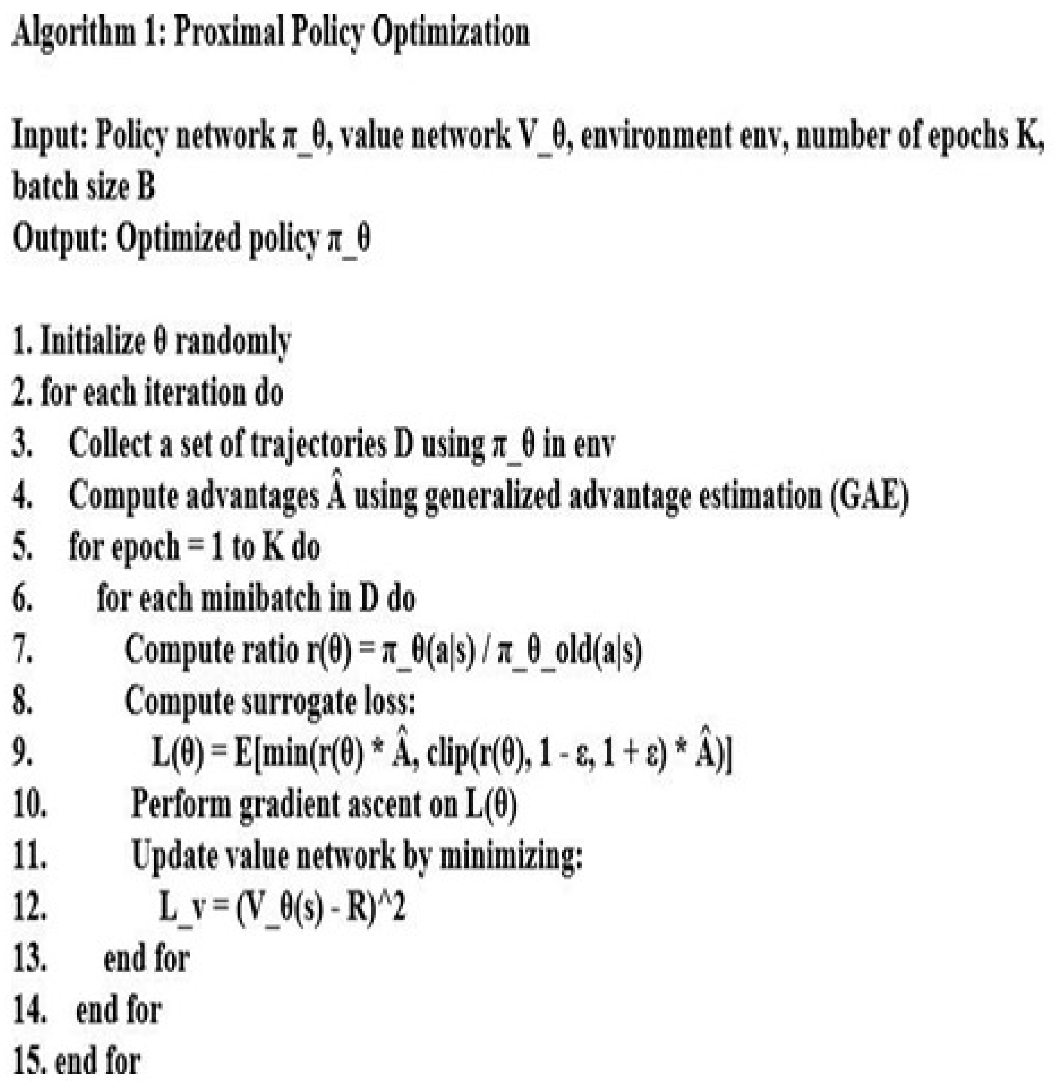
Proximal Policy Optimization (PPO)

PPO is a policy gradient technique that balances stability and performance. To avoid significant policy changes that would cause training to become unstable, it employs a clipped surrogate aim. PPO is one of the most reliable contemporary reinforcement learning algorithms since it increases learning stability and efficiency.



Algorithm 1: Proximal Policy Optimization

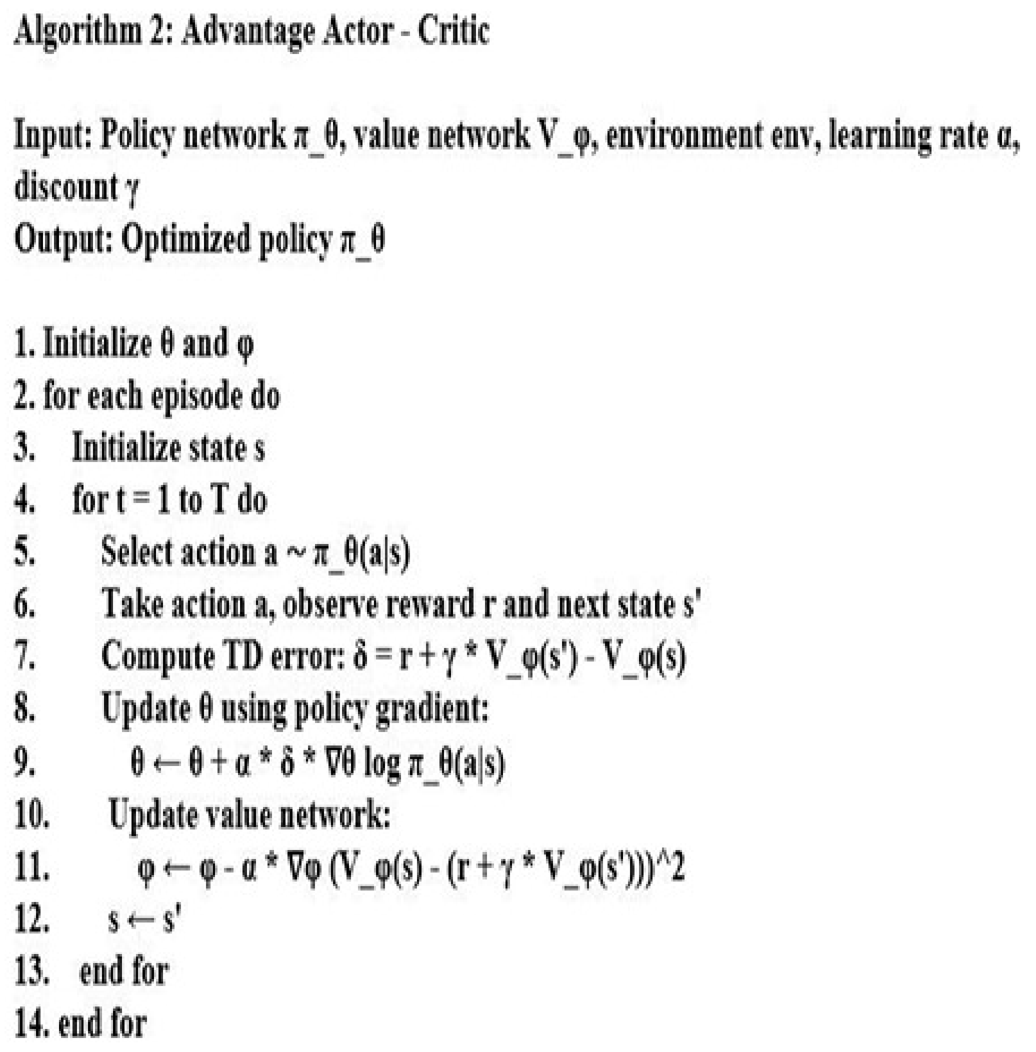
Input: Policy network π_θ, value network V_θ, environment env, number of epochs K, batch size B
Output: Optimized policy π_θ

```
1. Initialize θ randomly
2. for each iteration do
3.   Collect a set of trajectories D using π_θ in env
4.   Compute advantages Â using generalized advantage estimation (GAE)
5.   for epoch = 1 to K do
6.     for each minibatch in D do
7.       Compute ratio r(θ) = π_θ(a|s) / π_θ_old(a|s)
8.       Compute surrogate loss:
9.         L(θ) = E[min(r(θ) * Â, clip(r(θ), 1 - ε, 1 + ε) * Â)]
10.      Perform gradient ascent on L(θ)
11.      Update value network by minimizing:
12.        L_v = (V_θ(s) - R)^2
13.    end for
14.  end for
15. end for
```

Proximal Policy Optimization (PPO) for Lunar Lander

Advantage Actor-Critic (A2C)

A2C is a synchronous variant of Actor-Critic methods in which the critic assesses the value function while the actor learns the policy. It stabilizes training and lowers update variance by using the advantage function, which is the difference between expected and actual rewards.



Algorithm 2: Advantage Actor - Critic

```
Input: Policy network $\pi\_\theta$, value network $V\_\varphi$, environment env, learning rate $\alpha$,
discount $\gamma$
Output: Optimized policy $\pi\_\theta$

1. Initialize $\theta$ and $\varphi$
2. for each episode do
3.    Initialize state s
4.    for t = 1 to T do
5.       Select action $a \sim \pi\_\theta(a|s)$
6.       Take action a, observe reward r and next state s'
7.       Compute TD error: $\delta = r + \gamma * V\_\varphi(s') - V\_\varphi(s)$
8.       Update $\theta$ using policy gradient:
9.          $\theta \leftarrow \theta + \alpha * \delta * \nabla\theta \log \pi\_\theta(a|s)$
10.      Update value network:
11.         $\varphi \leftarrow \varphi - \alpha * \nabla\varphi (V\_\varphi(s) - (r + \gamma * V\_\varphi(s')))$^2
12.      s ← s'
13.   end for
14. end for
```

Advantage Actor Critic for Lunar Lander

Monte Carlo

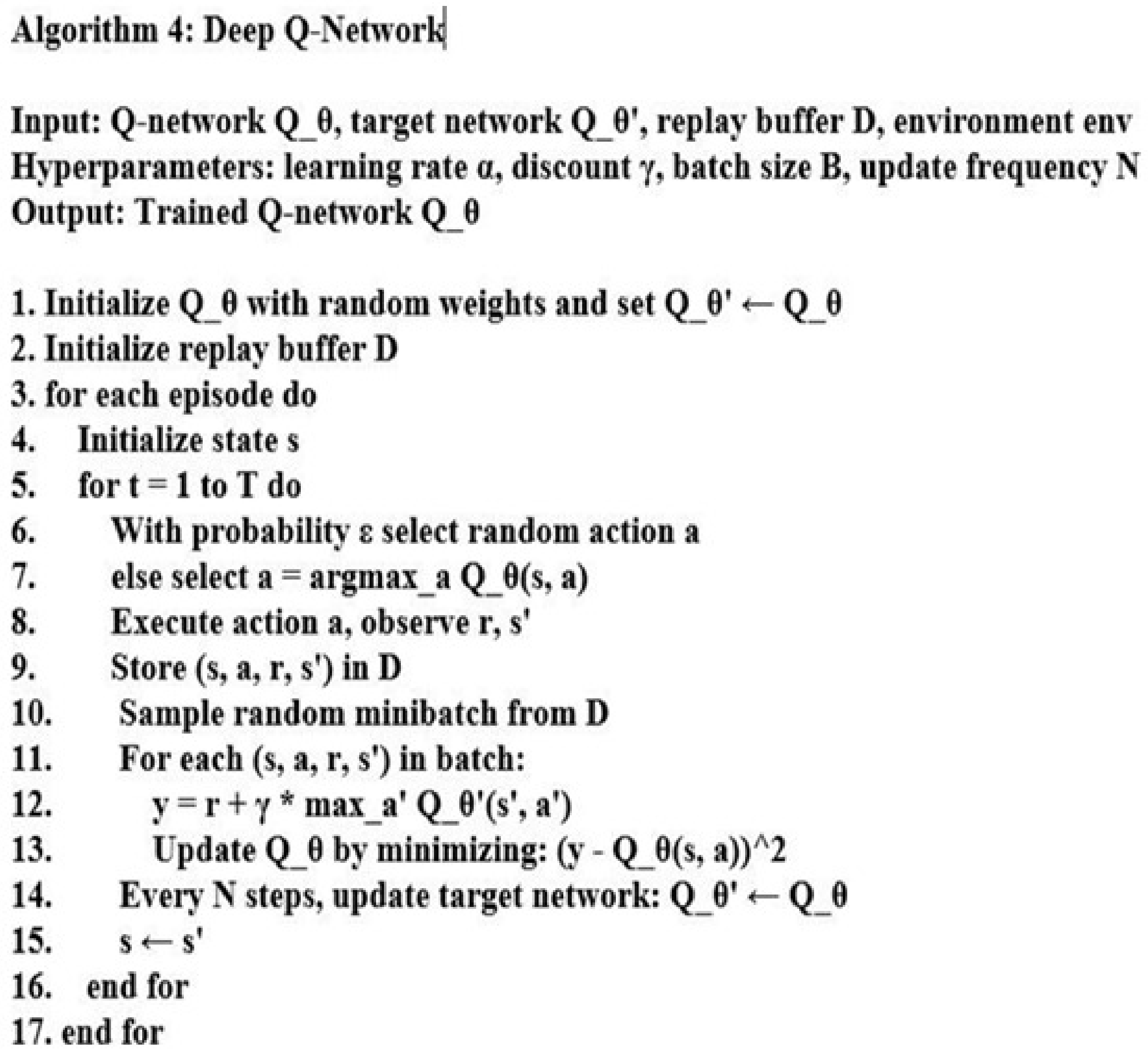
Without bootstrapping, Monte Carlo algorithms learn from entire episodes. The average return from each state they visit is used to estimate value functions, and these estimates are used to update rules. It functions well in settings with well-defined episodes.

```
Algorithm 3: Monte Carlo

Input: Policy $\pi$, state-value table V, environment env, learning rate $\alpha$, discount $\gamma$
Output: Improved policy $\pi$

1. Initialize $\pi$ arbitrarily and $V(s) = 0$ for all s
2. for each episode do
3.    Generate episode: S_0, A_0, R_1, ..., S_T using $\pi$
4.    $G \leftarrow 0$
5.    for t = T-1 to 0 do
6.       $G \leftarrow \gamma * G + R\_\{t+1\}$
7.       if (S_t, A_t) not in (S_0, A_0), ..., (S_{t-1}, A_{t-1}) then
8.          $V(S\_t) \leftarrow V(S\_t) + \alpha * (G - V(S\_t))$
9.          Update policy $\pi$ to be greedy w.r.t. V
10.   end for
11. end for
```

Monte Carlo for Lunar Lander

Deep Q – Network

DQN is a value-based reinforcement learning technique that approximates the Q-value function using a deep neural network. To increase stability and learning effectiveness, it uses experience replay and a different target network.



Algorithm 4: Deep Q-Network

Input: Q-network Q_θ, target network Q_θ', replay buffer D, environment env
Hyperparameters: learning rate α, discount γ, batch size B, update frequency N
Output: Trained Q-network Q_θ

```
1. Initialize Q_θ with random weights and set Q_θ' ← Q_θ
2. Initialize replay buffer D
3. for each episode do
4.    Initialize state s
5.    for t = 1 to T do
6.       With probability ε select random action a
7.       else select a = argmax_a Q_θ(s, a)
8.       Execute action a, observe r, s'
9.       Store (s, a, r, s') in D
10.      Sample random minibatch from D
11.      For each (s, a, r, s') in batch:
12.         y = r + γ * max_a' Q_θ'(s', a')
13.         Update Q_θ by minimizing: (y - Q_θ(s, a))^2
14.      Every N steps, update target network: Q_θ' ← Q_θ
15.      s ← s'
16.   end for
17. end for
```

Deep Q – Network for Lunar Lander