**Step 1: Initialize**

1. Start by initializing:
   * **Actor (policy network):** A neural network that maps states to actions.
   * **Critic (Q-function network):** Two neural networks that estimate the value of taking an action in a state. (SAC uses two critics to reduce overestimation bias.)
   * **Temperature parameter (α):** This controls how much importance is given to entropy. Higher α means more focus on exploration.
2. Create a replay buffer:
   * This is a memory where past experiences (state, action, reward, next state) are stored. It helps the agent learn from past experiences instead of relying only on the current one.

**Step 2: Interact with the Environment**

1. The agent observes the **current state** from the environment (e.g., the position of a robot in a room).
2. The actor chooses an action based on its policy. This action is **stochastic** (randomness is added to encourage exploration).
3. The environment responds with:
   * A **reward** for the action.
   * The **next state** the agent transitions to.
4. Store this experience (state, action, reward, next state) in the replay buffer.

**Step 3: Update the Critic (Value Function)**

1. Sample a batch of experiences from the replay buffer.
2. Compute the **target Q-value** using the formula:

Qtarget=r+γ⋅(min⁡(Q1′,Q2′)−α⋅log(π(a′∣s′)))Q\_{\text{target}} = r + \gamma \cdot \left( \min(Q\_1', Q\_2') - \alpha \cdot \text{log}(\pi(a'|s')) \right)Qtarget​=r+γ⋅(min(Q1′​,Q2′​)−α⋅log(π(a′∣s′)))

* + rrr: Reward received.
  + γ\gammaγ: Discount factor (how much future rewards are valued).
  + min⁡(Q1′,Q2′)\min(Q\_1', Q\_2')min(Q1′​,Q2′​): Minimum of the two Q-value estimates from the target critic networks (to avoid overestimation).
  + α⋅log(π(a′∣s′))\alpha \cdot \text{log}(\pi(a'|s'))α⋅log(π(a′∣s′)): Entropy term, encouraging exploration.

1. Update the critic networks by minimizing the difference between their predicted Q-values and the target Q-value.

**Step 4: Update the Actor (Policy Network)**

1. Update the policy to maximize the expected reward and entropy:

πnew←arg⁡max⁡(Q(s,a)−α⋅log(π(a∣s)))\pi\_{\text{new}} \gets \arg\max \left(Q(s, a) - \alpha \cdot \text{log}(\pi(a|s))\right)πnew​←argmax(Q(s,a)−α⋅log(π(a∣s)))

* + This step ensures the policy selects actions that are not only rewarding but also diverse (high entropy).

1. Use gradient descent to update the policy network.

**Step 5: Adjust the Entropy Temperature (α\alphaα)**

1. SAC allows α\alphaα (the temperature) to be learned automatically.
2. Update α\alphaα to control the trade-off between exploration and exploitation: α←α−η⋅(log(π(a∣s))+Htarget)\alpha \gets \alpha - \eta \cdot (\text{log}(\pi(a|s)) + H\_{\text{target}})α←α−η⋅(log(π(a∣s))+Htarget​)
   * HtargetH\_{\text{target}}Htarget​: Target entropy, which defines the desired level of randomness.

**Step 6: Repeat**

1. Repeat Steps 2–5 for many episodes until the agent learns a good policy.
2. Over time, the agent's actions become less random and more focused on maximizing rewards.