ESE 6510 Notes: Issues in Q-Learning and SAC Algorithm

Vaibhav Thakkar

Contents

Issues with Fitted Q-Iteration

While Fitted Q-Iteration is powerful, using neural networks with the Bellman operator introduces several practical challenges:

Issue 1: Stabilizing the Target Network

Problem

The target $y_i = r_i + \gamma \max_{a'} Q_\phi(s'_i, a')$ uses *the same parameters* ϕ that we are updating. This creates a *moving target* and can cause instability or divergence.

Hack 1 – Use a separate target network:
Maintain two networks with parameters ϕ and ϕ':

- ϕ is the *current* network, updated every step
- ϕ' is the *target* network, updated slowly (frozen for K steps or via Polyak averaging)

The target is then:

$$y_i = r_i + \gamma \max_{a'} Q_{\phi'}(s'_i, a')$$

Polyak (exponential moving average) update:

$$\phi' \leftarrow \beta\, \phi' + (1 - \beta)\, \phi, \quad \beta \approx 0.999$$

This provides a *slowly-moving* regression target, greatly stabilizing training.

Issue 2: Overestimation Bias in Q-Learning

Problem: Overestimation

The max operator in the Bellman target causes systematic **overestimation** of Q-values.

Why does overestimation occur?
Suppose $\hat{Q}(s, a)$ is an unbiased but noisy estimate of the true $Q^*(s, a)$. Then:

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

The maximum tends to select actions with positive estimation noise.

Hack 2 – Minimum (Clipped) Q-Learning:
Maintain *two independent Q-functions* and use their minimum in the Bellman target:

$$y_i = r_i + \gamma \min_{k=1,2} \max_{a'} Q_{\phi'_k}(s'_i, a')$$

- Each critic has independent approximation errors
- Taking the minimum introduces a small pessimistic bias
- This pessimism is far less harmful than overestimation

This idea is known as *minimum clipping* and is the core stabilization trick later used in SAC.

Hack 3 – Double Q-Learning:
Decouple **action selection** and **action evaluation** using two networks:

$$y_i = r_i + \gamma\, Q_{\phi_2}\left(s'_i,\ \arg\max_{a'} Q_{\phi_1}(s'_i, a')\right)$$

In practice with a target network:

- ϕ_1 selects the action
- ϕ_2 evaluates it

This reduces—but does not eliminate—overestimation.

Issue 3: Correlated Samples (Replay Buffer)

Problem: Sample Correlation

Consecutive transitions (s_t, a_t, s_{t+1}) are highly correlated. Training on sequential batches violates the i.i.d. assumption of SGD and leads to poor, unstable updates.

Hack 3 – Experience Replay Buffer:

- Store transitions (s, a, r, s') in a large replay buffer $\mathcal{B}$
- At each update step, sample a *random mini-batch* from $\mathcal{B}$
- Breaks temporal correlations, improves data efficiency, enables off-policy reuse

Issue 4: Poor Exploration

Problem: Insufficient Exploration

ε-greedy and similar exploration strategies are not sophisticated enough to explore structured environments, especially in robotics.

Hack 4 – Noise for Exploration:

Add noise to actions during data collection:

$$a_t = \arg\max_a Q_\phi(s_t, a) + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma^2)$$

Or in the parameter space (parameter noise / NoisyNets).

Issue 5: Not Directly Maximizing the RL Objective

Problem

Q-learning minimizes a surrogate Bellman residual loss, **not** the true RL objective $\mathbb{E}_\tau[R(\tau)]$. There is no convergence guarantee with function approximation.

Soft Actor-Critic (SAC)

Motivation

Standard Q-learning and actor-critic methods have several shortcomings in continuous action spaces:

- **Hard to explore:** pure reward maximization leads to deterministic, brittle policies
- **Sample inefficiency:** on-policy methods need fresh data after each update
- **Convergence issues:** Q-learning with function approximation can diverge

Key Insight – Maximum Entropy Reinforcement Learning:
Instead of only maximizing reward, also maximize the *entropy* of the policy:

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

where $\mathcal{H}(\pi(\cdot|s)) = -\mathbb{E}_{a\sim\pi}[\log \pi(a|s)]$ is the Shannon entropy and $\alpha > 0$ is the *temperature* controlling the exploration/exploitation trade-off.

Why Maximum Entropy?

- Encourages the agent to explore all promising actions, not just the single best
- Makes the policy more robust to model errors and distributional shift
- Prevents premature convergence to suboptimal deterministic policies

Soft Bellman Equations (Q-only Formulation)

Soft Q-function:

$$Q_{\text{soft}}(s,a) = r(s,a) + \gamma\, \mathbb{E}_{s'\sim p,\, a'\sim\pi} \left[Q_{\text{soft}}(s',a') - \alpha \log \pi(a'|s') \right]$$

- The entropy term is absorbed directly into the Q-target
- No separate $V_{\text{soft}}(s)$ network is required
- This simplifies implementation and improves numerical stability

The policy is optimized to maximize this soft Q-function, balancing reward and entropy automatically.

SAC Architecture

SAC maintains three networks:

1. **Policy network** $\pi_\theta(a|s)$: a squashed Gaussian $a = \tanh(\mu_\theta(s) + \sigma_\theta(s)\cdot\varepsilon)$, $\varepsilon \sim \mathcal{N}(0, I)$
2. **Two Q-networks** $Q_{\phi_1}(s,a)$, $Q_{\phi_2}(s,a)$: mitigate overestimation via "clipped double-Q"
3. **Two target Q-networks** $Q_{\phi_1'}$, $Q_{\phi_2'}$: slowly updated copies for stable targets

SAC Objective Functions

Q-function loss (minimize Bellman residual for both critics):

$$\mathcal{L}_Q(\phi_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{B}} \left[\left(Q_{\phi_i}(s,a) - y \right)^2 \right]$$

where the target y uses *clipped double-Q* to reduce overestimation:

$$y = r + \gamma \left(\min_{i=1,2} Q_{\phi_i'}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s') \right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

Policy loss (maximize Q + entropy):

$$\mathcal{L}_\pi(\theta) = \mathbb{E}_{s\sim\mathcal{B},\,\varepsilon\sim\mathcal{N}} \left[\alpha \log \pi_\theta(\tilde{a}_\theta(s,\varepsilon)|s) - \min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s,\varepsilon)) \right]$$

where $\tilde{a}_\theta(s,\varepsilon) = \tanh(\mu_\theta(s) + \sigma_\theta(s) \odot \varepsilon)$ (reparameterization trick).

SAC Algorithm (Pseudocode)

Algorithm 1 Soft Actor-Critic (SAC)

Initialize:

- Policy π_θ, critics Q_{ϕ_1}, Q_{ϕ_2}
- Target critics $Q_{\phi'_1} \leftarrow Q_{\phi_1}$, $Q_{\phi'_2} \leftarrow Q_{\phi_2}$
- Replay buffer $\mathcal{B} \leftarrow \emptyset$
- Temperature α (fixed hyperparameter)

for each environment step **do**

  Sample $a_t \sim \pi_\theta(\cdot|s_t)$

  Execute a_t, observe r_t, s_{t+1}

  Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{B}$

end for

for each gradient step (every G environment steps) **do**

  Sample mini-batch $\{(s, a, r, s')\}_{i=1}^{N}$ from $\mathcal{B}$

  Compute targets:

$$\tilde{a}' \sim \pi_\theta(\cdot|s'), \quad y = r + \gamma \left(\min_{i=1,2} Q_{\phi'_i}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s') \right)$$

  Update critics:

$$\phi_i \leftarrow \phi_i - \lambda_Q \nabla_{\phi_i} \frac{1}{N} \sum_j (Q_{\phi_i}(s_j, a_j) - y_j)^2, \quad i = 1, 2$$

  Update policy:

$$\tilde{a}_\theta \leftarrow \mu_\theta(s) + \sigma_\theta(s) \odot \varepsilon, \ \varepsilon \sim \mathcal{N}(0, I)$$

$$\theta \leftarrow \theta - \lambda_\pi \nabla_\theta \frac{1}{N} \sum_j \left(\alpha \log \pi_\theta(\tilde{a}|s_j) - \min_i Q_{\phi_i}(s_j, \tilde{a}) \right)$$

  Soft update target networks:

$$\phi'_i \leftarrow \tau \, \phi_i + (1 - \tau) \, \phi'_i, \quad \tau \approx 0.005$$

end for

SAC: Key Design Choices and Intuitions

Squashed Gaussian Policy

The policy outputs $a = \tanh(\mu + \sigma\varepsilon)$ for $\varepsilon \sim \mathcal{N}(0, I)$.
The tanh squashing ensures bounded actions (required for many control tasks).
Log-probabilities require a correction:

$$\log \pi(a|s) = \log \mathcal{N}(\operatorname{arctanh}(a); \mu, \sigma) - \sum_d \log(1 - a_d^2)$$

Clipped Double-Q Trick

Taking $\min(Q_{\phi_1}, Q_{\phi_2})$ for both the policy and target computation counteracts overestimation bias that comes from the max inside Q-learning targets.

Why Q-Learning is Rarely Used in Robotics

Overview

Despite theoretical elegance, Q-learning (and DQN variants) face fundamental challenges in real-world robotics:

1. Requires Stabilization Tricks

Q-learning requires extensive engineering to work:

- Target networks (separate slow/fast copies)
- Experience replay buffers
- Careful hyperparameter tuning

None of these have fundamental solutions – they are heuristic patches.

2. Maximizing Over Actions is Non-Trivial

$$a^* = \arg\max_{a \in \mathcal{A}} Q_\phi(s, a)$$

- For discrete actions: easy (enumerate all)
- For continuous actions: requires inner optimization loop at every step – very expensive

3. Slow to Stabilize – Many Tricks Needed

1. Takes a lot of care to stabilize (moving targets)
2. Not really maximizing the RL objective; proxy loss (TD error)
3. Much more computationally expensive per sample
4. It takes a lot of samples to stabilize the algorithm

4. Robotics-Specific Challenges

Continuous state/action spaces:

- Continuous state space: we can fit $V(s)$ with a NN, but must know dynamics for FVI
- If we don't know dynamics: fit Q directly (FQI), but now $\arg\max_a Q(s, a)$ is expensive
- High-dimensional sensorimotor spaces make exploration extremely hard

What roboticists use instead:

- **SAC** (Soft Actor-Critic) – off-policy, continuous actions, entropy bonus
- **PPO / TRPO** – stable, if simulated resets are cheap
- **Model-based RL** – learn dynamics, plan with it (much more sample-efficient)
- **Imitation learning / RLHF** – start from expert demonstrations

Bottom Line on Q-Learning in Robotics

- Off-policy nature is attractive (data reuse)
- But instability, overestimation, exploration, and continuous action problems severely limit practicality
- SAC solves many of these but adds entropy regularization machinery
- Simulation + transfer (sim-to-real) is currently the dominant paradigm