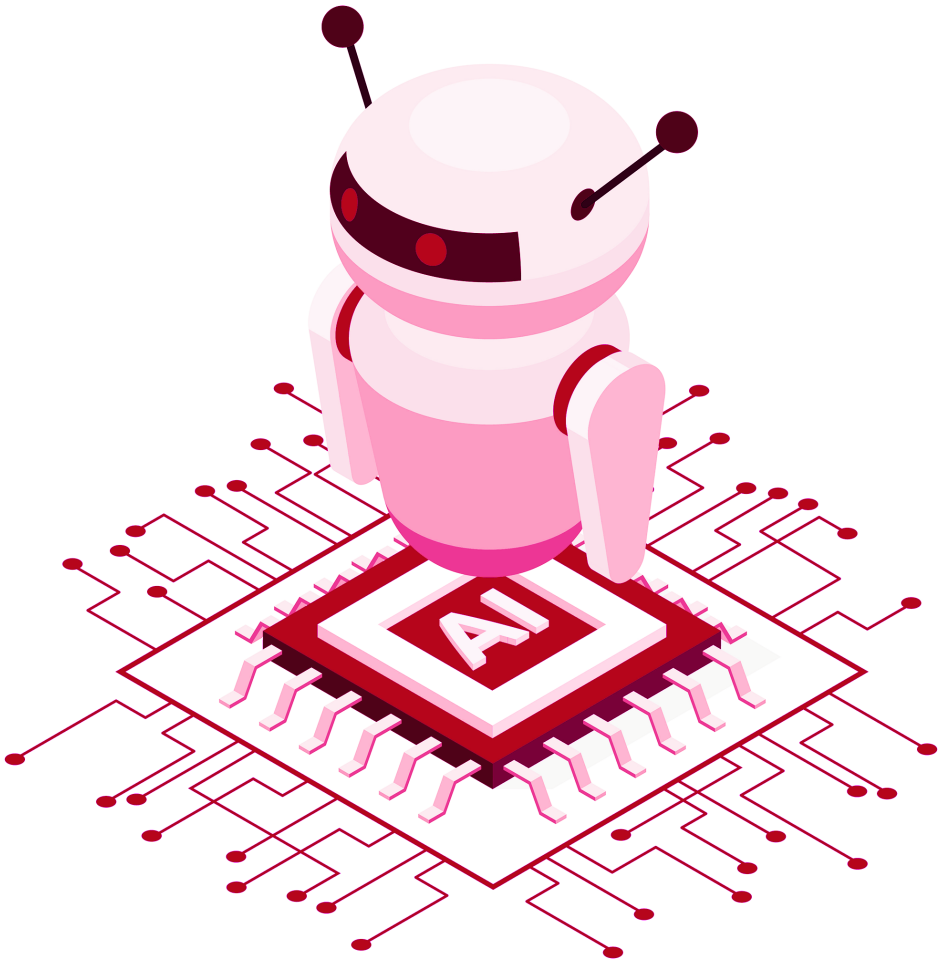

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 1:

Introduction to RL

By:

Ali Najar

401102701

RIML

Spring 2025

Contents

Grading

The grading will be based on the following criteria, with a total of 100 points:

Task	Points
Task 1: Solving Predefined Environments	45
Task 2: Creating Custom Environments	45
Clarity and Quality of Code	5
Clarity and Quality of Report	5
Bonus 1: Writing a wrapper for a known env	10
Bonus 2: Implementing pygame env	20
Bonus 3: Writing your report in Latex	10

Notes:

- Include well-commented code and relevant plots in your notebook.
- Clearly present all comparisons and analyses in your report.
- Ensure reproducibility by specifying all dependencies and configurations.

1 Task 1: Solving Predefined Environments [45-points]

Note: The outputs of cells are removed from the uploaded notebook. See this link for a version with outputs.

FrozenLake-v1:

we used FrozenLake-v1 as our first env. In this part we trained 10 models with hyperparameters as below:

- PPO_1 : "learning_rate": 0.0001, "gamma": 0.99, "timesteps": 50000
- PPO_2 : "learning_rate": 0.01, "gamma": 0.99, "timesteps": 50000
- PPO_3 : "learning_rate": 0.0001, "gamma": 0.7, "timesteps": 50000
- PPO_4 : "learning_rate": 0.0001, "gamma": 0.99, "timesteps": 5000
- PPO_5 (with reward wrapper) : "learning_rate": 0.0001, "gamma": 0.99, "timesteps": 50000
- DQN_1 : "learning_rate": 0.0001, "gamma": 0.99,"buffer_size"=1000000 , "timesteps": 100000
- DQN_2 : "learning_rate": 0.01, "gamma": 0.99,"buffer_size"=1000000 , "timesteps": 100000
- DQN_3 : "learning_rate": 0.0001, "gamma": 0.7 ,"buffer_size"=1000000, "timesteps": 100000
- DQN_4 : "learning_rate": 0.0001, "gamma": 0.99 ,"buffer_size"=100000, "timesteps": 100000
- DQN_5 (with reward wrapper) : "learning_rate": 0.0001, "gamma": 0.99 ,"buffer_size"=1000000, "timesteps": 100000

Learning Curves

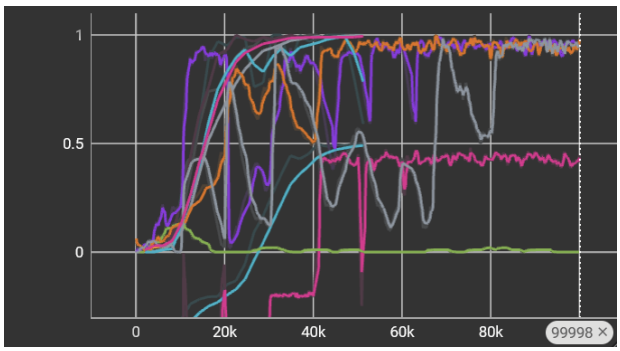

Figure 1: All models reward per episode curves

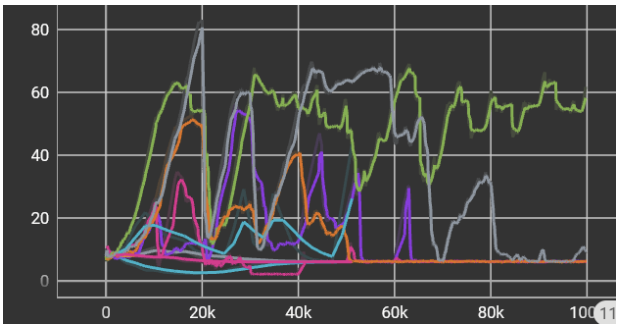

Figure 2: All models mean episode length curves

1. Sample Efficiency and Reward Analysis

From (3) it can be seen that PPO converges at almost 50000 timesteps quite smoothly, while DQN has many fluctuation and struggles to converge at even 100000 timesteps.

It can also be seen that both algorithm reach maximum possible reward of 1 in the end (Although DQN cannot converge very well).

2. Hyperparameter Comparison

- Learning Rate

  The above diagrams (4),(5) show that a learning rate of 0.0001 provides a good trade-off between convergence speed and stability. we can see that in PPO and DQN, lower learning rates lead to

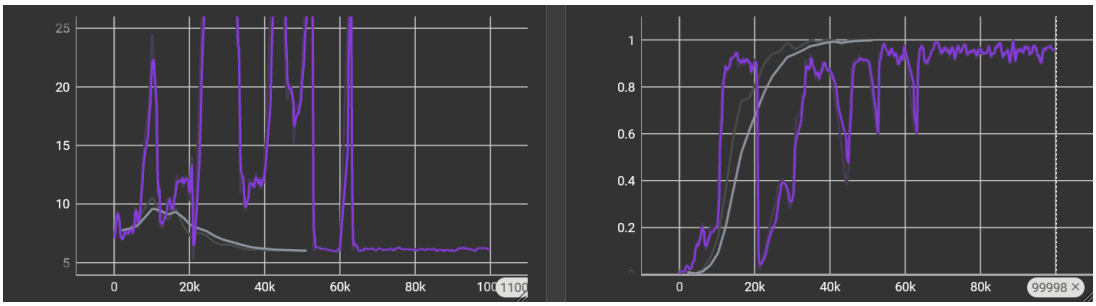

Figure 3: DQN_1 and PPO_1 ep_len_mean and ep_rew_mean comparison

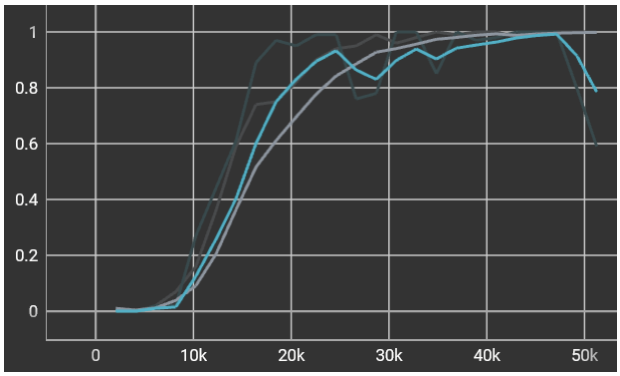

Figure 4: PPO_1 vs. PPO_2

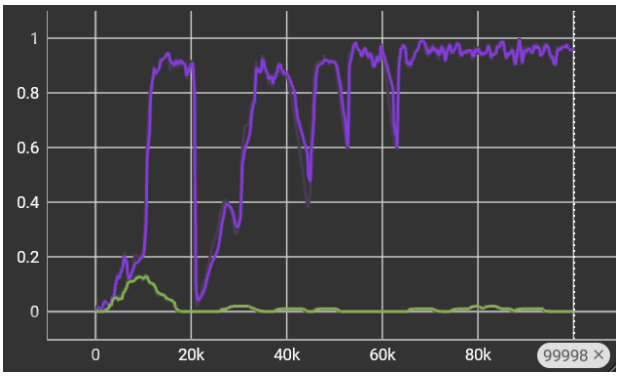

Figure 5: DQN_1 vs. DQN_2

better convergence. On the other hand, having high learning rates leads to aggressive updates, the optimizer may skip over the optimal region, causing oscillations or even divergence in training.

- γ Parameter

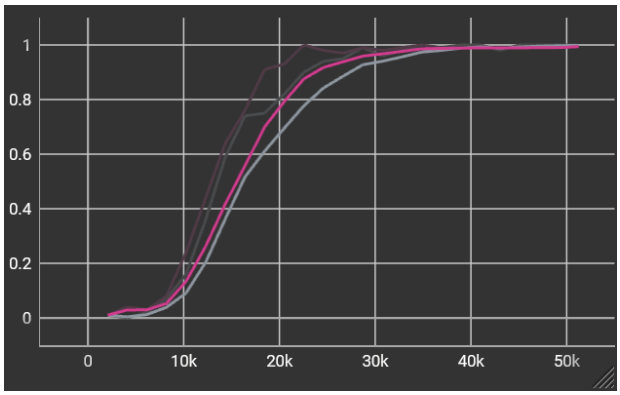

Figure 6: PPO_1 vs. PPO_3

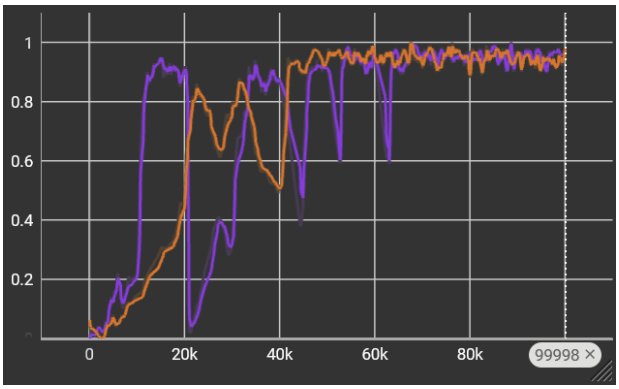

Figure 7: DQN_1 vs. DQN_3

From the above diagrams (6),(7) it can be seen that for PPO, a slightly lower γ (pink one) can lead to faster learning early on (because the agent focuses on immediate, more certain rewards).

DQN also suggests that lower γ (0.7) tends to learn faster since like PPO the agent focuses on immediate, more certain rewards. the orange plot shows DQN with $\gamma = 0.7$.

If the environment's rewards are strongly delayed, $\gamma = 0.7$ might not give the agent enough incentive to plan far ahead. If immediate rewards are sufficient, a smaller discount might be enough and could reduce learning instability, like the FrozenLake env.

- Timesteps

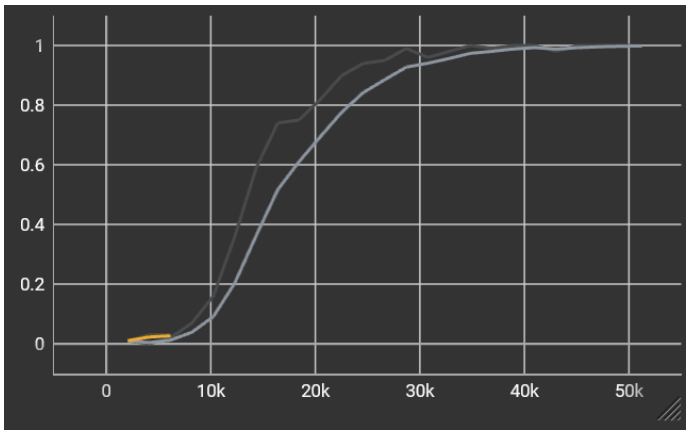

Figure 8: PPO_1 vs. PPO_4

It is obvious that the agent needs enough timesteps to find the optimal policy. here it is evident that 5000 timesteps is not enough for PPO to converge.

- buffer_size

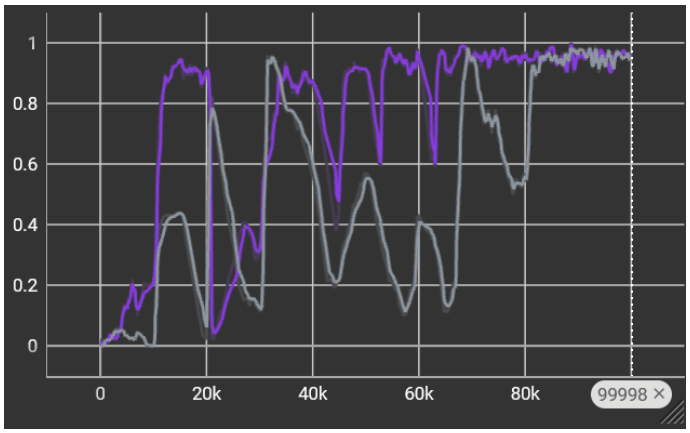

Figure 9: DQN_1 vs. DQN_4

In the Frozen Lake environment, using a larger replay buffer (1,000,000) generally provides more stable training and higher final performance compared to a smaller buffer (100,000). Because Frozen Lake is relatively small and can have sudden failures (like slipping into a hole), a larger buffer retains a wider variety of past experiences, preventing the agent from repeatedly forgetting important states and transitions. so this leads to more intense fluctuations when buffer size is relatively small.

3. Reward Wrapper

It can be seen from (10) and (11) that (step cost = 0) learn faster and achieve higher returns overall, because the agent is not penalized for each action it takes—exploration doesn't incur additional negative rewards. In contrast, (step cost = -1) rise more slowly and plateau at a lower return, reflecting that each step carries a penalty. While the agent with a step cost is motivated to find shorter, more efficient paths, it also becomes more risk-averse and can take longer to discover a good strategy, resulting in a lower final performance compared to the no-penalty case.

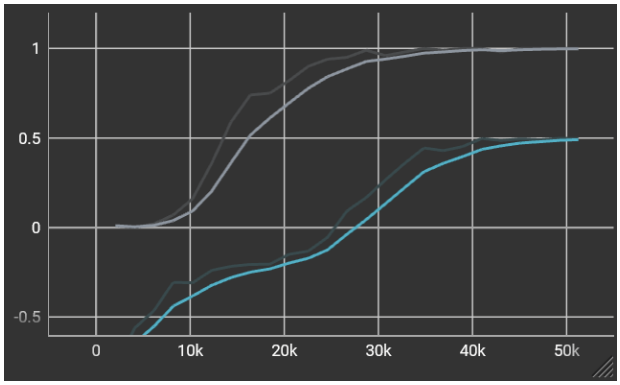

Figure 10: PPO_1 vs. PPO_5

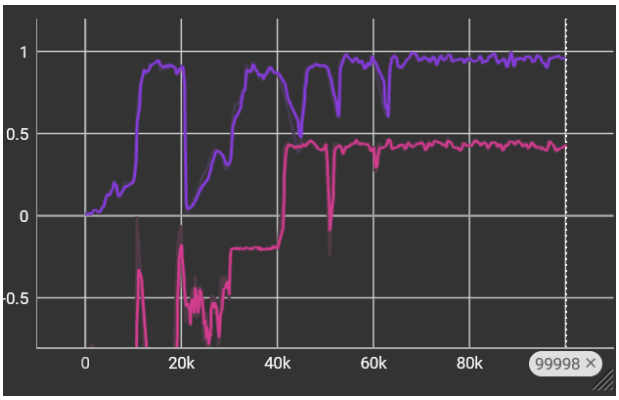

Figure 11: DQN_1 vs. DQN_5

CartPole-v1:

we used CartPole-v1 as our second env. In this part we trained 10 models with hyperparameters as below:

- PPO_1 : "learning_rate": 0.001, "gamma": 0.99, "timesteps": 100000
- PPO_2 : "learning_rate": 0.1, "gamma": 0.99, "timesteps": 100000
- PPO_3 : "learning_rate": 0.001, "gamma": 0.7, "timesteps": 100000
- PPO_4 : "learning_rate": 0.001, "gamma": 0.99, "timesteps": 20000
- PPO_5 (with reward wrapper) : "learning_rate": 0.001, "gamma": 0.99, "timesteps": 100000
- DQN_1 : "learning_rate": 0.001, "gamma": 0.99,"buffer_size"=1000000 , "timesteps": 100000
- DQN_2 : "learning_rate": 0.1, "gamma": 0.99,"buffer_size"=1000000 , "timesteps": 100000
- DQN_3 : "learning_rate": 0.001, "gamma": 0.7 ,"buffer_size"=1000000, "timesteps": 100000
- DQN_4 : "learning_rate": 0.001, "gamma": 0.99 ,"buffer_size"=100000, "timesteps": 100000
- DQN_5 (with reward wrapper) : "learning_rate": 0.001, "gamma": 0.99 ,"buffer_size"=1000000, "timesteps": 100000

Learning Curves

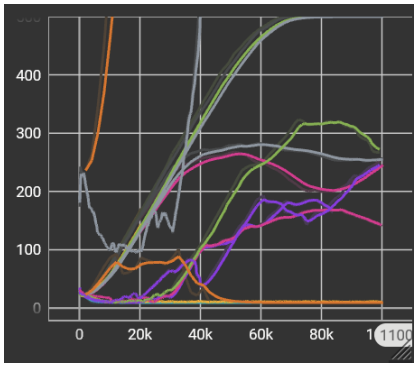

Figure 12: All models reward per episode curves

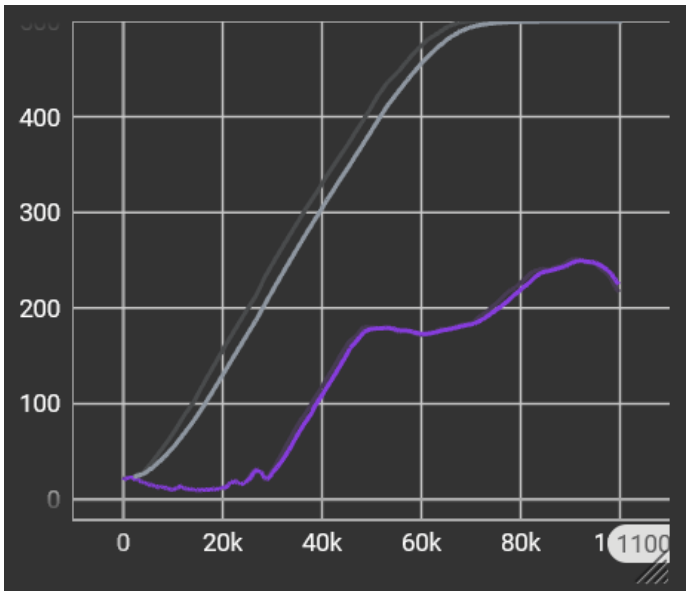

Figure 13: DQN_1 and PPO_1 ep_rew_mean comparison

1. Sample Efficiency and Reward Analysis

It can be seen that CartPole environment contains more complexity than the FrozenLake. So it takes about 70k timesteps for PPO to converge while DQN struggles to converge at even 100k timesteps and does not converge. PPO can be seen to do better on continuous environments.

2. Hyperparameter Comparison

- Learning Rate

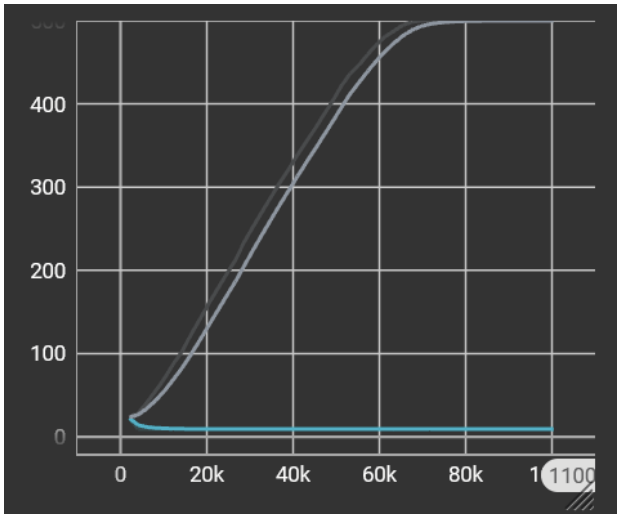

Figure 14: PPO_1 vs. PPO_2

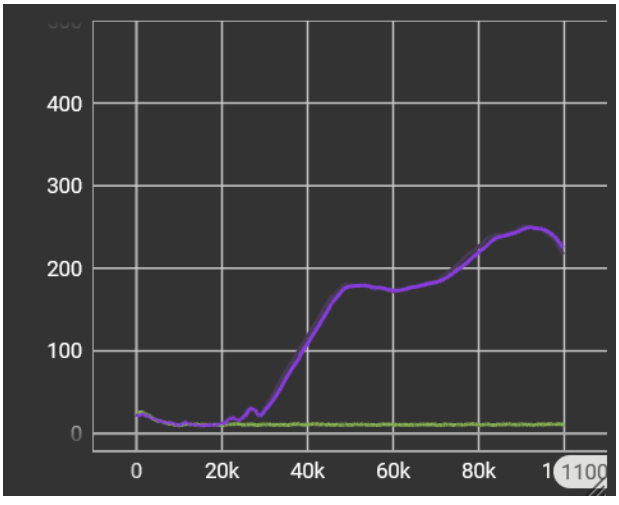

Figure 15: DQN_1 vs. DQN_2

Similar to before, it can be seen that almost no learning has occurred when learning rate is too high.

- γ Parameter

  In this CartPole experiment, the pink curve (lower γ) clearly plateaus at a lower average return, while the gray curves (higher γ) continue to improve and reach higher scores. A lower discount

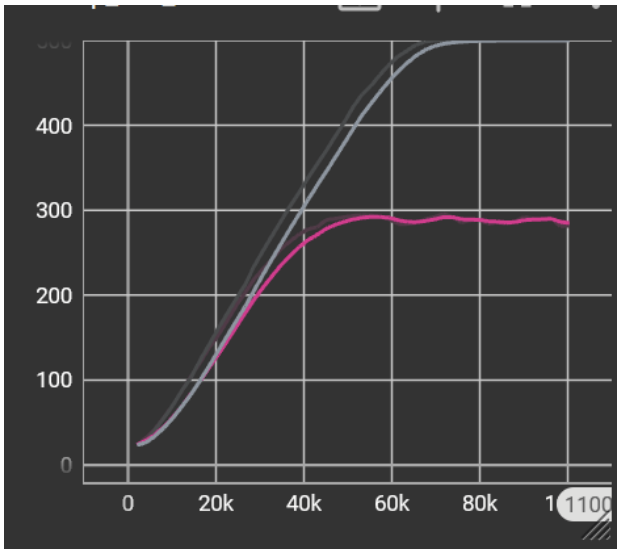

Figure 16: PPO_1 vs. PPO_3

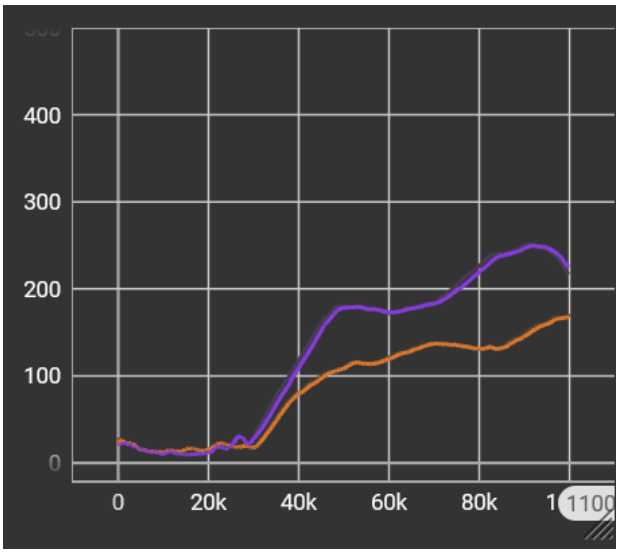

Figure 17: DQN_1 vs. DQN_3

factor causes the agent to focus more on immediate rewards, making it less inclined to learn the longer-horizon balance strategies that yield higher returns. Conversely, the higher discount factor values place more weight on future rewards, encouraging the agent to keep the pole balanced for longer and ultimately achieve better performance.

- Timesteps

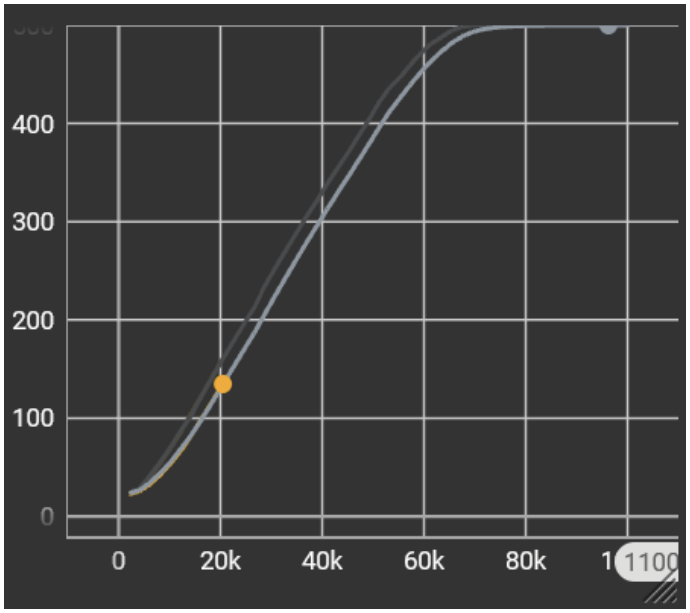

Figure 18: PPO_1 vs. PPO_4

It is obvious that the agent needs enough timesteps to find the optimal policy. here it is evident that 20000 timesteps is not enough for PPO to converge.

- buffer_size

In this CartPole experiment, the agent using a smaller replay buffer (purple line) consistently achieves higher returns and learns faster, whereas the agent with the larger replay buffer (gray line) lags behind and plateaus at lower scores. In a relatively simple environment like CartPole, a very large buffer

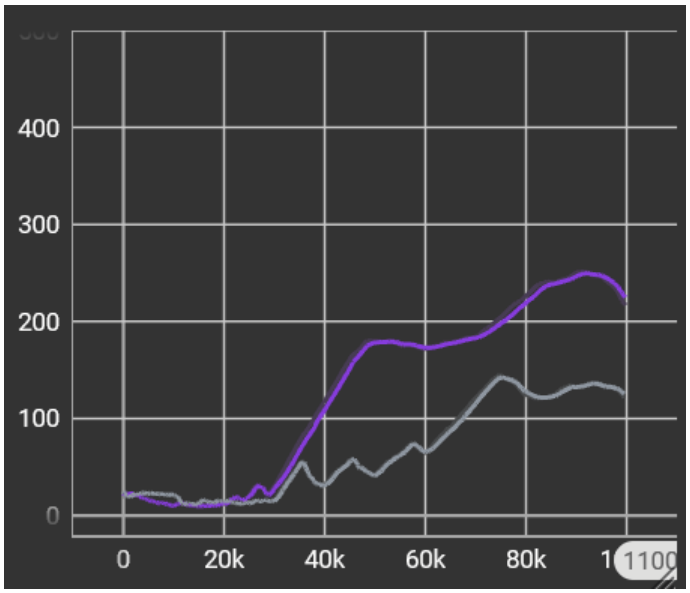

Figure 19: DQN_1 vs. DQN_4

can dilute recent, more relevant experiences with many stale transitions, slowing adaptation and making the agent less responsive to its current behavior. The smaller buffer, on the other hand, focuses training more on recent data—thus helping the agent stabilize the pole more effectively and reach higher performance.

3. Reward Wrapper

It can be seen from (20) and (21) that scaling the reward makes convergence difficult. this could be due to the fact that scaling rewards could act similar as scaling the learning rate which is crucial in convergence. (Just my own thought.)

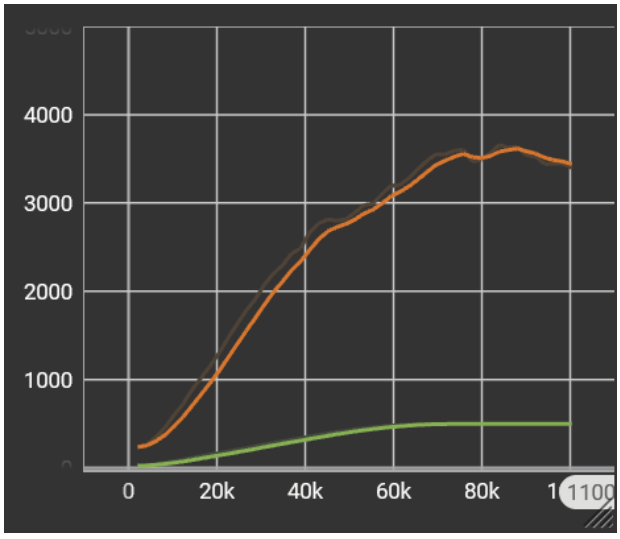

Figure 20: PPO_1 vs. PPO_5

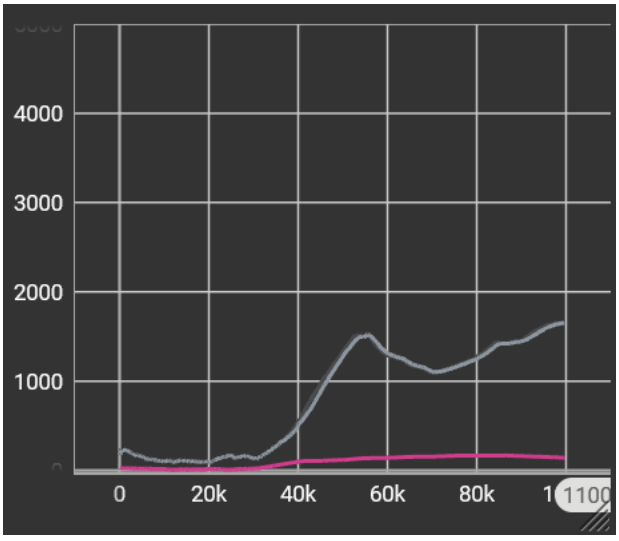

Figure 21: DQN_1 vs. DQN_5

SL vs. RL: Supervised Learning (SL) typically relies on labeled data (i.e., input–output pairs) and aims to minimize a static loss function. However, RL problems fundamentally differ in several ways:

- **No Direct Supervision:** In RL, the agent discovers how to act by interacting with an environment, which provides only a scalar reward signal rather than explicit "correct actions." If we tried to apply

SL, we would need labeled actions for every state, but RL often deals with trial-and-error exploration, where we do not know the "right" action ahead of time.

- **Sequential Decision-Making:** RL deals with policies that maximize cumulative rewards over multiple steps (often an infinite horizon). Actions at one step influence future states and rewards. In SL, each example is typically considered independent. This temporal aspect makes RL's data non-i.i.d. and complicates the straightforward application of SL.
- **Delayed Rewards:** In RL, an action's value might only become clear after many steps (e.g., balancing a pole for a long time, or reaching a goal in a maze). SL does not typically handle such delayed, sparse signals well without specialized training data or additional engineering.
- **Changing Data Distribution:** Because the agent's policy changes over time, the distribution of visited states (and thus training samples) changes. This makes standard SL assumptions—where training and testing data come from the same distribution—less valid.
- **Online Learning Requirement:** RL agents often learn "online" as they interact with the environment. SL methods generally assume a static training set. Constantly retraining from scratch or incrementally updating SL models can be cumbersome if the environment changes or if the agent's behavior shifts over time.

2 Task 2: Creating Custom Environments [45-points]

Environment Definition

In this task, we model a custom 4×4 grid-world as a Markov Decision Process (MDP) with the following components:

- **State Space (S):** All cells in a 4×4 grid, i.e.,

$$S = \{(i, j) \mid 1 \leq i \leq 4,\ 1 \leq j \leq 4\}.$$

- **Action Space (A):** The set of possible actions is defined as

$$A = \{\text{Up}, \text{Down}, \text{Left}, \text{Right}\}.$$

- **Reward Function (R):** The agent receives a reward based on its current state and action. For example, one design can be:

$$R(s, a, s') = \begin{cases} 1, & \text{if } s' \text{ is the designated goal state,} \\ 0, & \text{otherwise.} \end{cases}$$

  Alternatively, a small step penalty (e.g., -0.1) can be introduced to encourage shorter paths.

- **Transition Probability (P):** Since the environment is deterministic, taking an action in a state leads to a unique next state (unless the action would take the agent off the grid, in which case the state remains unchanged):

$$P(s' \mid s, a) = 1 \quad \text{(for the intended transition).}$$

Learning Curves

In this grid-world experiment (22), PPO (gray curve) quickly converges toward higher (less negative) returns and stays relatively stable, whereas DQN (purple curve) shows more volatility and consistently lower returns. One likely reason is that policy-gradient methods like PPO update the policy directly and can more smoothly adjust actions in response to the reward signal, whereas DQN's value-based updates can be more sensitive to replay-buffer composition, learning-rate settings, and exploration noise. Consequently, PPO appears to find a near-optimal strategy faster in this grid-world setup, while DQN struggles with larger fluctuations before partially improving.

Hyperparameters

- Learning Rate

  It can be seen (23),(24) that when the learning rate is too high there are too much fluctuations.

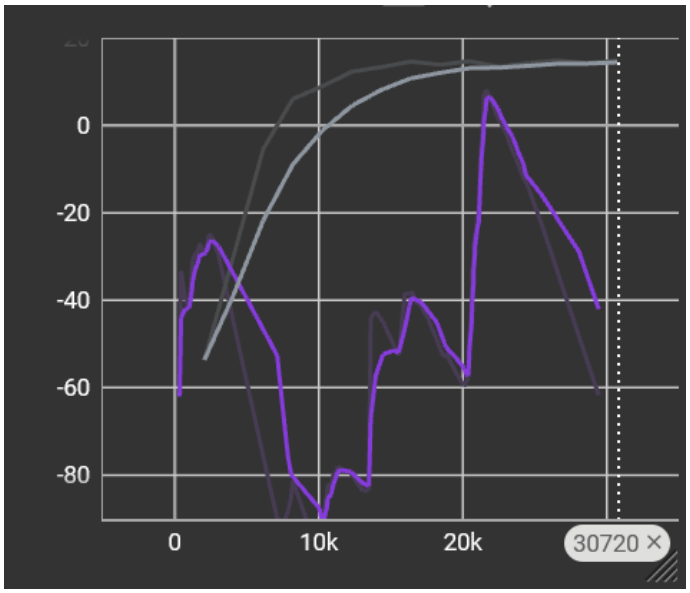

Figure 22: PPO vs. DQN

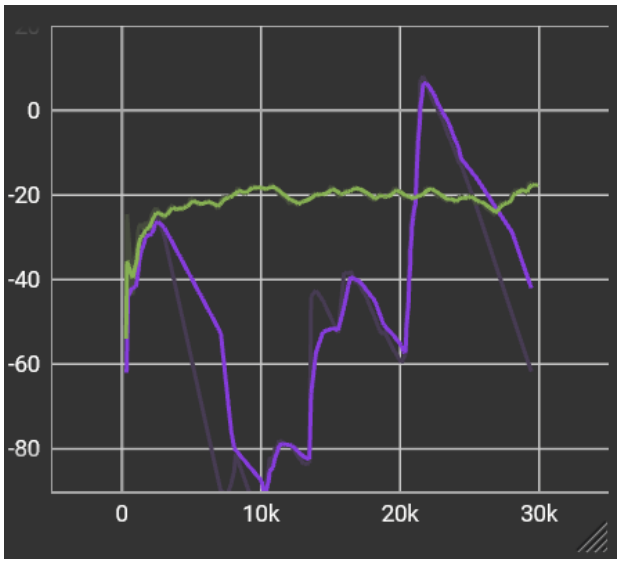

Figure 23: PPO_1 vs. PPO_2

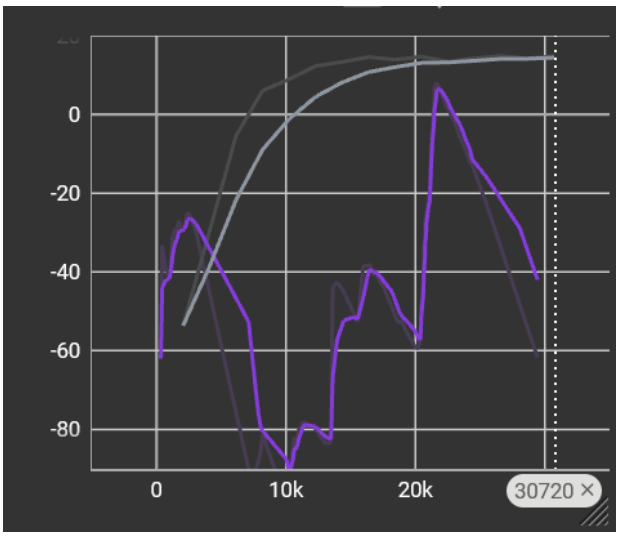

Figure 24: DQN_1 vs. DQN_2

It can be seen that the results is very similar to FrozenLake (since their environments are very similar.). So we avoid repeating the same results for other hyperparameters.

3 Task 3: Pygame for RL environment [20-points]

Chrome-Dino

This report describes a custom Gymnasium environment designed to mimic the Chrome Dino game. The environment uses Pygame for rendering and asset handling. The agent controls a dinosaur that can perform three actions: *no-op*, *jump*, or *duck*. Various obstacles (cacti and birds) scroll from right to left, and the agent must avoid collisions to maximize reward.

State Space:

- A 3-dimensional Box space: `[dino_y, dino_vel_y, closest_dist]`
- `dino_y` $\in [0, 400]$ (vertical position)
- `dino_vel_y` $\in [-20, 20]$ (vertical velocity)
- `closest_dist` $\in [0, 1000]$ (distance to the nearest obstacle)

Action Space:

- A discrete set of 3 actions: `0 = NoOp`, `1 = Jump`, `2 = Duck`

Reward Function:

- +1 for each time step survived.
- +5 for passing an obstacle (obstacle leaves the screen).
- -200 for colliding with an obstacle (episode terminates).

Episode Termination:

- Collision with an obstacle (done $=$ True).
- A step limit (e.g., 10,000 steps) for truncation.

In the PPO training setup, the agent utilizes an `MlpPolicy` to approximate both its policy and value function using a multi-layer perceptron, which is well-suited for handling vectorized observations from the environment. The learning rate is set to `0.001`, providing a balanced step size for gradient updates to ensure stable yet efficient convergence. Finally, the agent is trained for a total of `2,000,000` timesteps, offering ample interaction with the environment to effectively explore and learn an optimal policy over time. **the sample test video `dino_test.mp4` is the folder.**

References

[1] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 2nd Edition, 2020. Available online: http://incompleteideas.net/book/the-book-2nd.html

[2] A. Raffin et al., "Stable Baselines3: Reliable Reinforcement Learning Implementations," GitHub Repository, 2020. Available: https://github.com/DLR-RM/stable-baselines3.

[3] Gymnasium Documentation. Available: https://gymnasium.farama.org/.

[4] Pygame Documentation. Available: https://www.pygame.org/docs/.

[5] CS 285: Deep Reinforcement Learning, UC Berkeley, Pieter Abbeel. Course material available: http://rail.eecs.berkeley.edu/deeprlcourse/.

[6] Cover image designed by freepik