Towards Practical PPO: Implementation and Validation of 8 PPO Optimization Methods Based on SB3

Abstract

Proximal Policy Optimization (PPO) is a widely used algorithm in the field of reinforcement learning, and Stable Baselines3 (SB3) provides an efficient basic implementation for it. However, there is still room for optimization in practical application scenarios such as complex environment adaptation, convergence speed, and stability. This paper aims to extend the PPO framework of SB3 by introducing 8 targeted improvement tricks to enhance algorithm performance, including dynamic clip_range adjustment (linear scheduling/KL divergence-based adaptive adjustment), Dual-Clip, Entropy decay, Winsorization (advantage clipping and normalization), PopArt value network normalization, Policy regularization, Actor-Critic network layer sharing (Totally Split/Deeply share with only different heads/Half share half split), and value function clipping (clip_range_vf). Experiments verified in typical environments such as CartPole-v1, LunarLander-v3, and MountainCarContinuous-v0 show that most tricks are effective: dynamic clip_range, Dual-Clip, Entropy decay, Actor-Critic deep sharing, and Policy regularization in specific scenarios can significantly accelerate convergence speed, improve reward peaks, and enhance training stability; Winsorization and PopArt have no obvious improvement effects but do not impair basic performance; value function clipping optimizes the rationality of value estimation. The extended PPO framework in this paper enriches the functional options of SB3, enhances the algorithm's adaptability to different scenarios, and provides a more flexible and efficient solution for the practical application of reinforcement learning.

近端策略优化（PPO）是强化学习领域应用广泛的算法，Stable Baselines3（SB3）为其提供了高效的基础实现，但在复杂环境适配、收敛速度与稳定性等实际应用场景中仍有优化空间。本文旨在扩展SB3的PPO框架，引入8种针对性改进trick以提升算法性能，具体包括：动态调整clip_range（线性调度/基于KL散度自适应）、双裁剪（Dual-Clip）、熵系数衰减（Entropy decay）、优势函数截断与归一化（Winsorize）、PopArt价值网络归一化、策略正则化（Policy regularization）、Actor-Critic网络层共享（完全分离/深度共享/半共享半分离）及价值函数裁剪（clip_range_vf）。实验在CartPole-v1、LunarLander-v3、MountainCarContinuous-v0等典型环境中验证表明，大部分trick表现有效：动态clip_range、双裁剪、熵系数衰减、Actor-Critic深度共享及特定场景下的策略正则化，能显著加

快收敛速度、提升奖励峰值与训练稳定性；优势函数截断与PopArt效果虽不明显，但未损害基础性能；价值函数裁剪则优化了价值估计的合理性。本文扩展后的PPO框架丰富了SB3的功能选项，增强了算法对不同场景的适配性，为强化学习实际应用提供了更灵活高效的解决方案。

Implementation Details in stable_baseline3

1. Set training mode
2. Update clip_range:
    a. **float** for constant clip range, **callable schedule** for a schedule function, for example linear

```
Code block
__init__(clip_range: Union[float, Schedule] = 0.2,
self.clip_range = clip_range
...
self.clip_range = FloatSchedule(self.clip_range)
...
clip_range = self.clip_range(self._current_progress_remaining)
...
# param value_schedule: Constant value or callable schedule
# (e.g. LinearSchedule, ConstantSchedule)
if isinstance(value_schedule, FloatSchedule):
    self.value_schedule: Schedule = value_schedule.value_schedule
elif isinstance(value_schedule, (float, int)):
    self.value_schedule = ConstantSchedule(float(value_schedule))
```

    b. An instance of a schedule function:

```
Code block
from stable_baselines3.common.schedules import LinearSchedule,
ConstantSchedule
clip_range_scheduler = LinearSchedule(  # 训练10000步后，clip_range降到0.01
并保持
    initial_value=0.2, final_value=0.01,schedule_timesteps=10000)
PPO(...,clip_range=clip_range_scheduler,...)
```

    c. clip_range_vf: clip range for the value function

1. Feature and observation:
    a. features 是从观测值 obs 中提取的特征向量:

```
Code block
```

```
features = self.extract_features(obs)
if self.share_features_extractor:
    latent_pi, latent_vf = self.mlp_extractor(features)
else:
    pi_features, vf_features = features
    latent_pi = self.mlp_extractor.forward_actor(pi_features)
    latent_vf = self.mlp_extractor.forward_critic(vf_features)
```

i. obs （observations）：原始的观测值，比如图像、传感器数据等

ii. features：经过预处理和特征提取后的表示

iii. extract_features() 对 obs 进行预处理（如归一化、转置), 通过extractor（CNN、Flatten）提取特征

iv. latent_pi and latent_vf: 通过 MLP从 features 进一步提取的高级特征, 预测action的log prob和value，entropy

b. Share network definition

Code block

```
self.features_extractor = self.make_features_extractor()
self.features_dim = self.features_extractor.features_dim
if self.share_features_extractor:
    self.pi_features_extractor = self.features_extractor
    self.vf_features_extractor = self.features_extractor
else:
    self.pi_features_extractor = self.features_extractor
    self.vf_features_extractor = self.make_features_extractor()
```

2. Advantage normalization:

Code block

```
advantages = rollout_data.advantages
# Normalization does not make sense if mini batchsize == 1, see GH issue #325
if self.normalize_advantage and len(advantages) > 1:
    advantages = (advantages - advantages.mean()) / (advantages.std() + 1e-8)
```

3. Policy network loss: clip with clip_range:

Code block

```
ratio = th.exp(log_prob - rollout_data.old_log_prob)
policy_loss_1 = advantages * ratio
policy_loss_2 = advantages * th.clamp(ratio, 1 - clip_range, 1 + clip_range)
policy_loss = -th.min(policy_loss_1, policy_loss_2).mean()
# clip_fraction:记录在当前 batch 中有多少比例的样本触发了 clipping 机制
```

a. To max policy_adv = advantages * ratio, loss is defined as -policy_adv

b. To clip, we use the smaller (th.min) one of (policy_adv, clipped_policy_adv)

4. Value network loss:

Code block

```
if self.clip_range_vf is None: values_pred = values
else: # Clip the difference between old and new value
 # NOTE: this depends on the reward scaling
    values_pred = rollout_data.old_values + th.clamp(
        values - rollout_data.old_values, -clip_range_vf, clip_range_vf)
        # Value loss using the TD(gae_lambda) target
value_loss = F.mse_loss(rollout_data.returns, values_pred)
```

a. Note: here it clips the (values - old_values), which is more reasonable than directly clipping the values:

i. We must be more careful about the scale of the values, which is hard to estimate exactly

b. GAE:

5. Entropy:

Code block

```
distribution = self._get_action_dist_from_latent(latent_pi)
entropy = distribution.entropy()
_, _, entropy = self.policy.evaluate_actions(rollout_data.observations,
actions)
entropy_loss = -th.mean(entropy)
```

a. Motivation of entropy:

i. $H(\pi) = -\Sigma\, \pi(a|s) \log \pi(a|s)$

ii. Min entropy loss -> maximize $H(\pi)$ -> 鼓励探索，防止收敛到local optimal

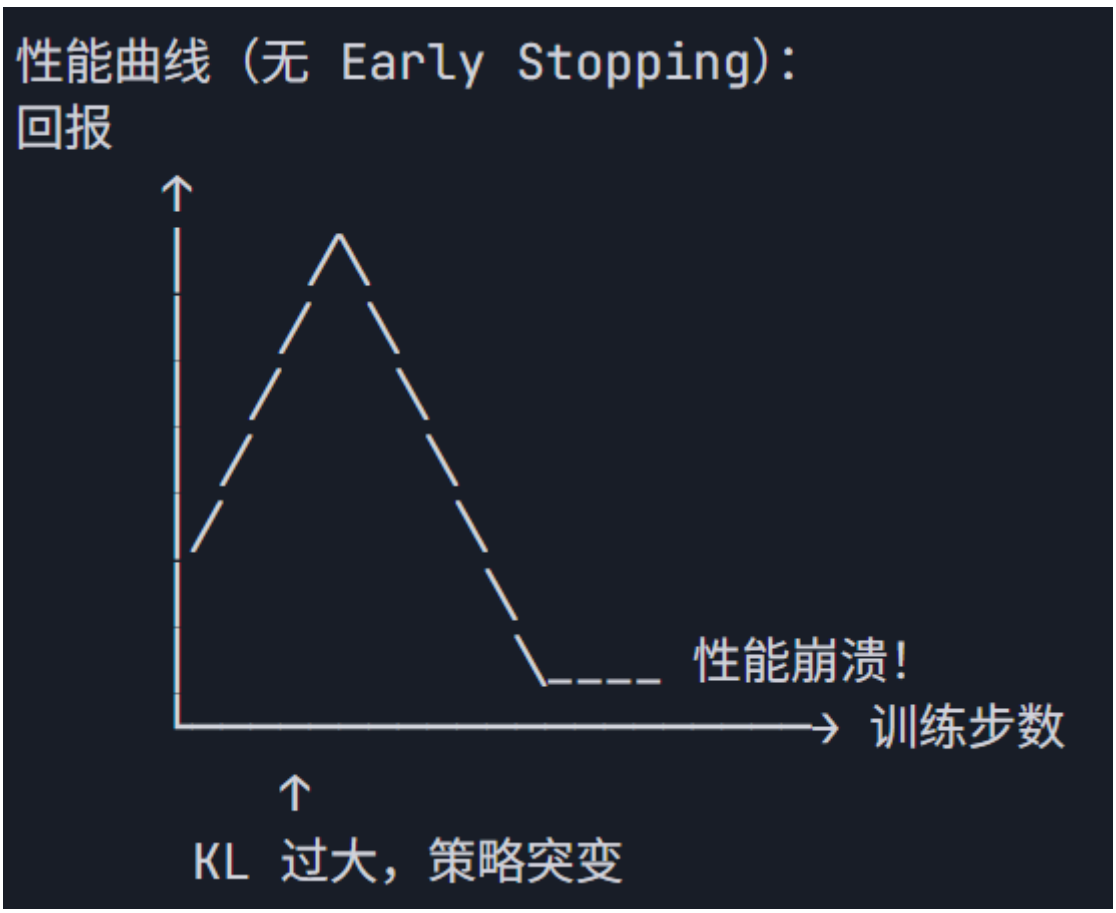
6. Construct loss:

```
Code block
    loss = policy_loss + self.ent_coef * entropy_loss + self.vf_coef *
    value_loss
```

7. KL Divergence:

```
Code block
1   with th.no_grad():
2       log_ratio = log_prob - rollout_data.old_log_prob
3       approx_kl_div = th.mean((th.exp(log_ratio) - 1) -
    log_ratio).cpu().numpy()
4   if self.target_kl is not None and approx_kl_div > 1.5 * self.target_kl:
5       break
```

   a. Why should we stop when being too large: 策略变化剧烈
      i. 数据失效： 用旧策略收集的数据不再代表新策略的行为: 需要用新策略重新收集数据
      ii. 梯度估计错误： 基于错误数据的梯度会误导训练, then性能崩溃： 策略可能突然变差

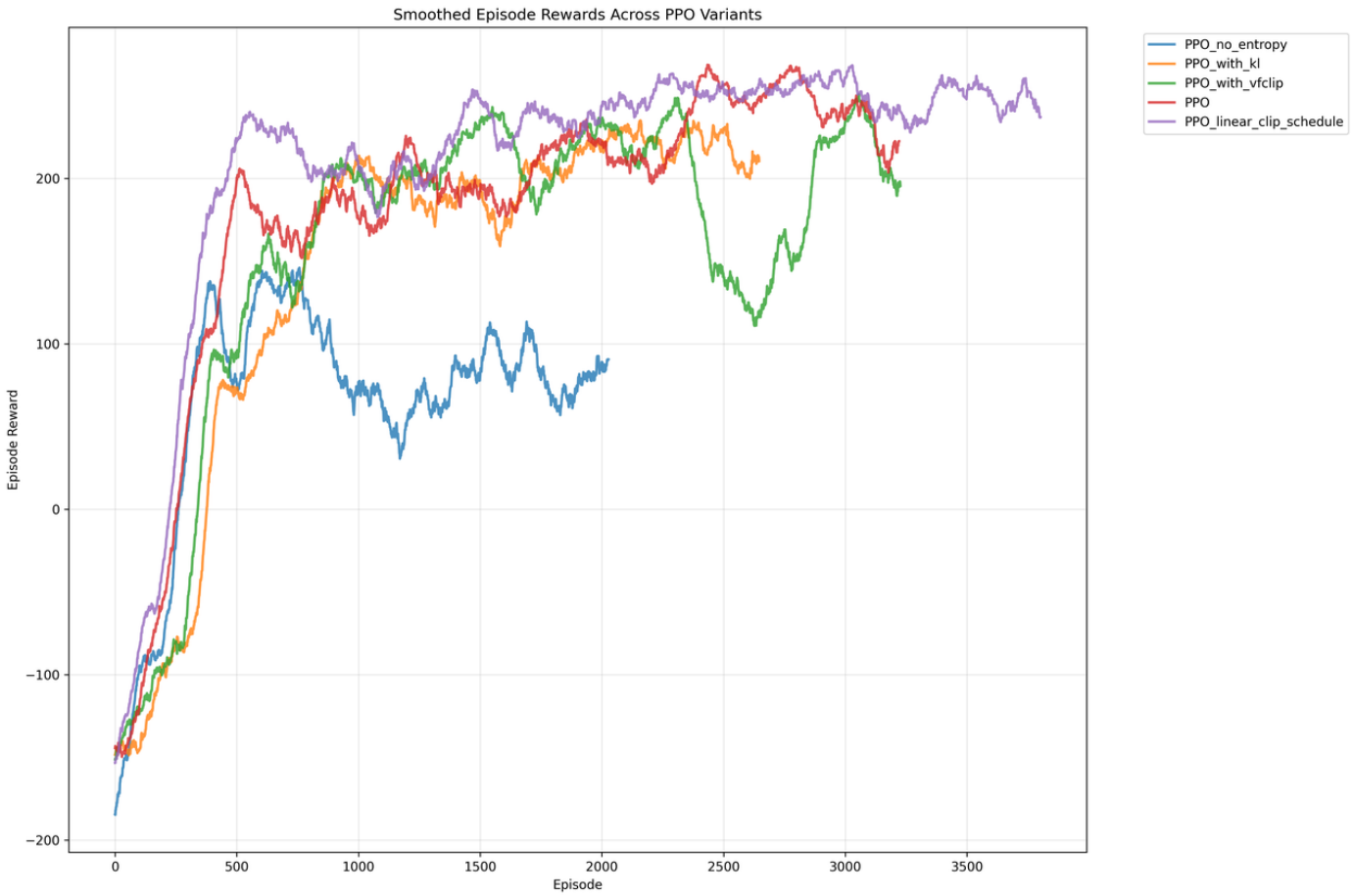


      iii.

8.

Performance comparison for different settings

1. Linear range clip: clip_range



2. 123

tricks

Cliping on loss

Asymmetric Clip（Dual-Clip）

1. Motivation: The standard PPO only performs clipping on the probability ratio \(r\) within the range of $1 \pm \varepsilon$. However, when the advantage $A < 0$ (i.e., the action is suboptimal), an extreme value of $r * A$ will result in **over-punishment** of the policy (for instance, a sudden massive negative reward received in one step may cause the policy to completely abandon this action).
2. Idea: When advantage <0: the ratio undergoes **asymmetric clipping** — the upper bound remains 1+ε, while the lower bound is relaxed from 1−ε to c (c<1−ε, where c is a positive value). It means: when the ratio is very small, that ratio < 1-ε, we allow a smaller c to clip the ratio but not 1-ε to avoid that using the larger 1-ε would lead to over-punishment
3. Implementation:

```
Code block
```

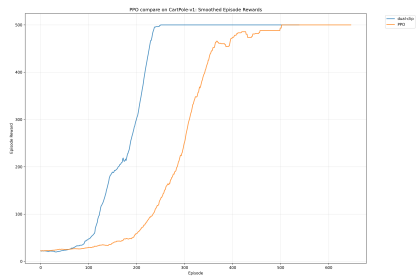
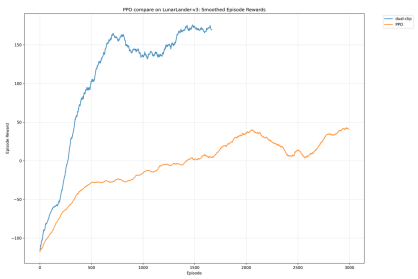
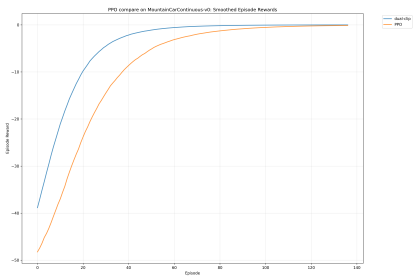
```
ε = 0.2, c = 0.6 # 核心：c>0 and c < 1-ε），负优势时的ratio下界（放宽后的阈值，<1-ε=0.8）
ratio = torch.exp(new_log_prob - old_log_prob)  # 概率比r
# 核心：按A的符号做非对称clip（Dual-Clip的极简实现）
clipped_ratio = torch.where(advantages >= 0,
    torch.clamp(ratio, 1-ε, 1+ε),
    torch.clamp(ratio, c, 1+ε) # advantages <0（差动作）→ 非对称clip，下界放宽到c，上界不变
)
policy_loss = -torch.min(ratio * advantages, clipped_ratio * advantages).mean()
```

4. Setting and Result:
   a. Setting:

Code block

```
"PPO_dual_clip": {
            "clip_range": 0.2,
            "clip_range_vf": None,
            "ent_coef": 0.01,
            "target_kl": None,
            "learning_rate": 3e-4,
            "n_steps": 2048,
            "batch_size": 64,
            "n_epochs": 10,
            "gae_lambda": 0.95,
            "max_grad_norm": 0.5,
            "device": "cpu",
#
            "dual_clip": 0.5,
        },
```

   b. Result:

    i. Observation: we observe that in the 3 environments, the initial performance of the Episode Reward in dual-clip (blue) is better than or similar to that of PPO (orange), and the rate of increase is faster and the rounds with stable high rewards are reached earlier.

    ii. Analysis: 模型训练初期，由累积奖励计算得到的目标值存在偏差，降低对负价值的惩罚力度，能稳定正向探索（优势值 > 0）的方向，避免单次负反馈导致探索偏航，减少训练方向修正的额外成本。

5. Applicable scenario: The dual-clip PPO core is adapted for reinforcement learning tasks involving continuous/discrete motion control, **sparse** rewards, and **training oscillations**; precise trajectory tracking for intelligent robots (such as robotic arm grasping) is a typical application scenario.

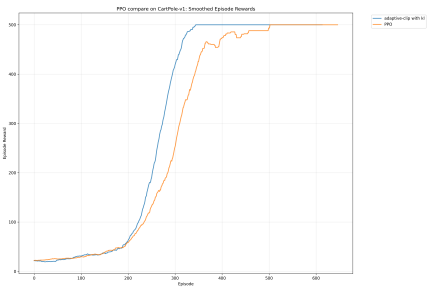
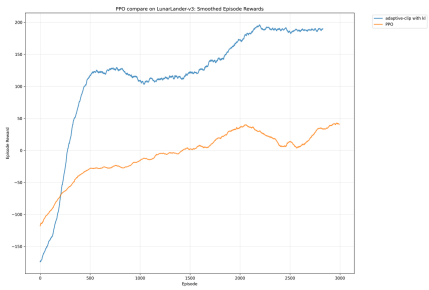
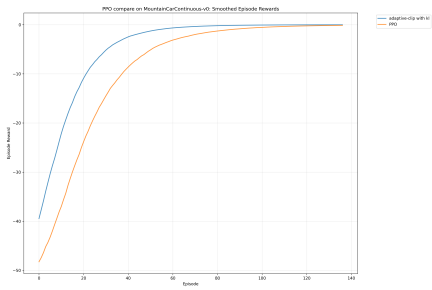
Adaptable clip range with KL

1. Motivation: A fixed ε (ClipRange) cannot adapt to the policy update requirements of different training phases:
    a. An excessively large ε will cause fast policy updates, leading to a high deviation between the new and old policies. This violates PPO’ s idea of constraining the update magnitude, triggering policy oscillation, or forgetting the effective knowledge in the old policy
    b. An excessively small ε will make the policy prone to falling into local optima in the later stages of training.
2. Idea:
    a. The KL divergence(KL) serves as a direct metric for measuring the distribution between the old and new policies. Dynamically adjusting the update strength via KL is more flexible than using fixed parameters:
        i. KL persistently low, policy overly conservative updates → increase ε
        ii. KL consistently high, policy overly aggressive updates → decrease ε
3. Implementation

Code block

```
clip_range = 0.2, target_kl = 0.01  # KL目标值（连续控制常用0.01，离散0.02）
kl_list = np.mean([kl.item() for kl in kl_tensor_list])  # 计算平均KL
if avg_kl < 0.5 * target_kl: # KL过小 → 策略更新太保守，增大clip_range（探索更激进）
    clip_range = min(clip_range * 1.5, 0.5)  # 乘1.5增大，上限0.5避免过大
elif avg_kl > 1.5 * target_kl: # KL过大 → 策略更新太激进，减小clip_range（更保守）
    clip_range = max(clip_range * 0.5, 0.1)  # 乘0.5减小，下限0.1避免过小
else:
    clip_range = clip_range  # 保持当前值（可加注释说明，或直接pass）
```

4. Result

   a. Observation: Compared to PPO, adaptive-clip with kl converges faster and reaches stable high rewards earlier in multiple reinforcement learning environments, and has better overall reward performance.

   b. analysis: adaptive-clip with kl表现更优，核心是通过KL散度动态调整ClipRange（ε），解决了固定ε无法适配不同训练阶段的问题：训练初期，动态ε既避免了固定ε过大引发的策略震荡、知识遗忘，也规避了ε过小导致的更新过慢，实现奖励快速提升与快速收敛；训练中后期，可根据KL高低灵活调ε，KL低则增大ε以跳出局部最优，KL高则减小ε以抑制激进更新，避免了固定ε的PPO易陷入局部最优、奖励持续波动的问题，因此能稳定维持高奖励表现。

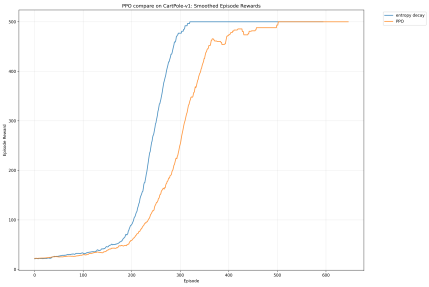
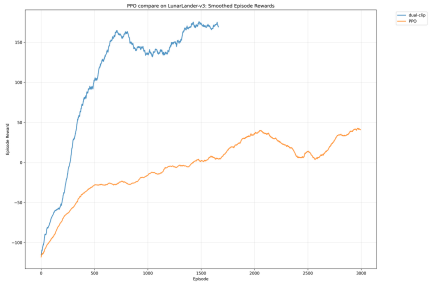
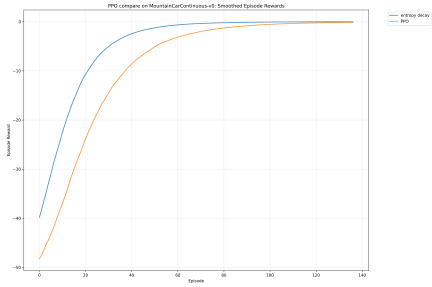
5. Applicable scenario: same as dual clip PPO.

Entropy decay

1. Motivation: The impact of a fixed entropy coefficient on training: If the entropy weight is too low, the policy tends to rapidly converge to locally optimal deterministic behavior, losing exploration. If the entropy weight remains fixed throughout the training process, the policy will approach the optimal solution in the later training phase, yet continuous high-entropy exploration will cause behavioral oscillations, preventing stable convergence to the optimal policy.
2. Idea:
   a. Entropy decay, just like the learning rate decay and start from a large entropy value
3. Implementation:

```
Code block
1   if self.entropy_decay is not None:
2       self.ent_coef *= self.entropy_decay
```

4. Result

a. Observation: Compared to PPO, decaying the coefficient of the entropy with the number of steps converges faster and reaches stable high rewards earlier in multiple reinforcement learning environments, and has better overall reward performance.

b. Analysis: decaying the coefficient of entropy can make the policy converge faster.

5. Applicable scenario： applications that requires thorough exploration of the action/environment space in the early stages and rapid convergence to a stable optimal policy in the later stages, and it is a reinforcement learning task with high requirements for the balance between exploration and utilization.

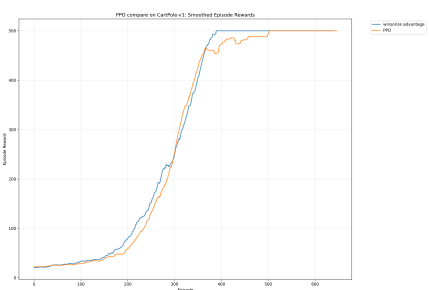
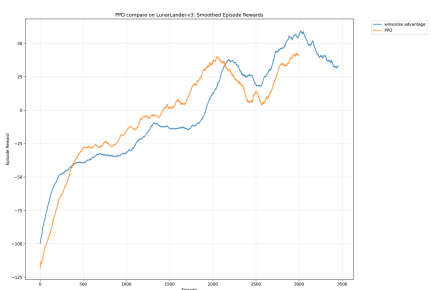
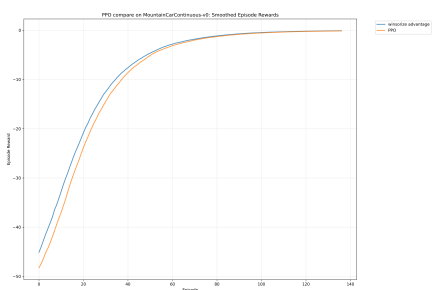
Advantage processing

Winsorize (clip advantage)

1. Motivation: simple normalization fails to suppress extreme values beyond the 99th percentile, which still dominate gradient descent. We need to stop gradient skew and boosts the model’ s core pattern learning.
2. Idea: first clipping out-of-range values, similar to cutting off the top/bottom scores. Normalizing the clipped data aligns its distribution with the true features of most samples.
3. Implementation in SB3

```
lower = np.percentile(advantages, bound)
upper = np.percentile(advantages, 100-bound)
advantages_clipped = np.clip(advantages, lower, upper)
```

4. Result

a. Observation: The improvement is not very obvious.

b. Analysis: although some extreme advantages may lead to very unstable update of the policy. It also makes the mdoel miss some important updates that really works.

5. Applicable scenario: Suitable for tasks with extremely long reward distribution tails (such as Atari's super high scores, sudden rewards in real-world environments), and scenarios with large fluctuations in rewards.

Training

PopArt

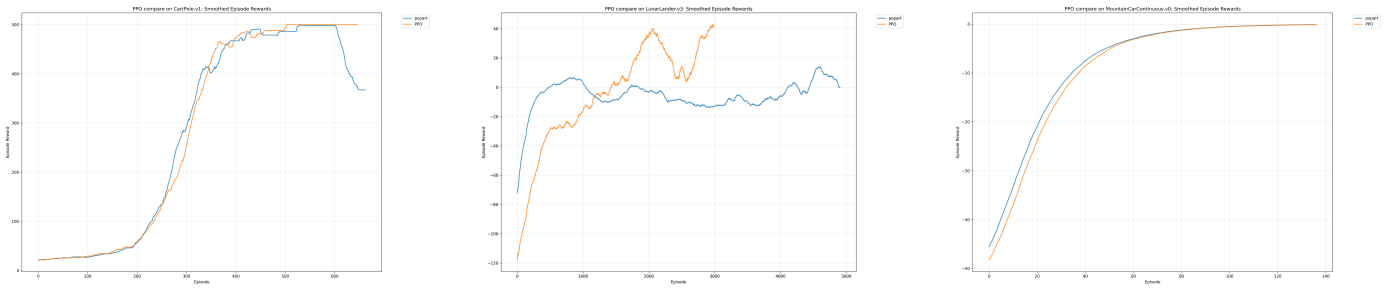
1. motivation: full-network normalization that both norm value and policy network impairs policy network stability by contaminating its representation space, creating a need for a method to improve value estimation without disrupting policy learning.
2. Idea：adjusts the value head’ s output layer to fit return distributions, preserving policy network representation space and balancing value accuracy with policy stability.
3. Implementation in SB3

Code block

```
class ValueHead(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.linear = nn.Linear(input_dim, 1)
        self.mean = nn.Parameter(th.zeros(1), requires_grad=False)
        self.var = nn.Parameter(th.ones(1), requires_grad=False)
    def update_popart(self, targets):
        batch_mean = targets.mean()
        batch_var = targets.var()
        delta = batch_mean - self.mean
        self.mean.data = self.mean + delta * 0.1  # sliding update
(coefficient 0.1)
        self.var.data = 0.9 * self.var + 0.1 * batch_var
    def forward(self, x):
        x = self.linear(x)
        return x * th.sqrt(self.var + 1e-8) + self.mean
...
if self.popart:
    self.value_net = ValueHead(self.mlp_extractor.latent_dim_vf)
else:
    self.value_net = nn.Linear(self.mlp_extractor.latent_dim_vf, 1)
...
if self.popart:
    self.policy.value_net.update_popart(rollout_data.returns)
```

```
24
```

4. Result



   a. Observation: it seems is not helpful to the performance.

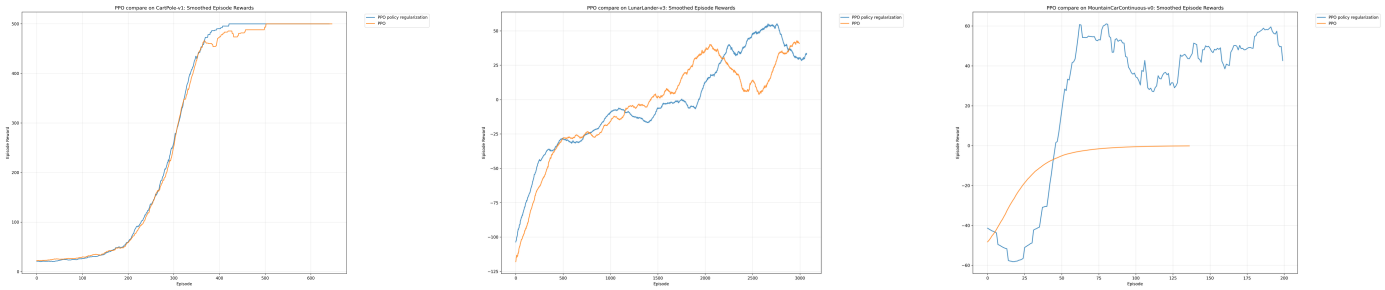
Policy regularization

1. Motivation: The policy network outputs action logits, but an excessively large logit scale will push the policy toward a one-hot distribution (i.e., only a single action is chosen), leading to an overly sharp KL divergence across policy iterations.
2. Idea: Incorporate a small-magnitude L2 regularization term on the action logits to restrict their scale, thus regularizing the policy to a smoother distribution.
3. Implementation in SB3

```
Code block

1   if self.policy_reg:
2       l2_reg = 1e-4 * th.norm(log_prob, p=2)
3       policy_loss += l2_reg
```

4. Result



   a. Observation: In CartPole-v1, the initial rewards for both are similar, with blue reaching 500 earlier and stabilizing, while orange catches up later. In LunarLander-v3, blue has a lower initial reward but rises rapidly, while orange rises slowly, In MountainCarContinuous-v0, blue first decreases and then rises, quickly climbing with significant fluctuations, while orange rises slowly to near 0, with blue consistently offering significantly higher rewards.

b. analysis： CartPole-v1：离散低维动作空间，logit 尺度天然受限；LunarLander-v3：高复杂度混合 / 连续动作空间，logit 尺度被任务稀释；MountainCarContinuous-v0：纯低维连续动作空间，任务驱动 logit 尺度急剧放大。

5. Applicable scenario：纯低维连续动作空间，任务聚焦单一动作

Network design

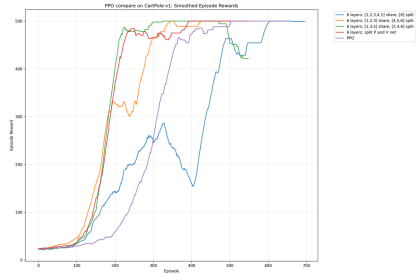
Shared layers for actor and critic

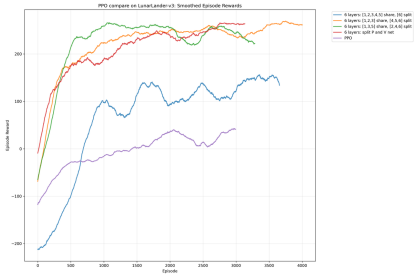
1. Motivation: Both the Actor (policy network, outputting action distribution) and the Critic (value network, outputting state value) need to extract core features (such as position, velocity, and pose) from the environmental state, and their underlying feature learning requirements highly overlap. Completely separating the networks would lead to parameter redundancy, increasing training computation and the risk of overfitting. Sharing underlying feature layers allows for the reuse of feature extraction capabilities, reduces the number of parameters, and improves training efficiency. Shared layers enable the Actor and Critic to learn consistent state representations, avoiding training oscillations caused by inconsistent feature learning, and reducing the probability of overfitting in scenarios with limited data.
2. Idea
    a. 2.1 Totally Split: Actor 与 Critic 使用**完全独立的神经网络**，无任何共享层，各自独立提取状态特征、完成任务输出（Actor 输出动作分布，Critic 输出状态价值）。特点：策略与价值更新完全解耦，但参数冗余度高、训练效率低。
    b. Deeply share, only diff head: 构建**共享的底层全部网络层**，仅在顶层分离输出头：共享层提取通用状态特征，Actor 头输出动作分布参数（如连续动作的高斯均值 / 方差、离散动作的 logits），Critic 头输出状态价值 V(s)。特点：参数效率最高、训练速度最快，适合特征需求高度重合的场景。
    c. Half share half split: 底层**部分网络层共享**（提取通用状态特征），上层**部分网络层分离**（分别学习策略、价值的专属特征），最后接各自的输出头。特点：平衡 “共享的效率” 与 “分离的灵活性”，适配特征复杂度中等的场景。
3. Implementation in SB3:
    a. Add a new policy class that support general-purpose Actor-Critic policy that allows flexible configuration of shared layers.

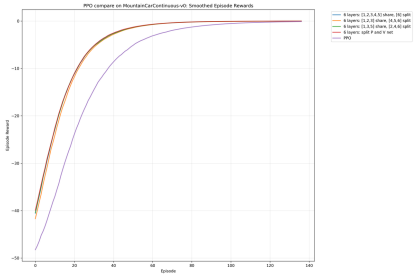
```
Code block
1   class CustomSharedPolicy(ActorCriticPolicy):...
```

4. Result

a. Network strucutures







i. 蓝色：6 layers [1,2,3,4,5] share, [6] split（深度共享，仅最后一层分离）

ii. 橙色：6 layers [1,2,3] share, [4,5,6] split（半共享）

iii. 绿色：6 layers [1,3,5] share, [2,4,6] split（半共享）

iv. 红色：6 layers: split P and V net（完全分离）

v. 紫色：PPO（默认深度共享结构）

b. Actor-Critic 共享底层层（尤其是深度共享）能显著提升收敛速度与最终性能，完全分离因参数冗余、特征解耦不足表现最差。

5. Applicable scenario: 所有Actor-Critic 架构的强化学习控制任务，尤其是数据有限、训练资源受限、需要快速收敛的场景，共享层能有效提升参数效率与训练稳定性。