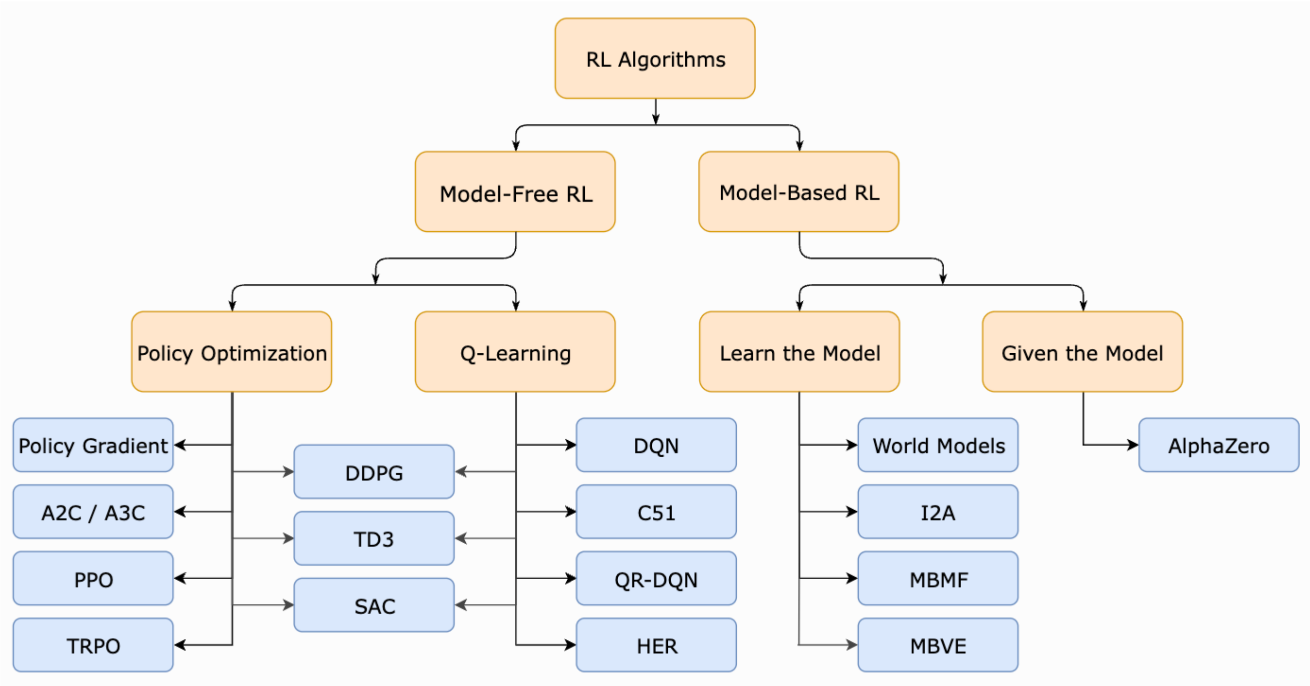
Policy Gradient methods for RL



Policy Gradient methods are a class of **Reinforcement Learning (RL)** algorithms that directly optimize the **policy** (the agent's behavior) by gradient ascent on the expected reward. Unlike value-based methods (e.g., Q-learning), which learn a value function and derive a policy from it, policy gradient methods **parametrize the policy** and adjust the parameters to maximize performance.

Key Idea

The goal is to maximize the **expected return** (cumulative reward) by updating the policy parameters **θ** in the direction of the gradient of the performance measure $J(\theta)$:

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

where:

- α = learning rate
- $\nabla_\theta J(\theta)$ = gradient of the objective function w.r.t. policy parameters

Objective Function

The performance measure $J(\theta)$ is typically the **expected return** under the policy:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

where:

- $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$ is a trajectory (sequence of states, actions, and rewards).
- $R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$ is the discounted return of the trajectory.

Policy Gradient Theorem

The **Policy Gradient Theorem** provides a way to compute the gradient analytically:

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

This means:

- We sample trajectories using the current policy π_θ.
- For each action taken, compute the gradient of the log-probability of that action.
- Weight this gradient by the total reward $R(\tau)$.

Variants of Policy Gradient Methods

1. **REINFORCE (Monte Carlo Policy Gradient)**
   - Uses **full trajectory returns** $R(\tau)$ as the reward signal.
   - High variance but unbiased.
2. **Actor-Critic Methods**
   - Combines policy gradient (actor) with a value function (critic).
   - The critic reduces variance by using a baseline (e.g., state-value $V(s)$) instead of raw returns.
   - Example: **Advantage Actor-Critic (A2C/A3C)**, where the advantage $A(s, a) = Q(s, a) - V(s)$ is used.
3. **Proximal Policy Optimization (PPO)**
   - Improves stability by limiting policy updates to avoid drastic changes.
   - Uses a clipped objective function to ensure small updates.
4. **Trust Region Policy Optimization (TRPO)**
   - Uses constrained optimization to ensure updates stay within a "trust region."
   - More stable but computationally expensive.

Advantages of Policy Gradient Methods

- Can learn **stochastic policies** (useful in partially observable environments).
- Naturally handle **continuous action spaces**.
- Directly optimize the policy, avoiding issues like policy degradation in value-based methods.

Disadvantages

- High variance in gradient estimates (can be mitigated with baselines or actor-critic methods).
- Sample-inefficient compared to some value-based methods (e.g., DQN).

Example: REINFORCE Algorithm

1. Initialize policy parameters θ.
2. Generate a trajectory τ using π_θ.
3. Compute $R(\tau)$.
4. Update $\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot R(\tau)$.
5. Repeat until convergence.

Conclusion

Policy Gradient methods are powerful for RL tasks, especially when dealing with **complex action spaces** or **stochastic environments**. Modern improvements like PPO and A2C make them more stable and scalable.

Proximal Policy Optimization (PPO) - A Deep Dive

PPO is a state-of-the-art **policy gradient** method designed to improve training stability and sample efficiency. It addresses key issues in traditional policy optimization, such as **large policy updates** that can destabilize learning.

---

1. Core Idea of PPO

PPO belongs to the **Actor-Critic** family, where:

- The **Actor** (policy) selects actions.
- The **Critic** (value function) evaluates actions and provides feedback.

The key innovation in PPO is its **constrained optimization approach**, ensuring that policy updates are **not too large**, preventing catastrophic performance drops.

---

2. Key Components of PPO

(1) Policy Objective Function

The objective is to maximize:

$$J(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ (**probability ratio** of new vs. old policy).
- $\hat{A}_t$ = **advantage estimate** (how much better an action is compared to average).
- ϵ = clipping hyperparameter (typically **0.1 to 0.3**).

(2) Clipped Surrogate Objective

- The **clipping** prevents drastic updates by limiting how much $r_t(\theta)$ can deviate from 1.
- If the advantage $\hat{A}_t$ is positive (good action), the update is capped at $1+\epsilon$.
- If the advantage $\hat{A}_t$ is negative (bad action), the update is floored at $1-\epsilon$.

This ensures **small, stable policy updates**.

(3) Advantage Estimation

PPO typically uses **Generalized Advantage Estimation (GAE)** to compute $\hat{A}_t$:

$$\hat{A}_t = \sum_{k=0}^{\infty} (\gamma\lambda)^k \delta_{t+k}$$

where:

- $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ (TD error).
- λ = smoothing parameter (usually **0.9–0.99**).
- γ = discount factor.

This reduces variance while maintaining bias control.

---

3. PPO Algorithm Steps

1. **Collect trajectories** using the current policy $\pi_{\theta_{\text{old}}}$.
2. **Compute advantages** $\hat{A}_t$ using GAE.
3. **Optimize the clipped objective** via stochastic gradient ascent (usually over multiple epochs).
4. **Update the policy** $\theta \leftarrow \theta_{\text{new}}$.

5. **Repeat** until convergence.

---

4. Why PPO? Key Benefits

✅ **Stable Training** (clipping prevents destructive updates).
✅ **Sample Efficient** (reuses data via multiple epochs).
✅ **Works Well in Continuous & Discrete Action Spaces**.
✅ **Hyperparameter Robustness** (less sensitive than TRPO).

---

5. PPO Variants

- **PPO-Clip** (Standard version, uses clipping).
- **PPO-Penalty** (Uses KL divergence penalty instead of clipping).
- **PPO with Adaptive KL** (Dynamically adjusts KL penalty).

---

6. Practical Implementation Tips

- **Batch Size**: Larger batches reduce variance (e.g., 64–2048).
- **Learning Rate**: Typically **3e-4** (Adam optimizer works well).
- **GAE Lambda**: **0.9–0.99** balances bias-variance tradeoff.
- **Clipping Range** ϵ: **0.1–0.3** (0.2 is common).
- **Number of Epochs per Update**: **3–10** (avoids overfitting).

---

7. PPO vs. Other Policy Gradients

Method	Key Feature	Pros	Cons
PPO	Clipped updates	Stable, efficient	Slightly complex
TRPO	Constrained KL divergence	Theoretically sound	Computationally heavy
A2C/A3C	Advantage-based	Simple, parallelizable	High variance
REINFORCE	Pure Monte Carlo	Simple	High variance, inefficient

---

8. Applications of PPO

- **Robotics** (continuous control).
- **Game AI** (Dota 2, StarCraft II).
- **Autonomous Driving**.
- **Finance** (portfolio optimization).

---

9. Code Example (Pseudocode)

```
for iteration in range(num_iterations):
    # Collect trajectories using current policy
    trajectories = collect_data(pi_old)

    # Compute advantages using GAE
    advantages = compute_gae(trajectories)

    # Optimize policy for K epochs
    for epoch in range(K):
        batches = split_into_minibatches(trajectories)
        for batch in batches:
            # Compute clipped surrogate loss
            loss = -min(
                ratio * advantages,
                clip(ratio, 1-eps, 1+eps) * advantages
            )

            # Update policy via gradient ascent
            optimizer.step(loss)

    # Update old policy
    pi_old = pi_new
```

---

10. Conclusion

PPO is **one of the most popular RL algorithms** due to its balance between **stability, efficiency, and performance**. Its **clipped objective** makes it robust, while **GAE** reduces variance. If you're implementing policy gradients, PPO is often the best choice.

In-Depth Explanation of PPO's Policy Objective Function

The **Proximal Policy Optimization (PPO)** objective function is designed to **maximize policy performance while preventing excessively large updates** that could destabilize training. The key formula is:

$$J(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right]$$

Let's break down **each component** and explain its **mathematical intuition**.

---

1. Probability Ratio $r_t(\theta)$

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

What it Represents:

- Measures **how much the new policy π_θ deviates** from the old policy $\pi_{\theta_{\text{old}}}$.
- If $r_t(\theta) > 1$, the new policy is **more likely** to take action a_t in state s_t.
- If $r_t(\theta) < 1$, the new policy is **less likely** to take that action.

Intuition:

- Encourages **increasing the probability of good actions** (where advantage $\hat{A}_t > 0$).
- Discourages **bad actions** (where $\hat{A}_t < 0$) by reducing their probability.

---

2. Advantage Estimate $\hat{A}_t$

$$\hat{A}_t = R_t - V(s_t)$$

(where R_t is the **discounted return**, and $V(s_t)$ is the **value function**)

What it Represents:

- Measures **how much better an action is** compared to the average action in that state.
- If $\hat{A}_t > 0$, the action was **better than expected**.
- If $\hat{A}_t < 0$, the action was **worse than expected**.

Intuition:

- Acts as a **weight** for policy updates:
  - High $\hat{A}_t$ → Stronger push to increase $\pi_\theta(a_t|s_t)$.
  - Low $\hat{A}_t$ → Decrease $\pi_\theta(a_t|s_t)$.

---

3. Clipping Function $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$

$$\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) = \begin{cases} 1+\epsilon & \text{if } r_t(\theta) > 1+\epsilon \\ 1-\epsilon & \text{if } r_t(\theta) < 1-\epsilon \\ r_t(\theta) & \text{otherwise} \end{cases}$$

What it Represents:

- **Limits how much the policy can change** in a single update.
- ϵ (e.g., 0.2) controls the **clipping range**.

Intuition:

- Prevents **destructive updates** where $r_t(\theta)$ becomes too large or too small.
- Ensures **smooth learning** by keeping updates within a **trust region**.

---

4. The Min Operator $\min(\cdot)$

$$\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)$$

What it Represents:

- Takes the **minimum** between the **unclipped** and **clipped** objectives.

Intuition:

- **Clipping is only active when it helps** (i.e., when the unclipped update would be too large).
- Forms a **pessimistic bound** (lower bound) on policy improvement:
  - If $\hat{A}_t > 0$, the objective is **capped** at $(1 + \epsilon)\hat{A}_t$.
  - If $\hat{A}_t < 0$, the objective is **floored** at $(1 - \epsilon)\hat{A}_t$.

---

5. Full Objective Interpretation

The complete objective:

$$J(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t \right) \right]$$

Key Behaviors:

1. **When $\hat{A}_t > 0$ (Good Action):**
   - We want to **increase** $\pi_\theta(a_t|s_t)$, but **not too much**.
   - The update is **capped at** $(1 + \epsilon)\hat{A}_t$.
2. **When $\hat{A}_t < 0$ (Bad Action):**
   - We want to **decrease** $\pi_\theta(a_t|s_t)$, but **not too much**.
   - The update is **floored at** $(1 - \epsilon)\hat{A}_t$.
3. **When $r_t(\theta)$ is within $[1 - \epsilon, 1 + \epsilon]$:**
   - The update follows the **standard policy gradient** $r_t(\theta)\hat{A}_t$.

---

6. Why This Works: The Intuition Behind PPO

- **Prevents Overly Large Updates**:
  - Without clipping, a large $r_t(\theta)$ could lead to **catastrophic policy collapse** (e.g., the policy becomes deterministic too quickly).
  - Clipping ensures **conservative updates**, improving stability.
- **Balances Exploration & Exploitation**:
  - The probability ratio $r_t(\theta)$ allows **controlled exploration** while avoiding extreme deviations.
- **Lower Variance than REINFORCE**:
  - Using $\hat{A}_t$ (instead of Monte Carlo returns) reduces variance.

---

7. Comparison to Unclipped Policy Gradient

Update Rule	Behavior	Problem
Standard PG: $r_t(\theta)\hat{A}_t$	Can make **unbounded updates**	May destabilize training
PPO (Clipped): $\min(\cdot)$	**Limits update size**	More stable, avoids collapse

---

8. Practical Implications

- **Clipping Range ϵ:**
  - Too small ($\epsilon = 0.1$) → Slow learning.
  - Too large ($\epsilon = 0.3$) → Risk of instability.
  - **Default:** $\epsilon = 0.2$.
- **Advantage Normalization:**
  - Often, $\hat{A}_t$ is normalized (mean=0, std=1) to improve stability.
- **Multiple Epochs per Batch:**
  - PPO reuses data for **3-10 gradient steps**, improving sample efficiency.

---

9. Summary of Key Intuitions

1. $r_t(\theta)\hat{A}_t$**:** The core policy gradient term.
2. **Clipping:** Preovershooting updates.
3. **Min Operator:** Ensures updates are **conservative**.
4. **Advantage $\hat{A}_t$:** Guides whether to increase or decrease action probability.

Final Thoughts

PPO's objective is **a smart balance between performance and stability**, making it one of the most reliable RL algorithms. The **clipping mechanism** is the key innovation, preventing the policy from changing too rapidly while still allowing efficient learning.

How Clipping Works in PPO's Surrogate Objective

The **clipping mechanism** in Proximal Policy Optimization (PPO) is the key innovation that prevents excessively large policy updates, ensuring stable training. Let's break it down step-by-step.

---

1. Recap: PPO's Surrogate Objective

The core objective is:

$$J(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right]$$

where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ (**probability ratio** of new vs. old policy).
- $\hat{A}_t$ = **advantage** (how much better a_t is than average).
- ϵ = clipping hyperparameter (typically **0.1–0.3**).

---

2. The Clipping Function

The `clip` operation restricts $r_t(\theta)$ to the range $[1 - \epsilon, 1 + \epsilon]$:

$$\text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) = \begin{cases} 1 + \epsilon & \text{if } r_t(\theta) > 1 + \epsilon, \\ 1 - \epsilon & \text{if } r_t(\theta) < 1 - \epsilon, \\ r_t(\theta) & \text{otherwise.} \end{cases}$$

Visualization of Clipping

```
Probability Ratio (r_t)
|
1+ε |------------
    |           /
1   |          /
1-ε |_________/
    |
    -------------------->
      Policy Update Direction
```

- **Inside the bounds** ($1 - \epsilon \leq r_t \leq 1 + \epsilon$): No clipping; normal update.
- **Outside the bounds**: The gradient is "clipped" to avoid drastic changes.

---

3. The Min Operator: Why It Matters

The `min` operator ensures the update is **pessimistic** (i.e., it takes the **worst-case** improvement to avoid instability):

$$\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta))\hat{A}_t\right)$$

Two Cases:

(1) When $\hat{A}_t > 0$ (Good Action)

- We want to **increase** $\pi_\theta(a_t|s_t)$, but not too much.
- The update is **capped** at $(1+\epsilon)\hat{A}_t$.

(2) When $\hat{A}_t < 0$ (Bad Action)

- We want to **decrease** $\pi_\theta(a_t|s_t)$, but not too much.
- The update is **floored** at $(1-\epsilon)\hat{A}_t$.

---

4. Numerical Example

Suppose:

- $\epsilon = 0.2$,
- $\hat{A}_t = +0.5$ (good action),
- Old policy probability $\pi_{\theta_{\text{old}}}(a_t|s_t) = 0.4$.

Case 1: New Policy Increases Probability ($\pi_\theta = 0.6$)

$$r_t(\theta) = \frac{0.6}{0.4} = 1.5$$

- **Unclipped update**: $1.5 \times 0.5 = 0.75$.
- **Clipped update**: $\text{clip}(1.5, 0.8, 1.2) \times 0.5 = 1.2 \times 0.5 = 0.6$.
- **Final update**: $\min(0.75, 0.6) = 0.6$.

→ The update is **capped at 0.6** instead of 0.75.

Case 2: New Policy Decreases Probability ($\pi_\theta = 0.2$)

$$r_t(\theta) = \frac{0.2}{0.4} = 0.5$$

- **Unclipped update**: $0.5 \times 0.5 = 0.25$.
- **Clipped update**: $\text{clip}(0.5, 0.8, 1.2) \times 0.5 = 0.8 \times 0.5 = 0.4$.
- **Final update**: $\min(0.25, 0.4) = 0.25$.

→ The update is **not clipped** because $r_t(\theta)\hat{A}_t$ is already conservative.

5. Why Clipping Works

- **Prevents Catastrophic Updates**: Without clipping, a large $r_t(\theta)$ could lead to **policy collapse** (e.g., the policy becomes deterministic too quickly).
- **Smoother Learning**: Ensures updates are **conservative** and incremental.
- **Robust to Hyperparameters**: Less sensitive to learning rate choices than vanilla policy gradients.

---

6. Connection to Trust Region Optimization

PPO is a **first-order approximation** of Trust Region Policy Optimization (TRPO), which enforces a hard constraint on policy updates using KL divergence.

- **TRPO**: Uses complex second-order methods.
- **PPO**: Simpler (just clipping), but empirically works as well.

---

7. Practical Tips for Clipping

- **Typical** ϵ: 0.1–0.3 (0.2 is a good default).
- **Normalize Advantages**: Reduces variance in updates.
- **Multiple Epochs**: Reuse data for 3–10 epochs per batch.

---

8. Summary

Scenario	Clipping Effect
$\hat{A}_t > 0$	Caps $r_t(\theta)$ at $1 + \epsilon$; prevents over-optimism.
$\hat{A}_t < 0$	Floors $r_t(\theta)$ at $1 - \epsilon$; prevents over-pessimism.
$r_t(\theta)$ in bounds	No clipping; standard policy gradient update.
