \theta_\pi, \theta_{Q,i})$$

Constraining Q

$$\theta_{Q,i+1} = \underset{\theta_{Q,i}}{\arg\min} \left(Q\left(s_0, a_0, \theta_\pi, \theta_{Q,i}\right) - y\right)^2$$

$$y = \hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \gamma \max_{a \in \overline{A}} Q\left(s_1, a, \theta_\pi, \theta_{Q,i}\right)$$

Constrain $\overline{A}$ to contain actions the behavior policy would take

$$\overline{A} = \{a \mid \pi(a \mid s_1; \theta_\pi) > \rho\}$$

where hyperparameter ρ determines the level of extrapolation

- Bigger ρ means less extrapolation, less optimal
- Smaller ρ means more extrapolation, more optimal
- Too much extrapolation leads to deadly triad!

Constraining Q

BCQ is a good algorithm, but it requires learning a policy using BC

Can we do offline Q learning without learning the behavior policy?

What if we make Q small for OOD actions?

$$\min_{a \notin \overline{A}} Q(s, a)$$

This still requires knowing $\overline{A}$ and use BC

Can we do this without knowing $\overline{A}$?

Constraining Q

Solution: Create two conflicting objectives:

- Learn Q as usual with TD error
- Minimize Q values

Key idea: This should force all $Q(s, a)$ to be small, unless $(s, a) \in \boldsymbol{X}$

$$\arg\min_{\theta_Q} \underbrace{\left(Q\left(s_0, a_0, \theta_\pi, \theta_Q\right) - y\right)^2}_{\text{TD error}} + \underbrace{\sum_{a \in A} Q\left(s_1, a, \theta_\pi, \theta_Q\right)}_{\text{Minimize Q}}$$

$$y = \hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \gamma \max_{a \in A} Q\left(s_1, a, \theta_\pi, \theta_{Q,i}\right)$$

Constraining Q

$$\arg\min_{\theta_Q} \underbrace{\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - y\right)^2}_{\text{TD error}} + \underbrace{\sum_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q)}_{\text{Minimize Q}}$$

$$y = \hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_{Q,i})$$

However, the scale of TD error and Q values can be different

Very sensitive to scale, we might just set all $Q(s, a) = -\infty$

We need to balance the second term a little better

Constraining Q

$$\underset{\theta_Q}{\arg\min} \ \underbrace{\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - y\right)^2}_{\text{TD error}} + \underbrace{\sum_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q)}_{\text{Minimize Q}}$$

$$y = \hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_{Q,i})$$

We can subtract Q for the action we take in the dataset!

$$\underset{\theta_Q}{\arg\min} \ \underbrace{\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - y\right)^2}_{\text{TD error}} + \underbrace{\sum_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) - Q(s_1, a_1, \theta_\pi, \theta_Q)}_{\text{Minimize Q}}$$

This balances the second term to be closer to zero

Constraining Q

$$\arg\min_{\theta_Q} \underbrace{\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - y\right)^2}_{\text{TD error}} + \underbrace{\sum_{a \in A} Q(s_1, a, \theta_\pi, \theta_Q) - Q(s_1, a_1, \theta_\pi, \theta_Q)}_{\text{Minimize Q}}$$

While minimizing all Q is useful, we care most about the biggest Q

We take $\max Q$, so we should emphasize minimizing $\max Q$

Replace sum with logsumexp, a combination of max and sum

$$\underbrace{\left(Q(s_0, a_0, \theta_\pi, \theta_Q) - y\right)^2}_{\text{TD error}} + \underbrace{\log\left(\sum_{a \in A} \exp Q(s_1, a, \theta_\pi, \theta_Q)\right) - Q(s_1, a_1, \theta_\pi, \theta_Q)}_{\text{Minimize Q}}$$

Constraining Q

Definition: Conservative Q Learning (CQL) learns a Q function that minimizes Q for out of distribution actions

$$\theta_{Q,i+1} = \arg\min_{\theta_{Q,i}} \underbrace{\left(Q(s_0, a_0, \theta_\pi, \theta_{Q,i}) - y\right)^2}_{\text{TD error}} + z^2$$

$$y = \hat{\mathbb{E}}[\mathcal{R}(s_1) \mid s_0, a_0] + \gamma \max_{a \in A} Q(s_1, a, \theta_\pi, \theta_{Q,i})$$

$$z = \underbrace{\left(\log \sum_{a \in A} \exp Q(s_1, a, \theta_\pi, \theta_{Q,i})\right)}_{\text{Minimize Q for all } a} - \underbrace{Q(s_1, a_1, \theta_\pi, \theta_{Q,i})}_{\substack{\text{Push up Q for} \\ \text{in-distribution } a_1}}$$

Conclusion

Conclusion

Today, we looked at offline RL

- Like IL, but learns optimal policy instead of expert policy

Two standard approaches to offline RL

- Behavioral cloning with weighted objectives
  - EMRL
  - RWR
  - MARWIL
- Breaking the deadly triad with Q learning
  - BCQ
  - CQL