Asynchronous Advantage Actor-Critic (A3C) Tutorial

September 21, 2025

1 Introduction

Asynchronous Advantage Actor-Critic (A3C) runs multiple agents in parallel, each updating a shared policy and value network. By decoupling workers and using asynchronous updates, A3C decorrelates experience, stabilizes learning without replay buffers, and exploits multi-core CPU hardware.

2 Theory and Formulas

2.1 Multi-worker Objective

Each worker maintains local parameters (θ', w') initialized from global parameters (θ, w). After collecting a rollout of length n, the worker computes multi-step advantages:

$$A_t = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_w(s_{t+n}) - V_w(s_t). \tag{1}$$

Gradients are accumulated locally before being applied to the shared parameters.

2.2 Asynchronous Updates

Policy and value gradients for each worker are

$$\nabla_\theta J \approx \sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, A_t + \beta \nabla_\theta H\big[\pi_\theta(\cdot \mid s_t)\big], \tag{2}$$

$$\nabla_w L_V \approx \sum_t \partial_w \frac{1}{2} A_t^2, \tag{3}$$

where β controls entropy regularization. After computing gradients, the worker atomically updates global parameters and syncs local copies.

2.3 Stability Considerations

Asynchronous execution increases gradient noise but improves exploration. Using smaller rollout lengths n, gradient clipping, and synchronized learning rates across workers maintains stability. Optimizers such as RMSprop with shared statistics are commonly used.

3 Applications and Tips

- **CPU-friendly training**: leverage multi-core servers without large replay buffers.
- **Sparse rewards**: parallel workers discover different trajectories, increasing successful rollouts.
- **Continuous and discrete control**: extend A3C with Gaussian actors or shared convolutional encoders.
- **Best practices**: synchronize parameters periodically, use per-worker learning rate annealing, and monitor gradient norms to detect divergence.

4 Python Practice

The script `gen_a3c_figures.py` emulates three asynchronous workers solving a cliff-world task. Each worker gathers short rollouts, computes multi-step advantages, and applies gradients to shared actor-critic tables. The script records the aggregated return curve and final policy probabilities.

Listing 1: Excerpt from $\text{gen}_a 3c_f igures.py$

```
for worker in range(num_workers):
    states, actions, rewards = collect_rollout(theta_local[worker],
       V_local[worker])
    advantages = compute_n_step_advantages(states, rewards, V_global)
    grad_theta, grad_V = accumulate_gradients(states, actions,
       advantages)
    theta_global += actor_lr * grad_theta
    V_global += critic_lr * grad_V
```

5 Result

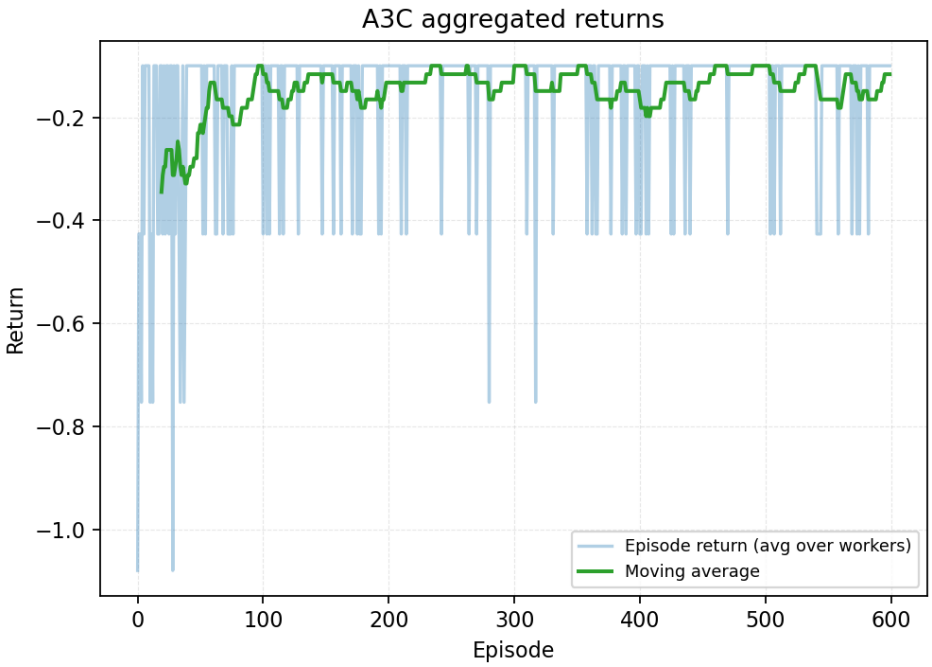

Figure 1: A3C episode returns aggregated across asynchronous workers

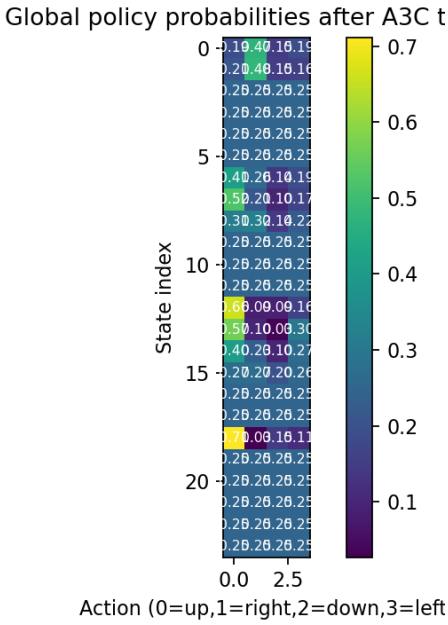

Figure 2: Action probability heatmap of the global policy after training

6 Summary

A3C combines multi-step advantage estimation with asynchronous workers to stabilize on-policy learning without replay buffers. Proper tuning of rollout length, entropy bonuses,

and optimizer settings balances gradient noise and convergence speed. The cliff-world example illustrates how worker diversity accelerates policy improvement and yields interpretable action probabilities.