Lab 5: Deep Reinforcement Learning: Comparing DQN, PPO, and A2C

Reinforcement Learning Course

1 Overview

In earlier labs, you explored the progression from *bandits* (no state) and exploration–exploitation to *Markov Decision Processes (MDPs)* and *dynamic programming* with known models, and then to *model-free control* with on-policy and off-policy tabular methods in `FrozenLake`. This lab is the capstone for the deep RL part of the course and extends those ideas to **neural function approximation** and **modern deep RL algorithms**.

The goal of lab is to compare *three* widely used families of deep RL on the same control problem:

1. **DQN** (Deep Q-Network): value-based, *off-policy* algorithm that learns an action–value function using experience replay and a target network.
2. **PPO** (Proximal Policy Optimization): *on-policy actor–critic* algorithm optimized via a clipped *policy-gradient* objective to ensure stable updates.
3. **A2C** (Advantage Actor–Critic): *on-policy actor–critic* method in which a stochastic policy (actor) is trained using policy gradients and a value function (critic) is used for variance reduction.

Runtime Note

To keep the lab **low-code**, time-efficient, and feasible within a **2-hour** session, you will rely on pre-implemented algorithms from `Stable-Baselines3 (SB3)` running on `Gymnasium` environments. We also provide **pre-executed code**, since running these algorithms step-by-step can be slow. In practice, DQN requires about **12 minutes**, PPO roughly **5 minutes**, and A2C around **17 minutes**. Pre-executed code ensures that you can focus on analyzing results and understanding reinforcement learning concepts rather than waiting for lengthy computations. However, feel free to rerun the algorithms; either with the default settings or with your own modified configurations; if you have the time.

Reproducibility Note

When running the notebook on the default **CPU runtime** in Google Colab, your results should be *identical* to the pre-executed outputs. SB3 and Gymnasium behave deterministically on CPU when a fixed seed is used.
However, if you switch to a **GPU** or **TPU** runtime or run the notebook locally on your own hardware, you may observe differences in learning curves or final returns. This is expected. Deep RL training relies on numerical operations implemented in low-level libraries such as BLAS, cuBLAS, and cuDNN, many of which are *not fully deterministic*. Additionally, GPU kernels, multithreaded CPU operations, and nondeterministic scheduling can produce slight floating-point differences. These tiny discrepancies accumulate over thousands of gradient updates, which can cause noticeable variation between runs.

The primary environment for this lab is `LunarLander-v3`, which uses discrete actions and offers richer dynamics along with a shaped reward function. However, you may also experiment with `CartPole-v1` using all three algorithms (DQN, PPO, and A2C) as a simpler comparison task, which is included as an optional exercise at the end of the lab.

You will **train agents**, **plot learning curves** (episode reward and length versus time), and **compare** speed, stability, and sample efficiency. The provided notebook also shows how to **record videos** of trained agents.

Key Ideas

DQN (Value-Based, Off-Policy). Approximate $Q_\theta(s, a)$ with a neural network and minimize the TD loss using a *target network* θ^-. With a replay buffer $\mathcal{D}$ and discount γ,

$$\mathcal{L}(\theta) \;=\; \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\Big[\big(r + \gamma \max_{a'} Q_{\theta^-}(s', a') - Q_\theta(s, a)\big)^2\Big].$$

Replay reduces correlation in updates; the target network stabilizes training by providing a slowly moving target.

PPO (Actor–Critic, On-Policy). Directly optimize a stochastic policy $\pi_\theta(a|s)$ with an *actor–critic* baseline and a *clipped* surrogate objective to prevent overly large updates:

$$\mathcal{L}^{\text{PPO}}(\theta) \;=\; \mathbb{E}\Big[\min\big(r_t(\theta)A_t,\; \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\big)\Big],$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ and A_t is an advantage estimate.

A2C (Actor–Critic, On-Policy). Jointly learn a policy and a value function $V_\phi(s)$. Use TD advantages to reduce variance:

$$A_t \;=\; \big(r_t + \gamma V_\phi(s_{t+1})\big) - V_\phi(s_t),$$

and maximize $\mathbb{E}[\log \pi_\theta(a_t|s_t)A_t] + \beta\,\mathcal{H}(\pi_\theta(\cdot|s_t))$ with entropy bonus β to encourage exploration.

Implementation note. As said, we use SB3 implementations of DQN, PPO, and A2C to focus on *interpretation* and *comparison*, not boilerplate coding.

Task 1: Getting Familiar with `LunarLander-v3`

Objective. Understand the `LunarLander-v3` environment under a **random policy**:

- What the state (observation vector) represents,
- Which discrete actions are available (e.g. fire main engine, side thrusters),
- How rewards are given and when episodes terminate,
- How well a random policy performs (episode length, total reward, "success" frequency).

Procedure. Using the provided notebook:

1. Run a single random episode and inspect:
   - The initial observation,
   - How the lander moves when actions are sampled uniformly.
2. Run many random episodes (e.g. 100) with episode statistics recording enabled, and compute:
   - Average and standard deviation of episode rewards,
   - Average episode length,
   - A simple success rate (fraction of episodes achieving a reward above a chosen threshold).
3. Record a short video of the random policy and watch it in the notebook.

Be prepared to explain.

- What each component of the observation vector represents (position, velocity, angle, leg contact, etc.).
- Which actions the agent can take and how they affect the lander.
- Typical performance of a random policy in terms of reward and stability.

Task 2: DQN on `LunarLander-v3` (Value-Based)

Objective. Train your first deep RL agent: a DQN on `LunarLander-v3`. The goal is to observe how a value-based deep RL algorithm behaves when combined with replay buffers and a target network.

Procedure.

1. Train DQN for a fixed number of timesteps.
2. Plot the episode rewards, episode lengths, and a smoothed training signal.
3. Evaluate the trained agent on several episodes and report mean and standard deviation of returns.
4. Record a short video of the trained DQN agent.

Be prepared to explain.

- Why DQN needs experience replay and a target network for stability.
- How value-based learning differs from the random policy in Task 1.
- What kind of landing behaviour DQN tends to produce (soft, hard, oscillatory, unstable, etc.).

Task 3: PPO on `LunarLander-v3` (Actor–critic)

Objective. Train a PPO agent and compare its learning behaviour to the DQN agent from Task 2. PPO is an on-policy actor–critic method and often behaves differently from value-based Q-learning.

Procedure.

1. Train PPO for a fixed number of timesteps.
2. Use the monitoring tools to plot episode rewards, episode lengths, and the training signal.
3. Evaluate the trained PPO agent on multiple episodes.
4. Record a short video of the trained PPO lander.

Be prepared to explain.

- How PPO's clipped objective controls update size and improves training stability.
- How PPO's performance and sample efficiency compare to the DQN results from Task 2.
- The differences in behaviour between the DQN lander and the PPO lander (from the videos).

Task 4: A2C on `LunarLander-v3` (Actor–critic)

Objective. Train an A2C (Advantage Actor–Critic) agent on `LunarLander-v3`. This method sits between DQN and PPO: it uses both a policy and a value function.

Procedure.

1. Train A2C with a fixed timestep budget.
2. Plot the episode rewards, episode lengths, and training signal.
3. Evaluate the trained A2C agent on several episodes.
4. Record a short video of the trained A2C lander.

Be prepared to explain.

- How the critic reduces variance in the policy-gradient update.
- How A2C compares to PPO and DQN in terms of learning stability, speed, and robustness.
- Differences visible in the landing behaviour of all three agents.

Task 5: Comparative Discussion on `LunarLander-v3`

Objective. After training all three algorithms (DQN, PPO, A2C), examine your plots and videos and think about how the methods differ in practice.

Reflect on the following questions. Use your learning curves, evaluation results, and recorded agent videos to form your thoughts:

- **Learning speed:** Which algorithm improved fastest in the early phase of training?
- **Stability:** Which learning curves were smooth, and which showed oscillations or collapse?
- **Final performance:** Which agent achieved the best overall behaviour after the training time?
- **Exploration:** How did ε-greedy exploration in DQN compare to stochastic exploration in PPO and A2C?
- **Behaviour in videos:** How do the landing behaviours differ across the three agents (e.g., smooth landing, oscillation, crash, hovering, over-correction)?
- **Algorithmic insight:** How do these differences relate to the underlying mechanisms:
  - value-based,
  - policy-gradient,
  - actor–critic,
  - on-policy vs. off-policy learning?

Task 6 (Optional): All Three Algorithms on `CartPole-v1`

Objective. As an optional extension, run DQN, PPO, and A2C on the simpler control problem `CartPole-v1` and compare the results to `LunarLander-v3`.

Procedure.

1. Use the provided notebook cells to train DQN, PPO, and A2C on `CartPole-v1`.
2. Plot and inspect their learning curves (episode reward and length).
3. Record brief videos of the trained agents on `CartPole-v1`.
4. Reflect on how algorithm behaviour differs between `CartPole-v1` and `LunarLander-v3`.

Be prepared to explain.

- Did all three algorithms find good policies quickly on `CartPole-v1` compared to `LunarLander-v3`?
- Whether the ranking of algorithms by performance or stability is the same across both environments.