1.1 The REINFORCE Algorithm

REINFORCE is a foundational policy gradient method in reinforcement learning that directly optimizes the expected return of a policy through gradient ascent. The algorithm operates as follows:

- **Trajectory Sampling:** The agent interacts with the environment to generate trajectories consisting of states, actions, and rewards.
- **Return Calculation:** The discounted cumulative rewards for each trajectory are computed as:

$$G_t = \sum_{k=t+1} \gamma^{k-t} r_k, \tag{1}$$

  where γ is the discount factor.

- **Policy Gradient Estimation:** The gradient of the expected return with respect to the policy parameters is estimated using:

$$\nabla_\vartheta J(\vartheta) = \mathbb{E}_\pi \left[G_t \nabla_\vartheta \log \pi_\vartheta(A_t|S_t) \right]. \tag{2}$$

- **Policy Update:** The policy parameters are updated via gradient ascent:

$$\vartheta \leftarrow \vartheta + \alpha \nabla_\vartheta J(\vartheta), \tag{3}$$

  where α is the learning rate.

Despite its simplicity, REINFORCE suffers from high variance in gradient estimates, which can hinder its scalability to complex tasks such as aligning LLMs.

1.2 Challenges in RLHF

RLHF implementations often encounter the following challenges:

- **Computational Overhead:** Methods like PPO require a critic network, increasing memory and computational demands.
- **Training Instability:** The interdependence between the policy and value networks in PPO can lead to convergence issues, particularly for large and complex models [3].
- **Scalability:** Many advanced methods introduce additional hyperparameters and architectural components, complicating their deployment at scale.

REINFORCE++, by design, addresses these challenges through its simplicity and efficiency, making it a compelling alternative for RLHF tasks.

REINFORCE++ Enhancements

REINFORCE++ incorporates several key optimizations to enhance training stability and efficiency:

1.3 Token-Level KL Penalty

We implement a token-level Kullback-Leibler (KL) divergence penalty between the RL model and the supervised fine-tuning (SFT) model distributions. This penalty is incorporated into the reward function as follows:

$$r(s_t, a_t) = \mathbf{I}(s_t = [\mathrm{EOS}])r(x, y) - \beta\,\mathrm{KL}(t) \tag{4}$$

$$\mathrm{KL}(t) = \log\left(\frac{\pi^{\mathrm{RL}}_{\theta_{\mathrm{old}}}(a_t|s_t)}{\pi^{\mathrm{SFT}}(a_t|s_t)}\right) \tag{5}$$

where:

- x represents the input prompt
- y denotes the generated response
- $\mathbf{I}(s_t = [\mathrm{EOS}])$ indicates whether t is the final token
- β is the KL penalty coefficient

This approach facilitates better credit assignment and seamless integration with process reward models (PRM).

1.4 PPO-Clip Integration

We incorporate PPO's clipping mechanism to constrain policy updates:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{6}$$

Where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)}$ is the probability ratio of taking action a_t in state s_t under the new policy versus the old policy.
- $\hat{A}_t$ is the estimated advantage for token t.
- $\mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ restricts the probability ratio to be within the range of $[1-\epsilon, 1+\epsilon]$, where ϵ is a small hyperparameter (commonly set to around 0.2).

This formulation effectively allows the algorithm to take advantage of positive advantages while preventing excessively large updates that could destabilize training. The use of the minimum function ensures that if the ratio moves too far from 1 (either above or below), it does not contribute positively to the objective, thus maintaining a form of trust region for policy updates.

1.5 Mini-Batch Updates

To enhance training efficiency, we implement mini-batch updates with the following characteristics:

- **Batch Processing:** Data is processed in smaller, manageable chunks rather than full-batch updates.
- **Multiple Updates:** Each mini-batch allows for multiple parameter updates, improving convergence rates.
- **Stochastic Optimization:** Introduces beneficial randomness for better generalization.

1.6 Reward Normalization and Clipping

We implement comprehensive reward processing to stabilize training:

- **Normalization:** Standardizes rewards using z-score normalization to mitigate outliers.
- **Clipping:** Constrains reward values within predefined bounds to avoid instability.
- **Scaling:** Applies appropriate scaling factors for numerical stability during updates.

1.7 Advantage Normalization

The advantage function in REINFORCE++ is defined as:

$$A_t(s_t, a_t) = r(x, y) - \beta \cdot \sum_{i=t}^{T} \mathrm{KL}(i) \tag{7}$$

We normalize these advantages using z-score normalization:

$$A_{\text{normalized}} = \frac{A - \mu_A}{\sigma_A} \tag{8}$$

where μ_A and σ_A represent the batch mean and standard deviation respectively. Normalization ensures stable gradients and prevents divergence during training.