1. REINFORCE

Summary: REINFORCE introduced the idea of directly optimizing a policy using the policy gradient theorem. This method represented a significant shift from value-based methods by focusing on optimizing the policy itself.
Big Idea: The major change was moving away from value function-centric methods (like Q-learning) to directly optimizing the policy via gradient ascent. This allowed for more complex and flexible policies.
Motivation: REINFORCE addressed the difficulty of learning optimal policies directly, especially in environments with large or continuous state-action spaces.
Appropriateness: Suitable for problems with large state spaces or continuous action spaces where a direct parameterization of the policy is feasible.
Handling States, Time, Actions, Rewards: It parameterizes the policy and updates it based on rewards received. The return R_t from the start of an episode is used to evaluate the policy's performance.
Mathematical Formulation:

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

The gradient $\nabla_\theta J(\theta)$ reflects how policy parameters θ should be adjusted to maximize the expected return.

2. DQN, DDQN, Prioritized Replay DDQN

Summary: DQN (Deep Q-Network) uses a neural network to approximate Q-values, addressing the instability in Q-learning with function approximation. DDQN (Double DQN) mitigates Q-value overestimation by decoupling action selection and evaluation. Prioritized Replay DDQN improves efficiency by prioritizing important experiences.
Big Idea: The key innovation was the use of neural networks to approximate Q-values and the introduction of experience replay and target networks to stabilize training.
Motivation: These methods were developed to handle high-dimensional state spaces and stabilize learning, which earlier algorithms struggled with.
Appropriateness: Effective for problems with discrete action spaces and large state spaces, such as video games or complex control tasks.
Handling States, Time, Actions, Rewards: DQN uses a Q-network to approximate action-value functions. DDQN improves this by using a separate target network for stability. Prioritized Replay enhances efficiency by sampling experiences based on their importance.
Mathematical Formulation:

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

The loss function $L(\theta)$ measures the difference between the predicted Q-value and the target Q-value, driving the update of the Q-network parameters θ.

3. A2C/A3C

Summary: A2C (Advantage Actor-Critic) uses both an actor to update the policy and a critic to evaluate it. A3C (Asynchronous Actor-Critic Agents) extends A2C by running multiple agents in parallel to improve stability and learning speed.
Big Idea: The innovation lies in the use of both actor and critic networks to separately handle policy improvement and value estimation, and the parallelization in A3C to stabilize learning.
Motivation: These methods were designed to overcome the high variance in policy gradient methods and to speed up training through parallelism.
Appropriateness: Suitable for environments where multiple agents can interact concurrently, and where the learning process can benefit from both policy and value function optimization.
Handling States, Time, Actions, Rewards: A2C uses the advantage function $A(s_t, a_t)$ to guide policy updates and evaluates state values through the critic. A3C utilizes multiple agents to collect diverse experiences.
Mathematical Formulation:

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

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

The actor update uses the advantage function to guide policy improvement, while the critic update minimizes the error between predicted and actual returns.

4. DPG, DDPG

Summary: DPG (Deterministic Policy Gradient) and DDPG (Deep Deterministic Policy Gradient) extend Q-learning to continuous action spaces. DDPG enhances DPG with experience replay and target networks for improved stability.
Big Idea: The key innovation is the use of deterministic policy gradients for continuous action spaces and the introduction of experience replay and target networks in DDPG.
Motivation: These methods were developed to handle continuous action spaces and to stabilize training through improved learning techniques.
Appropriateness: Suitable for problems with continuous action spaces, such as robotic control or autonomous driving.
Handling States, Time, Actions, Rewards: DDPG uses actor-critic architecture with continuous action spaces, leveraging target networks and experience replay to stabilize learning.
Mathematical Formulation:

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

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

The actor update uses deterministic policy gradients, while the critic loss function updates the Q-network based on target values from the target networks.

5. TRPO, PPO

Summary: TRPO (Trust Region Policy Optimization) ensures policy updates stay within a trust region to improve stability, while PPO (Proximal Policy Optimization) simplifies TRPO with a clipped objective to constrain updates.
Big Idea: TRPO's innovation was to use a trust region constraint for policy updates, while PPO simplified this with a clipping mechanism, making it more practical and easier to implement.
Motivation: Both methods were developed to address issues of instability and inefficiency in policy optimization, particularly in high-dimensional spaces.
Appropriateness: Effective for high-dimensional and continuous action spaces where stability and efficiency in policy optimization are crucial.
Handling States, Time, Actions, Rewards: Both methods use the policy gradient approach to optimize policies. PPO simplifies TRPO by using a clipped objective function to limit policy changes.
Mathematical Formulation:

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

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

TRPO uses a trust region constraint to prevent large policy updates, while PPO uses a clipped objective function to achieve similar stability.

6. MCTS

Summary: Monte Carlo Tree Search (MCTS) is a search algorithm for making decisions in large spaces using simulation to evaluate potential actions.
Big Idea: The fundamental innovation is the use of simulation to explore and evaluate decisions, enabling effective decision-making in complex and large state spaces.
Motivation: MCTS was developed to make decisions in domains with large state spaces where traditional methods would be computationally prohibitive.
Appropriateness: Suitable for decision-making problems with large state spaces and where simulations can be used to evaluate potential actions, such as in game-playing and planning problems.
Handling States, Time, Actions, Rewards: MCTS builds a search tree where nodes represent states and edges represent actions. It uses simulations to estimate the value of different actions.
Mathematical Formulation:

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

The formula balances exploitation (average reward) and exploration (uncertainty) using the UCT (Upper Confidence Bound for Trees) algorithm.

7. AlphaGo, AlphaZero

Summary: AlphaGo and AlphaZero use deep learning combined with MCTS for game-playing. AlphaGo was designed for Go, while AlphaZero generalizes this approach for various games using self-play.
Big Idea: The innovation is combining deep learning with MCTS to create powerful game-playing agents. AlphaZero extended this to general game-playing with self-play.
Motivation: These methods address the need for high-performance game-playing agents that can learn and improve from self-play, avoiding reliance on human expertise.
Appropriateness: Ideal for strategic games and other complex environments where self-play and simulation are feasible for training.
Handling States, Time, Actions, Rewards: AlphaGo and AlphaZero use deep networks to estimate policy and value functions, combined with MCTS for decision-making.
Mathematical Formulation:

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

The loss function balances prediction of state values, policy probabilities, and regularization, guiding the training of the policy and value networks.