1. REINFORCE

Summary: REINFORCE introduces a fundamental innovation by employing Monte Carlo methods to estimate the return and update the policy based on actual returns from episodes. This approach improves policy gradient methods by providing complete episode returns for learning. The main problem solved is the need for accurate return estimation in environments with long episodes and sparse rewards. In REINFORCE, states and actions are used to compute returns, and policies are updated based on these returns. The return is computed as $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$, and the policy update is $\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \log \pi(a_t|s_t;\theta) \cdot G_t$, where G_t is the return from time t. **Objective:**

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R_t \right]$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Networks Used:

- **Policy Network** (Online)

Hyperparameters:

- Learning Rate (α): $1e^{-3}$ to $1e^{-4}$
- Discount Factor (γ): 0.99

2. DQN, DDQN, Prioritized Replay DDQN

DQN Summary: Deep Q-Network (DQN) is notable for combining Q-learning with deep neural networks to approximate the Q-value function, alongside using experience replay and a target network to stabilize training. This innovation addresses the problem of handling high-dimensional state spaces, such as those found in Atari games. DQN is appropriate for discrete action spaces with large state spaces. It manages states and actions using a neural network to approximate Q-values and utilizes experience replay for training stability. The Q-value update is defined as $Q(s_t, a_t) \leftarrow r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-)$, and the loss function is $L(\theta) = \mathbb{E}\left[\left(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta)\right)^2\right]$. **DDQN Summary:** Double DQN introduces the key innovation of reducing Q-value overestimation bias by using separate networks for action selection and value evaluation. This advancement improves the stability and accuracy of Q-value estimation. DDQN addresses the problem of overestimation bias present in DQN. It is suitable for discrete action spaces with high variance in Q-value estimates. In DDQN, states and actions are managed by two networks: one for selecting actions and one for evaluating them. The update rule is $Q(s_t, a_t; \theta) \leftarrow r_t + \gamma Q(s_{t+1}, \text{argmax}_{a'} Q(s_{t+1}, a'; \theta); \theta^-)$, and the loss function remains similar to DQN but uses the action selection network for computing the target. **Prioritized DDQN Summary:** Prioritized Experience Replay DDQN enhances DDQN by prioritizing the replay of important transitions, making learning more efficient. This innovation focuses on learning from more significant experiences to improve training efficiency. It solves the problem of inefficient learning from less informative experiences. It is suitable for complex environments where learning from significant experiences accelerates performance. States, actions, and rewards are managed through prioritized sampling of experiences, improving learning effectiveness. The priority update is based on the TD-error, and the loss function is similar to DDQN but incorporates sampling probabilities for transitions. **DQN Loss:**

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

DDQN Loss:

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

Prioritized Replay DDQN Loss:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')} \left[\frac{p_i}{\text{Priority\_Sum}} \left(r + \gamma Q_{\theta^-}(s', \text{argmax}_{a'} Q_\theta(s', a')) - Q_\theta(s, a) \right)^2 \right]$$

Networks Used:

- **Q-Network** (Online)
- **Target Q-Network** (Offline)

Hyperparameters:

- Learning Rate (α): $1e^{-4}$ to $1e^{-3}$
- Discount Factor (γ): 0.99
- Batch Size: 32 to 64
- Target Network Update Frequency: 1000
- Replay Buffer Size: $1e^5$ to $1e^6$
- Priority Exponent (β): 0.4 to 1.0
- Priority Scaling Factor (ϵ): $1e^{-5}$

3. A2C/A3C

A2C Summary: Advantage Actor-Critic (A2C) innovates by using the advantage function to reduce variance in policy gradient updates and combines actor-critic methods for more stable learning. This approach addresses the need for stability and efficiency in policy learning. A2C is appropriate for environments requiring stable policy optimization. It handles states and actions with a value function (critic) to estimate advantages, which helps update the policy (actor). The advantage function is $\text{Adv}(s, a) = Q(s, a) - V(s)$, and the policy update is $\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \log \pi(a_t|s_t;\theta) \cdot \text{Adv}(s_t, a_t)$. **A3C Summary:** Asynchronous Actor-Critic Agents (A3C) use multiple parallel agents to explore different parts of the environment and update a global model asynchronously. This innovation speeds up learning and enhances robustness by utilizing parallelism. A3C addresses the problem of slow and unstable learning by combining experiences from multiple agents. It is appropriate for environments where parallel computation can be leveraged. States and actions are managed by multiple agents exploring the environment, and rewards are used to update a shared global model asynchronously. The objective function is similar to A2C but includes asynchronous updates. **Actor Update:**

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t, a_t} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t) \right]$$

Critic Update:

$$L(\phi) = \mathbb{E}_{s_t} \left[\left(R_t - V_\phi(s_t) \right)^2 \right]$$

Networks Used:

- **Actor Network** (Online)
- **Critic Network** (Online)
- **Multiple Workers (A3C)**

Hyperparameters:

- Actor Learning Rate (α): $1e^{-4}$
- Critic Learning Rate (β): $1e^{-3}$
- Discount Factor (γ): 0.99
- Number of Workers (A3C): 16 to 32

4. DPG, DDPG

DPG Summary: Deterministic Policy Gradient (DPG) introduces the innovation of computing the gradient of expected rewards with respect to deterministic policy parameters, specifically for continuous action spaces. This method addresses the need for effective handling of continuous action spaces where traditional methods are not suitable. DPG is appropriate for continuous action spaces with deterministic policies. It handles states and actions by optimizing a deterministic policy and computes gradients directly. The policy gradient is $\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho_\pi} [\nabla_\theta \pi(a|s;\theta) \cdot \nabla_a Q(s, a;\theta)]$. **DDPG Summary:** Deep Deterministic Policy Gradient (DDPG) extends DPG by using deep neural networks for approximating both the policy and value functions, incorporating experience replay and target networks. This approach addresses the challenge of scaling to high-dimensional continuous action spaces. DDPG is suitable for high-dimensional continuous action spaces like robotic control. It manages states, actions, and rewards using deep networks for approximation and employs experience replay for training stability. The Q-value update is $Q(s_t, a_t; \theta^Q) \leftarrow r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}; \theta^\mu); \theta^Q)$, and the policy update is $\nabla_\theta^\mu J = \mathbb{E}\left[\nabla_a Q(s, a; \theta^Q) \nabla_\theta^\mu \mu(s; \theta^\mu)\right]$. **DPG/DDPG Actor Update:**

$$\nabla_\phi J(\phi) = \mathbb{E}_s \left[\nabla_a Q_\theta(s, a)\big|_{a=\mu_\phi(s)} \nabla_\phi \mu_\phi(s) \right]$$

DDPG Critic Loss:

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

Networks Used:

- **Actor Network** (Online)
- **Critic Network** (Online)
- **Target Actor Network** (Offline)
- **Target Critic Network** (Offline)

Hyperparameters:

- Actor Learning Rate (α): $1e^{-4}$ to $1e^{-3}$
- Critic Learning Rate (β): $1e^{-3}$
- Discount Factor (γ): 0.99
- Replay Buffer Size: $1e^5$ to $1e^6$
- Batch Size: 100
- Soft Update Rate (τ): 0.005
- Target Policy Noise: 0.2

5. TRPO, PPO

TRPO Summary: Trust Region Policy Optimization (TRPO) innovates by optimizing policies within a trust region to prevent large, destabilizing policy updates. This method solves the problem of maintaining policy stability during updates. TRPO is appropriate for complex environments where policy stability is crucial. It handles states and actions through constrained optimization, ensuring that policy changes remain within a trust region. The objective is to maximize $\mathbb{E}_{s_t}[\text{Adv}(s_t, a_t)]$, subject to a constraint on the Kullback-Leibler (KL) divergence: $\mathbb{E}_{s_t}[\text{KL}(\pi_{\text{old}}(a|s)\|\pi(a|s))] \leq \delta$. **PPO Summary:** Proximal Policy Optimization (PPO) simplifies TRPO's trust region optimization by using a clipped surrogate objective function to ensure stable policy updates. This innovation provides a practical and scalable approach to policy optimization. PPO solves the problem of maintaining stability in policy updates with a more straightforward implementation. It is suitable for both discrete and continuous action spaces. PPO manages states, actions, and rewards through a clipped objective function, balancing exploration and exploitation. The objective is to maximize $\mathbb{E}[\min(r_t(\theta) \cdot \text{Adv}(s_t, a_t), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot \text{Adv}(s_t, a_t))]$, where $r_t(\theta) = \frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_{\text{old}})}$. **TRPO Objective:**

$$\max_\theta \mathbb{E}_{s,a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\theta_{\text{old}}}}(s,a)\right]$$

PPO Objective:

$$\max_\theta \mathbb{E}_{s,a\sim\pi_{\theta_{\text{old}}}}\left[\min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right]$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

Networks Used:

- **Policy Network** (Online)
- **Value Network** (Online)

Hyperparameters:

- Learning Rate (α): $1e^{-4}$ to $3e^{-4}$
- Discount Factor (γ): 0.99
- KL Divergence Constraint (δ): 0.01 to 0.02
- Clip Range (ϵ): 0.1 to 0.2
- Number of Epochs: 3 to 10
- Batch Size: 64 to 2048

6. MCTS

Summary: Monte Carlo Tree Search (MCTS) innovates by utilizing simulations to explore potential future states and build a search tree based on empirical outcomes. This method addresses decision-making in complex environments by exploring and evaluating future possibilities. MCTS is appropriate for games and decision-making problems with large state spaces. It manages states and actions through a search tree and evaluates possible outcomes using simulations. The objective involves balancing exploration and exploitation with the Upper Confidence Bound (UCB1) formula: $\text{UCB1} = \bar{X}_j + C\sqrt{\frac{\ln N_i}{n_j}}$. **UCT Formula:**

$$Q(s,a) = \frac{w(s,a)}{n(s,a)} + c\sqrt{\frac{\ln N(s)}{n(s,a)}}$$

Networks Used:

- **Search Tree** (Online)

Hyperparameters:

- Exploration Constant (c): Empirical
- Number of Simulations: 1000 to 10000
- Tree Depth: Problem-specific

7. AlphaGo, AlphaZero

AlphaGo Summary: AlphaGo combines MCTS with deep neural networks for state evaluation and move selection, enabling it to tackle the complex game of Go. This innovation leverages deep learning to guide and improve MCTS. AlphaGo solves the problem of mastering Go, a game with immense complexity and strategic depth. It is appropriate for board games with large state spaces. States and actions are evaluated using neural networks, and MCTS is used for decision-making. The policy network outputs move probabilities, and the value network estimates winning chances from a given board state. **AlphaZero Summary:** AlphaZero generalizes the AlphaGo approach by combining MCTS with deep learning trained through self-play, making it applicable to multiple board games. This innovation creates a versatile system capable of mastering various games without domain-specific knowledge. AlphaZero addresses the need for a universal learning system that can generalize across different games. It is suitable for board games and other decision-making problems. AlphaZero uses self-play to generate training data and combines it with neural networks and MCTS for decision-making. The objective involves optimizing both policy and value networks through self-play and MCTS. **Loss Function:**

$$L(\theta) = \mathbb{E}_{(s,\pi,z)\sim D}\left[(z - V(s;\theta))^2 - \pi \cdot \log \pi(a|s;\theta) + \lambda\|\theta\|^2\right]$$

Networks Used:

- **Policy Network** (Online)
- **Value Network** (Online)

Hyperparameters:

- Learning Rate (α): $1e^{-3}$ to $1e^{-2}$
- Discount Factor (γ): 1.0
- MCTS Simulations: Several thousand
- Temperature: Annealed over time
- Regularization Term (λ): $1e^{-4}$ to $1e^{-3}$