Implementation Talk: Offline RL and Conservative Q-Learning

Aviral Kumar
Abstract: Conservative Q-Learning

“Bake the pessimism” into the Q-function

$$Q_{\text{CQL}}(s, a) := \arg \min_Q \max_{\mu} (\mathbb{E}_{s \sim D} \mathbb{E}_{a \sim \mu(s)} [Q(s, a)] - \mathbb{E}_{s, a \sim D} [Q(s, a)]) + \frac{1}{2\alpha} \mathbb{E}_{s, a, s' \sim D} [(Q(s, a) - y(s, a))^2]$$

Maximize the data Q-values

Minimize OOD Q-values

Standard TD error

Implementation in Discrete Action Settings

Reduces to a combination of TD-error + BC loss

$$
\min_\theta \alpha \left( \mathbb{E}_{s \sim D} \left[ \log \left( \sum_{a'} \exp(Q_\theta(s, a')) \right) \right] - \mathbb{E}_{s, a \sim D} [Q_\theta(s, a)] \right) + \text{TDError}(\theta; D)
$$

“standard NLL BC loss”

- Compute the log-sum-exp exactly!
- Often discretized representation of Q-values (C51) results in better training
- Use DR3 regularization to effectively leverage capacity

**DR3**: Add an explicit regularizer to minimize feature dot products!

**DR3 normalization**: Normalize features to have norm = 1

See the scaled Q-learning paper for how to use it with large networks!

K., Agarwal, Ma, Courville, Tucker, Levine. **DR3: Value-Based Deep RL Requires Explicit Regularization.** ICLR 2022

Implementation in Continuous Control

More tricky than discrete settings due to computation of log-sum-exp

\[
\log \sum_{a} \exp(Q_\theta(s, a)) - Q(s, a_{\text{data}}) \quad \rightarrow \quad \log \int_a \exp(Q_\theta(s, a)) da - Q(s, a_{\text{data}})
\]

\[
\log \int_a p(a|s) \exp(Q_\theta(s, a)) - \log p(a|s)) da = \log \mathbb{E}_{a \sim p(a|s)}[\exp(Q_\theta(s, a) - \log p(a|s))]
\]

Can be computed with samples!

Typically, CQL chooses \( p(a|s) \) to be:

\[
p(a|s) = \frac{1}{2} \pi(a|s) + \frac{1}{2} \text{Unif}(a)
\]

- 4-10 samples of actions suffice
- Can also omit \( \log p(a|s) \) if the action space is too large

Offline Hyperparameter Tuning

**Network**
Generally, use bigger networks (e.g., on D4RL tasks (256, 256) -> (512, 512, 512))

**Tuning the hyperparameter $\alpha$**

Run a sweep over a certain range of values, pick a sweet spot where TD error is small, and CQL regularizer is small.

If hard to minimize both CQL loss and TD-error, pick a larger model size!

If the CQL regularizer can be minimized to very small (or if Q-values are too small), pick a smaller model size or apply regularization!

Tuning overfitting, underfitting and checkpoint selection in this paper!

General Offline RL Recommendations

**Run SARSA first to check if basic details are fine**

- SARSA would help identify what’s going wrong irrespective of OOD actions!
- **For example**: target network update rate, size of the Q-function, discount factor

**Once SARSA passes, try to apply algorithm-specific tuning guidelines**

Conservatism $\alpha$, network capacity  
Overfitting, underfitting
What’s the Outcome?

Train a single policy on 40 Atari games

<table>
<thead>
<tr>
<th>Metric</th>
<th>DDQN vs. Baseline</th>
<th>DDQN + CQL vs. Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sessions</td>
<td>not stat sig</td>
<td>+ 0.24%</td>
</tr>
<tr>
<td>WAU</td>
<td>-0.69%</td>
<td>+ 0.18%</td>
</tr>
<tr>
<td>Volume</td>
<td>+7.72%</td>
<td>-1.73%</td>
</tr>
<tr>
<td>CTR</td>
<td>-7.79%</td>
<td>+2.26%</td>
</tr>
</tbody>
</table>

Pre-training on broad data Fine-tuning on limited, task-specific data

Real-robot pre-training and fine-tuning

Real-Time Mobile Notification Systems @ LinkedIn [Deployed]
Code References

- CQL implementation for continuous actions: [https://github.com/young-geng/JaxCQL](https://github.com/young-geng/JaxCQL)

- CQL implementation in Jax + parallelizable on TPUs + end-to-end from vision on robots: [https://github.com/Asap7772/PTR](https://github.com/Asap7772/PTR)

- CQL implementation in discrete action settings: [https://github.com/aviralkumar2907/CQL/tree/master/atari/batch_rl](https://github.com/aviralkumar2907/CQL/tree/master/atari/batch_rl)

- Parallel implementation (runs on TPUs) of scaled CQL: [https://tinyurl.com/scaled-ql-code](https://tinyurl.com/scaled-ql-code)

Thank You!