Reinforcement Learning: Crafter Project

Tumelo Mkwambe
2446873

Karabo Ledwaba
2569393

I. Proximal Policy Optimization

Proximal Policy Optimization (PPO) [1] is a policy gradient algorithm designed to achieve a balance between efficient learning and stable policy updates. It is adapted from Trust Region Policy Optimization (TRPO) [2], which constrains policy updates using the Kullback–Leibler (KL) divergence. Instead of enforcing a hard constraint, PPO introduces a clipped objective function that limits the magnitude of policy changes, making it less computationally intensive while maintaining comparable performance. The algorithm alternates between sampling data through interaction with the environment (referred to as rollout collection) and optimizing a surrogate objective function using gradient descent.

A. Rollout Collection

Given an actor-critic policy that simultaneously predicts actions and value estimates, the agent interacts with the environment for a fixed number of steps n. During each step, the policy outputs an action a_t sampled from a categorical distribution parameterized by the policy's logits $\pi_\theta(a_t|s_t)$, and the environment returns the next observation s_{t+1}, reward r_t, and termination signal. The collected trajectories $(s_t, a_t, r_t, \pi_\theta(a_t|s_t), V_\theta(s_t))$ are stored until a predefined horizon n_{steps} is reached.

After collecting rollouts, the advantages $\hat{A}_t$ are computed using Generalized Advantage Estimation (GAE) [3]:

$$\hat{A}_t = \delta_t + \gamma\lambda\hat{A}_{t+1},$$

where $\delta_t = r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)$, γ is the discount factor, and λ is a smoothing parameter that trades off bias and variance.

The advantage estimates represent how much better an action selected using the old policy is compared to the expected value under the current policy, helping to reduce variance and stabilize learning.

Finally, the empirical returns $\hat{R}_t$ are obtained by combining the estimated advantages with the predicted state values:

$$\hat{R}_t = \hat{A}_t + V_\theta(s_t).$$

These returns serve as targets for training the value function (optimizing the critic), while the advantages guide updates to the policy parameters during optimization.

B. Policy Optimization

Collected rollouts (observations, old policy actions, old policy log probabilities, returns, advantages) are organized in batches, which are used to optimize the actor or policy network and critic network. For each batch, we compute new log probabilities of the sampled actions under the updated policy $\pi_\theta(a_t|s_t)$ and evaluate the probability ratio:

$$r_\theta(t) = \frac{\pi_{\theta_{old}}(a_t|s_t)}{\pi_\theta(a_t|s_t)}.$$

The ratio $r_t(\theta)$ measures how much the new policy deviates from the old one for the same action. Proximal Policy Optimization (PPO) constrains this deviation using a clipped surrogate objective to prevent large, destabilizing policy updates:

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

where ϵ is a small hyperparameter that limits the range of policy updates. This objective ensures that if the new policy diverges too far from the old one, the update is suppressed, promoting stable and conservative learning.

The critic or value function is trained by minimizing the squared error between predicted values and the empirical returns:

$$L^{VF}(\theta) = \mathbb{E}\left[(\hat{R}_t - V_\theta(s_t))^2\right].$$

The final loss function combines both objectives with an entropy bonus $L^{\text{ENT}}(\theta)$ to encourage exploration:

$$L(\theta) = \mathbb{E}\left[L^{CLIP} - c_1L^{VF} + c_2L^{ENT}\right]$$

where c_1 and c_2 are coefficients that balance the contributions of the value loss and entropy regularization, respectively.

The policy optimization step is repeated for a specified number of epochs. The two steps (rollout collection and policy optimization) are repeated until a specified number of steps is reached. The full algorithm is shown in figure 4

C. Models Improvements

1) PPO-Agent-I: The base agent (ppo-agent-i) employed an actor-critic architecture with a ResNet50 feature extractor. A pretrained ResNet50 model was used, with its final classification layer removed to obtain high-level feature representations

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

Fig. 1. PPO-Clip Pseudocode From OpenAI Spinning Up

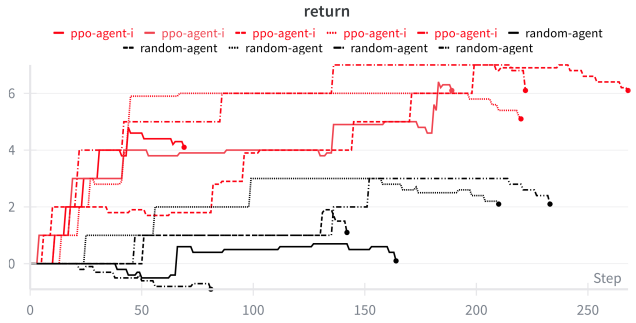

Fig. 2. PPO-Agent-I vs Random Agent

from input observations. These features were passed to two separate heads: an actor head, which produced a categorical distribution over the available actions, and a critic head, which estimated the state value.

The agent was trained using the Proximal Policy Optimization (PPO) algorithm as described above, optimizing the following surrogate objective:

$$L(\theta) = \mathbb{E}\left[L^{CLIP} - c_1 L^{VF} + c_2 L^{ENT} \right],$$

where L^{CLIP} represents the clipped policy loss, L^{VF} is the value function loss, and L^{ENT} is the entropy bonus that encourages exploration.

2) PPO-Agent-II: The PPO algorithm incorporated a soft KL divergence check during policy optimization to prevent excessively large updates to the policy within a single batch. This mechanism allows the algorithm to halt updates within the current batch if the policy changes too much, helping maintain stable learning and preventing catastrophic performance drops.

Although the ResNet50 feature extractor provided rich visual representations, it proved to be suboptimal for reinforcement learning in this setting. ResNet architectures are highly effective in supervised learning or off-policy methods

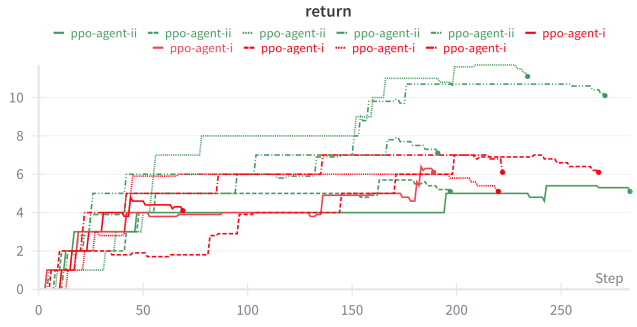

Fig. 3. PPO-Agent-I vs PPO-Agent-II

where large, independent batches of data are available. In PPO, however, the batches are smaller and observations are strongly correlated in time, and policy updates can slightly alter the observation distribution between epochs. This can lead to drifting features, unstable advantage estimates, and slower learning, ultimately resulting in lower returns.

To address these limitations, the ResNet50 backbone was replaced with the NatureCNN feature extractor [4]. This lightweight convolutional architecture, also referred to as "CnnPolicy" in Stable-Baselines3, was specifically designed for visual reinforcement learning tasks. NatureCNN uses three convolutional layers followed by a fully connected layer to extract spatial and semantic features from input images and produce a compact latent representation. This design balances representational power with computational efficiency, allowing faster policy updates, more stable advantage estimates, and improved sample efficiency. When applied to the PPO agent, the NatureCNN extractor led to more stable learning and higher average returns compared to the ResNet-based model.

D. PPO-Agent-III

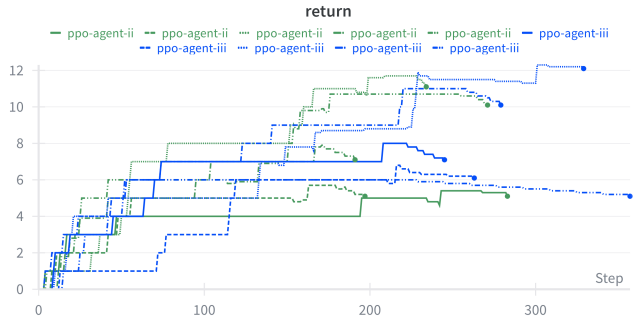

Fig. 4. PPO-Agent-II vs PPO-Agent-III

PPO-Agent-III builds upon the previous implementation by removing the soft KL divergence check used to restrict large policy updates. While the KL threshold mechanism in PPO-Agent-II contributed to training stability, it occasionally caused premature termination of policy updates, limiting the extent of policy improvement within an epoch. By disabling this

constraint, PPO-Agent-III allows the policy to explore a wider update range, potentially accelerating learning at the cost of slightly increased variance.

Empirically, this modification resulted in a modest improvement in performance. The agent achieved slightly higher average returns and faster convergence, suggesting that in this specific environment, the risk of policy divergence was minimal and the added flexibility in updates allowed for more effective policy optimization.

See README.md to train and evaluate models https://github.com/TumeloMkwambe/Crafter/

REFERENCES

[1] Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

[2] Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization." In International conference on machine learning, pp. 1889-1897. PMLR, 2015.

[3] Schulman, John, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. "High-dimensional continuous control using generalized advantage estimation." arXiv preprint arXiv:1506.02438 (2015).

[4] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves et al. "Human-level control through deep reinforcement learning." nature 518, no. 7540 (2015): 529-533