INVESTIGATION INTO SAMPLE EFFICIENCY IN RL ALGORITHMS

WEEK 02

VISWAK RB – 124104338

M.SC. DATA SCIENCE & ANALYTICS - UCC

SAC in Online Learning

SAC tries to **maximize two things** at once:

- The total reward it gets from the environment
- The amount of **randomness/ flexibility** in its actions (*Entropy*)

So, the main goal is : Maximize: ∑[Reward+α·Entropy]

Mathematically represented as –

$$J(\pi) = \sum_t \mathbb{E}_{(s_t, a_t) \sim \pi} \left[r(s_t, a_t) + \alpha \cdot \mathcal{H}(\pi(\cdot|s_t)) \right]$$

- $\pi(a|s)$: The policy — a probability of picking action a in state s
- $r(s_t, a_t)$: Reward received at time t
- $\mathcal{H}(\pi(\cdot|s_t))$: Entropy (uncertainty) of the policy at state s_t
- α: A temperature that controls **how much randomness** we want. Higher α → more exploration

Why Online Learning works better?

In **online learning**, SAC works great because it:

- Collects new data that matches its current policy π
- Keeps improving the policy using good and up-to-date information

So, policy π is learning from data that was generated by itself, the Q-values it learns are better

Soft Bellman Backup

The **Soft Bellman Backup** is the **core update rule** that SAC uses to learn the Q-values, and it reveals - Why SAC works so well when collecting its own data

$$Q_{\text{target}}(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} \left[V(s') \right]$$

Where,

$$V(s) = \mathbb{E}_{a' \sim \pi(\cdot|s)} \left[Q(s', a') - \alpha \log \pi(a'|s') \right]$$

$Q(s, a)$	Estimated total value of action a in state s
$r(s, a)$	Immediate reward from action a in state s
γ	Discount factor for future rewards (e.g. 0.99)
s'	Next state after taking action a

This further confirms **Why Online SAC Works Well**

- In **online training**, the agent is constantly gathering data using its **current policy** π.
- This means the actions a' used in the soft Bellman backup **actually exist** in the replay buffer.
- So, the Q-values it learns are grounded in **real, seen transitions**.
- This makes the estimate of future rewards (the Q-values) **accurate and stable**.

Why SAC Struggles in Offline?

1. In **offline RL**, we train using a fixed dataset D={(s,a,r,s′)} **collected by another policy**, not by interacting with the environment. This causes **overestimation of Q Values**

The dataset policy **β(a|s) is different from the current policy π(a|s)** :

- Actions in buffer were chosen by β
- But SAC updates using π, which may assign high probability to **out-of-distribution (OOD)** actions
- This causes Q-values to be overestimated.

Why SAC Struggles in Offline? (Cont.)

2. **Unseen Actions :**

- SAC samples actions $a' \sim \pi(a|s')$ during learning.
- But in offline RL, these actions **may not exist** in the dataset.
- When the Q-function tries to evaluate $Q(s', a')$, it tries random guesses.

3. This could be even worsened by **Entropy Term ($-\alpha \log \pi(a|s)$),** as it can push the policy toward diverse actions not supported by the dataset

Why SAC Works Across Many Environments?

1. SAC maximizes **both expected reward and entropy**:

$$J(\pi) = \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\right]$$

- α: temperature parameter controlling exploration vs exploitation.
- $\mathcal{H}(\pi) = -\mathbb{E}_{a\sim\pi}[\log\pi(a|s)]$: entropy of the policy.

This encourages **diverse actions**, improving exploration in **continuous and sparse reward** environments.

2. Soft Bellman Backup Stabilizes Learning

$$Q(s_t,a_t) = r(s_t,a_t) + \gamma\,\mathbb{E}_{s_{t+1}\sim p}\left[V(s_{t+1})\right]$$

with soft value function,

$$V(s_{t+1}) = \mathbb{E}_{a_{t+1}\sim\pi}\left[Q(s_{t+1},a_{t+1}) - \alpha\log\pi(a_{t+1}|s_{t+1})\right]$$

This **smooths out overestimation errors** and helps convergence.

Why SAC Works Across Many Environments?

3. Off-Policy and Sample Efficient

- SAC is **off-policy**, meaning it reuses past transitions via a **replay buffer**.
- This improves **sample efficiency**

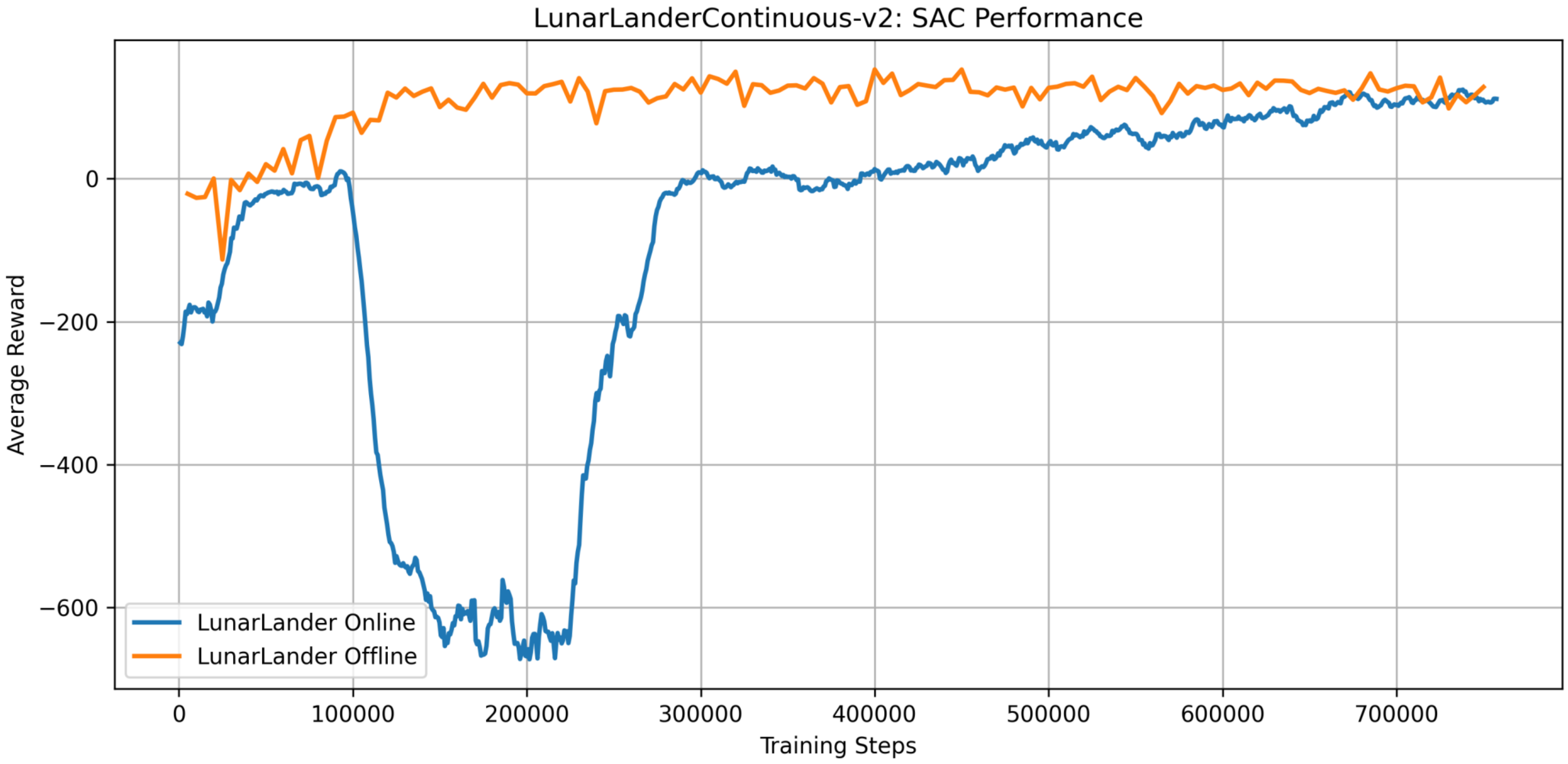
$$\mathcal{D} = \{(s_i, a_i, r_i, s'_i, d_i)\}_{i=1}^{N}$$

replay buffer D stores past experience tuples

SAC is more generalized as it can handle

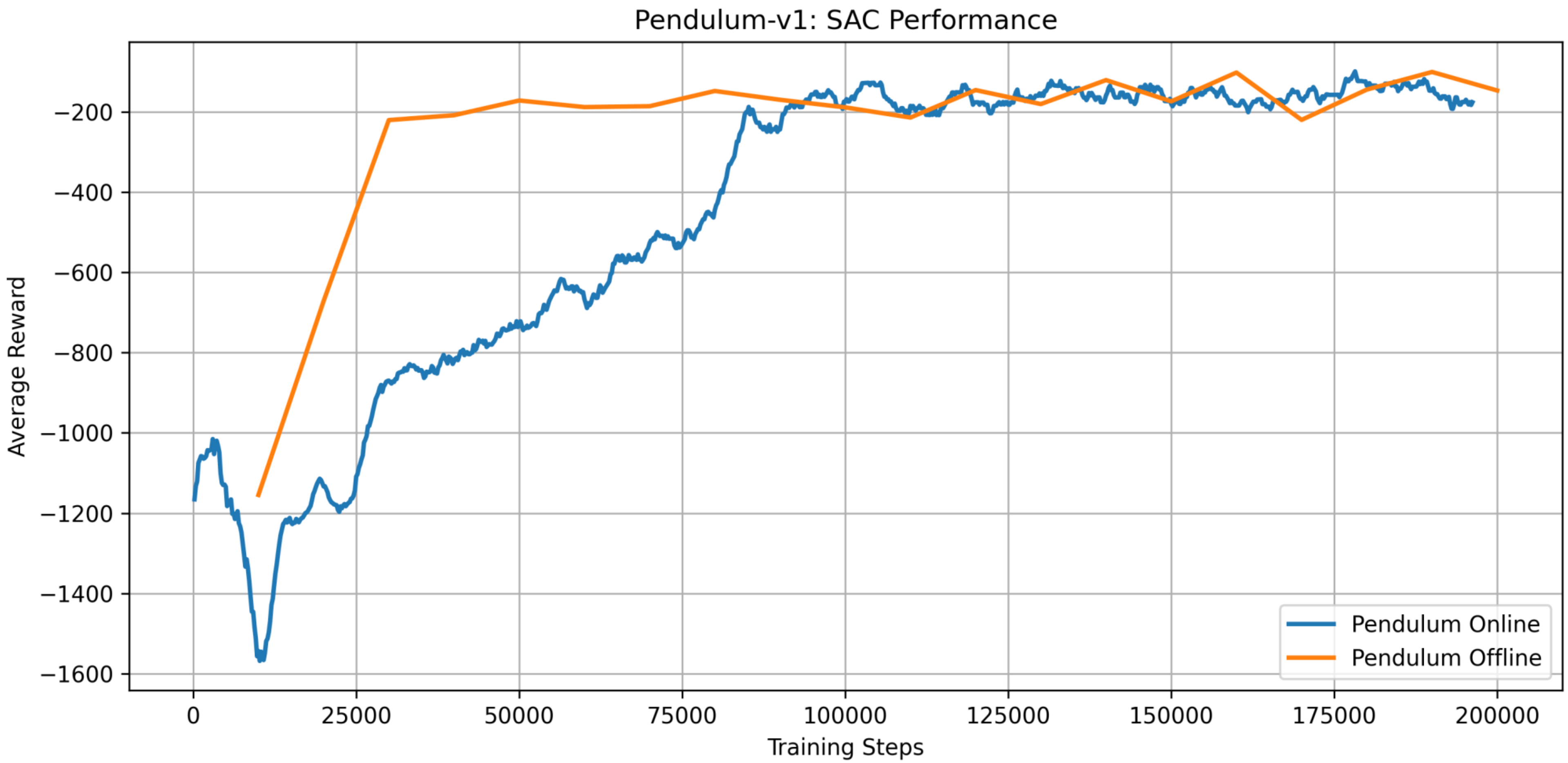
Continuous action spaces (e.g., Pendulum, LunarLanderContinuous), High-dimensional observations, Stochastic or deterministic dynamics

Results – Lunar Lander SAC



- **Offline SAC achieved high reward early**, indicating it learned efficiently from the dataset.
- **Online SAC showed unstable learning** with a major dip mid-training, typical of high exploration.
- **Offline was more sample efficient**, achieving convergence in ~200K steps vs ~700K+ for online.
- With longer training and better data, **offline SAC not only caught up but converged faster** than online.

Results – Pendulum SAC



- **Offline SAC converged much faster**, reaching strong performance in ~30K steps.
- **Online SAC started poorly** and required over 100K steps to catch up.
- **Both methods achieved similar final rewards**, showing SAC's robustness.
- **Offline SAC was more sample efficient**, learning effectively from a high-quality dataset.

SAC Sample Efficiency

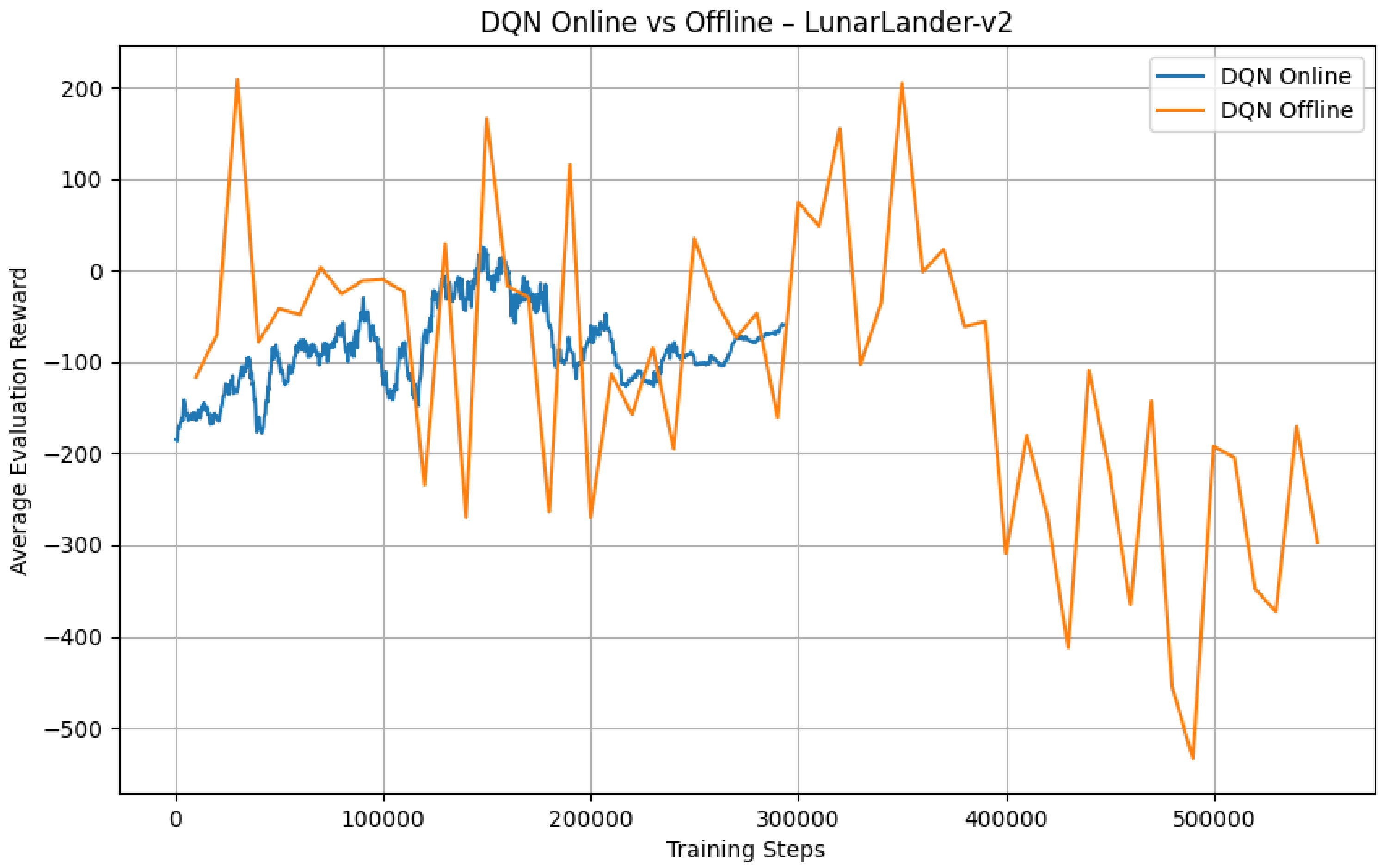
Configuration	Total Steps to Converge	Steps to 80% of Final Reward	Sample Efficiency	Interpretation
LunarLander (Online)	~1000 episodes (~772,000 steps)	~105 episodes (~550,000 steps)	Low	Learns slowly despite high sample budget.
LunarLander (Offline)	~150 episodes (~300,000 steps)	~24 episodes (~200,000 steps)	Efficient	Faster learning with fewer samples.
Pendulum (Online)	~1000 episodes (~200,000 steps)	~424 episodes (~90,000 steps)	Low	Slow convergence, needs more samples.
Pendulum (Offline)	~20 episodes (~30,000 steps)	~14 episodes (~20,000 steps)	Efficient	Quick and stable learning with few samples.

** Steps to 80% of final reward refers to the number of steps needed to reach 80% of the final average performance, indicating early learning efficiency.*

Applying DQN: Constraints & Experimental Use

- **DQN (Deep Q-Network)** is designed for **discrete action spaces** and therefore **not directly compatible** with continuous control environments like Pendulum-v1 or LunarLanderContinuous-v2, where actions are real-valued.
- To explore and benchmark DQN **purely for experimental comparison**, we used the **discrete variant** — LunarLander-v2.
- **Note:** This comparison is not strictly fair or fully representative, as SAC was evaluated on continuous action spaces and DQN on a discrete one. Hence, results may reflect **differences in environment complexity**, not just algorithm performance.

Results – LunarLander DQN



- **Both Online and Offline DQN** show unstable learning curves, with no clear convergence throughout training.
- **Online DQN** exhibits some short-term improvement but fluctuates significantly, hovering around rewards of -100 without sustained gains.
- **Offline DQN** is highly erratic, with extreme spikes and drops in performance — indicating poor generalization from static data.

Key papers – on why SAC struggles in Offline RL

1. Off-Policy Deep Reinforcement Learning without Exploration – Fujimoto et al., ICML 2019
This foundational paper introduces the problem of **extrapolation error** in offline RL — SAC and other off-policy algorithms may assign high value to unseen (out-of-distribution) actions, causing poor policies. The authors propose BCQ (Batch-Constrained Q-learning) to restrict actions to those seen in the dataset, which greatly improves stability and performance in offline settings.
Read here

2. Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction (BEAR) – Kumar et al., NeurIPS 2019
BEAR highlights **bootstrapping error** — the repeated use of inaccurate value estimates for unseen actions in SAC's critic leads to divergence. It proposes constraining the learned policy to remain close to the data distribution using a similarity metric, effectively improving stability.
Read here

3. Behavior Regularized Offline Reinforcement Learning (BRAC) – Wu et al., 2019 (Google Research)
BRAC shows that SAC fails offline mainly due to **unconstrained policy deviation**. By adding a regularization term (e.g., KL divergence) between the learned and behavior policy, SAC becomes significantly more stable and effective in offline training.
Read here

4. Conservative Q-Learning for Offline Reinforcement Learning (CQL) – Kumar et al., NeurIPS 2020
CQL addresses SAC's offline failure by making Q-learning **conservative** — penalizing Q-values of actions not in the dataset. This reduces overestimation and prevents the agent from exploiting erroneous Q-values for out-of-distribution actions.
Read here

Key papers – on why SAC struggles in Offline RL

5. A Minimalist Approach to Offline Reinforcement Learning (TD3+BC) – Fujimoto & Gu, NeurIPS 2021
This simple approach shows that just adding a **behavior cloning loss** to SAC or TD3's policy update significantly improves offline performance. It confirms that SAC mainly fails offline due to its policy choosing actions too far from those in the dataset.
Read here

6. Offline Reinforcement Learning with Implicit Q-Learning (IQL) – Kostrikov et al., ICLR 2022
IQL avoids the failure of SAC in offline RL by **never querying out-of-distribution actions**. It trains a value function using expectile regression and performs advantage-weighted behavior cloning, bypassing the pitfalls of standard SAC's critic updates.
Read here

7. Uncertainty-Based Offline Reinforcement Learning with Diversified Q-Ensemble (EDAC) – An et al., NeurIPS 2021
EDAC enhances SAC by using a **Q-network ensemble** to estimate uncertainty and conservatively penalize high-variance value predictions. This helps mitigate SAC's tendency to overestimate Q-values for out-of-distribution actions.
Read here

8. SpOiLer: Offline Reinforcement Learning using Scaled Penalties – Srinivasan & Knottenbelt, PMLR 2024
SpOiLer modifies SAC's Bellman backups by adding a **penalty proportional to the action's likelihood under the dataset**, making value estimates more pessimistic for unfamiliar actions. This method avoids overestimation without needing ensembles or behavior cloning.

Read here

References

- Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018).
  Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.
  arXiv preprint arXiv:1801.01290.
  🔗 https://arxiv.org/abs/1801.01290

- Haarnoja, T., Zhou, A., Tucker, G., & Levine, S. (2019).
  Soft Actor-Critic Algorithms and Applications.
  arXiv preprint arXiv:1812.05905.
  🔗 https://arxiv.org/abs/1812.05905

- Levine, S., Kumar, A., Tucker, G., & Fu, J. (2020).
  Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems.
  arXiv preprint arXiv:2005.01643.
  🔗 https://arxiv.org/abs/2005.01643

References

- Fu, J., Kumar, A., Nachum, O., Tucker, G., & Levine, S. (2020).
  D4RL: Datasets for Deep Data-Driven Reinforcement Learning.
  arXiv preprint arXiv:2004.07219.
  🔗 https://arxiv.org/abs/2004.07219
- Sutton, R. S., & Barto, A. G. (2018).
  Reinforcement Learning: An Introduction (2nd ed.).
  MIT Press.
  🔗 http://incompleteideas.net/book/the-book-2nd.html
- OpenAI Spinning Up.
  Spinning Up in Deep RL Documentation – Soft Actor-Critic.
  OpenAI, 2018.
  🔗 https://spinningup.openai.com/en/latest/algorithms/sac.html