Paper 4: Reinforcement Learning-Inspired Adjustment Module

Formulation

This module adapts portfolio weights dynamically using a reward signal inspired by reinforcement learning. The update rule is given by:

$$w_{t+1} = w_t(1 + \eta r_t)$$

Where:

- w_t: Current weights at time t.
- η: Learning rate (controls sensitivity to rewards).
- r_t: Reward signal, e.g., normalized returns or alpha signals.

Why this Works

By directly linking weight updates to recent performance or rewards, the algorithm promotes assets that perform well and penalizes underperformers, creating an adaptive, feedback-driven strategy.

Algorithm Steps

1. Compute reward signal r_t (can be instantaneous return or derived alpha).
2. Apply linear multiplicative update to each weight.
3. Clip weights to ensure positivity.
4. Renormalize weights to sum to 1.

Code Equivalent

```
def rl_adjust_weights(weights, reward_signal, eta=0.01):
    new_weights = weights * (1 + eta * reward_signal)
    new_weights = np.maximum(new_weights, 1e-5)  # Avoid negative weights
    new_weights /= np.sum(new_weights)
    return new_weights

# Example usage:
reward = returns.iloc[t].values @ weights
weights = rl_adjust_weights(weights, reward, eta=0.01)
```

Summary

- Emphasizes dynamic learning behavior.
- Promotes self-reinforcing strategies.
- Reduces reliance on static signal thresholds.