Section 4: DQN and Soft Actor-Critic

1 Part 0: Q-Learning Refresher

Goal: Learn the optimal action-value function $Q^*(s, a)$: the expected discounted return starting from state s, taking action a, then acting optimally.

Bellman Optimality Equation:

$$Q^*(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[\max_{a'} Q^*(s', a') \right]$$

Tabular Q-Learning Update:

Given a transition (s, a, s') and reward r, update:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[\underbrace{r + \gamma \max_{a'} Q(s', a')}_{\text{target}} - Q(s, a) \right]$$

From Tables to Neural Networks:

When state spaces are large (e.g., images), we can't store a table. Instead, we approximate Q with a neural network $Q_\theta(s, a)$.

Fitted Q-Learning Loss (squared TD error):

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,s') \sim \mathcal{D}} \left[\left(Q_\theta(s, a) - \left(r + \gamma \max_{a'} Q_\theta(s', a') \right) \right)^2 \right]$$

The Problem: Naive deep Q-learning is unstable!

Two sources of instability:

1. **Correlated samples:** Sequential transitions $(s_t, s_{t+1}, s_{t+2}, \ldots)$ are highly correlated
2. **Moving target:** The target $r + \gamma \max_{a'} Q_\theta(s', a')$ changes as we update θ

These problems motivate the DQN tricks in the next section.

2 Part 1: Deep Q-Networks (DQN)

DQN (Mnih et al., 2015) introduced two key ideas to stabilize deep Q-learning: **experience replay** and **target networks**. We also cover **Double DQN** (van Hasselt et al., 2016), a popular extension that addresses overestimation bias.

2.1 Trick 1: Experience Replay

Problem: Training on sequential data causes correlated gradients → unstable learning.

Solution: Store transitions in a replay buffer $\mathcal{D}$ and sample random mini-batches.

Replay Buffer:
1. Collect transition (s, a, s') and reward r during interaction
2. Store in buffer $\mathcal{D}$ (fixed size, FIFO)
3. Sample random mini-batch from $\mathcal{D}$ for each gradient step

Benefits:

- Breaks correlation between consecutive samples
- Reuses data efficiently (each transition used multiple times)
- Reduces non-stationarity of training data

2.2 Trick 2: Target Networks

Problem: The target $y = r + \gamma \max_{a'} Q_\theta(s', a')$ keeps changing as we update θ.

Solution: Use a separate *target network* $Q_{\bar{\theta}}$ that updates slowly.

DQN Loss with Target Network:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\left[\left(Q_\theta(s,a) - \underbrace{\left(r + \gamma \max_{a'} Q_{\bar{\theta}}(s', a')\right)}_{\text{computed with frozen } \bar{\theta}}\right)^2\right]$$

Two ways to update target network:

Method	Update Rule	Typical Value
Hard update	$\bar{\theta} \leftarrow \theta$ every N steps	$N = 10000$
Soft update (Polyak)	$\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$ each step	$\tau = 0.005$

2.3 Extension: Double Q-Learning

Double DQN (van Hasselt et al., 2016) addresses a known issue with Q-learning.

Problem: Q-learning overestimates Q-values due to the max operator.

Overestimation bias:

$$\mathbb{E}\left[\max_{a'} Q(s', a')\right] \geq \max_{a'} \mathbb{E}\left[Q(s', a')\right]$$

Using the same network to *select* and *evaluate* the best action amplifies noise.

Solution: Use online network to *select* action, target network to *evaluate*.

Double DQN Target:

$$y = r + \gamma\, Q_{\bar{\theta}}\left(s',\ \underbrace{\arg\max_{a'} Q_\theta(s', a')}_{\text{selected by online network}}\right)$$

2.4 Exploration: ϵ-Greedy

ϵ-Greedy Policy:

$$a = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a Q_\theta(s, a) & \text{with probability } 1 - \epsilon \end{cases}$$

Common schedule: Decay ϵ from 1.0 to 0.01 over first N environment steps.

2.5 Full DQN Algorithm

DQN Algorithm:

1. Initialize replay buffer $\mathcal{D}$, Q-network Q_θ, target network $Q_{\bar{\theta}} \leftarrow Q_\theta$
2. **For each step:**
   (a) Select action a using ϵ-greedy w.r.t. Q_θ
   (b) Execute a, observe r, s'; store (s, a, s') and r in $\mathcal{D}$
   (c) Sample mini-batch from $\mathcal{D}$
   (d) Compute target: $y = r + \gamma \max_{a'} Q_{\bar{\theta}}(s', a')$
   (e) Update θ by minimizing $(Q_\theta(s, a) - y)^2$
   (f) Update target network (hard or soft)

Implementation tip: Terminal States.

In practice, episodes terminate, and we must handle terminal states correctly.

The done flag: Store a flag d with each transition, where $d = 1$ if s' is terminal and $d = 0$ otherwise.
Target:

$$y = r + \gamma(1 - d) \max_{a'} Q_{\bar{\theta}}(s', a')$$

The $(1 - d)$ term ensures we don't bootstrap from terminal states (where there is no future return).

3 Part 2: Soft Actor-Critic (SAC)

DQN works for discrete actions. For **continuous actions**, we need a different approach.

3.1 From Q-Learning to Actor-Critic

Problem with continuous actions: Can't compute $\max_{a'} Q(s', a')$ exactly.

Solution: Learn a *policy* (actor) $\pi_\phi(a|s)$ alongside the Q-function (critic).

Actor-Critic Setup:

- **Critic** $Q_\theta(s, a)$: estimates expected return
- **Actor** $\pi_\phi(a|s)$: outputs actions (replaces arg max)

3.2 Maximum Entropy RL

SAC adds an **entropy bonus** to encourage exploration:

Maximum Entropy Objective:

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

where $\mathcal{H}(\pi(\cdot|s)) = -\mathbb{E}_{a\sim\pi(\cdot|s)}[\log \pi(a|s)]$ is the conditional entropy.

Why entropy regularization?

- Encourages exploration (don't collapse to deterministic policy too early)
- Makes learning more robust (captures multiple good solutions)
- Improves convergence in practice

The temperature α controls exploration vs. exploitation.

3.3 SAC Components

1. Stochastic Policy with Reparameterization

SAC uses a Gaussian policy:

Reparameterization Trick:

$$a = \tanh(\mu_\phi(s) + \sigma_\phi(s) \odot \epsilon), \quad \epsilon \sim \mathcal{N}(0, I)$$

The tanh squashes actions to $[-1, 1]$. The reparameterization allows backprop through sampling.

2. Clipped Double-Q (from TD3)

To reduce overestimation, SAC uses **two** Q-networks and takes the minimum:

Clipped Double-Q Target:

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

3. Automatic Temperature Tuning

Instead of manually setting α, SAC learns it by minimizing:

$$\mathcal{L}_\alpha = \mathbb{E}_{s\sim\mathcal{D},\, a\sim\pi_\phi(\cdot|s)} \left[-\alpha \left(\log \pi_\phi(a|s) + \mathcal{H}_{\text{tgt}} \right) \right]$$

where $\mathcal{H}_{\text{tgt}}$ is the target entropy (typically $-\dim(\mathcal{A})$).

Implementation tip: Optimize $\log \alpha$ instead of α directly to ensure $\alpha > 0$.

3.4 SAC Losses Summary

Critic Loss (for each Q_{θ_i}, $i \in \{1, 2\}$):

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

(This is the squared TD error.)

Actor Loss:

$$\mathcal{L}_\pi(\phi) = \mathbb{E}_{s\sim\mathcal{D},\, a\sim\pi_\phi(\cdot|s)} \left[\alpha \log \pi_\phi(a|s) - \min_{i=1,2} Q_{\theta_i}(s, a) \right]$$

Temperature Loss:

$$\mathcal{L}_\alpha = \mathbb{E}_{s\sim\mathcal{D},\, a\sim\pi_\phi(\cdot|s)} \left[-\alpha \left(\log \pi_\phi(a|s) + \mathcal{H}_{\text{tgt}} \right) \right]$$