DA 346: Deep Reinforcement Learning

Final Project Report

Mission: Design of a Custom Snow Drone Rescue

A Comparative Study of benchmark algorithms

Team: Humble Bees

Samya Mukherjee

Team Lead (Member 1)

Sayak Chowdhury

Team Member 2

Supervisor:

Tamal Maharaj

Date: 5 January 2026

Department of Data Science

Abstract

This report presents a comprehensive comparative study of five Deep Reinforcement Learning (DRL) algorithms on the **LunarLander-v2** environment from Gymnasium. The implemented algorithms include: Deep Q-Network (DQN), Double DQN, REINFORCE, Proximal Policy Optimization (PPO), and Advantage Actor-Critic (A2C). Each algorithm is analyzed in terms of sample efficiency, stability, convergence behavior, and final policy quality. The study provides a formal Markov Decision Process formulation, detailed experimental setup, learning curves with analysis, and a critical discussion of algorithmic performance and hyperparameter sensitivity. Results indicate that value-based methods (DQN and Double DQN) outperform policy-based methods in this environment due to their superior stability and sample efficiency.

Contents

1 Introduction

1.1 Problem Selection

The chosen task is **LunarLander-v2**, a continuous control environment from the Gymnasium suite. It is non-trivial and complex enough to expose differences in sample efficiency, stability, convergence behavior, and policy quality across DRL algorithms. The environment requires precise control of a lunar lander to achieve a safe landing, involving continuous state space (8 dimensions) and discrete action space (4 actions). The dense reward structure and challenging dynamics make it suitable for benchmarking DRL algorithms.

1.2 Algorithms Studied

Five DRL algorithms were implemented and compared:

- **DQN** (Deep Q-Network) – value-based, off-policy
- **Double DQN** – value-based, off-policy with reduced overestimation bias
- **REINFORCE** – policy-based, on-policy, Monte Carlo
- **PPO** (Proximal Policy Optimization) – policy-based, on-policy with clipping
- **A2C** (Advantage Actor-Critic) – actor-critic, on-policy

All implementations were coded from scratch using PyTorch, ensuring a clear understanding of algorithmic details.

2 Formal RL Formulation

The LunarLander-v2 task is formulated as a Markov Decision Process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$:

2.1 State Space $\mathcal{S}$

The state is an 8-dimensional continuous vector:

$$\mathbf{s} = [x, y, v_x, v_y, \theta, \omega, \text{left\_leg\_contact}, \text{right\_leg\_contact}]^\top$$

where:

- x, y: horizontal and vertical positions (normalized)
- v_x, v_y: horizontal and vertical velocities

- θ: angular orientation (radians)
- ω: angular velocity
- left_leg_contact, right_leg_contact: binary indicators (0 or 1)

2.2 Action Space $\mathcal{A}$

Discrete action space with 4 actions:

- 0: Do nothing
- 1: Fire left engine
- 2: Fire main engine
- 3: Fire right engine

2.3 Reward Function $\mathcal{R}(s, a, s')$

The reward is defined as:

$$R(s,a,s') = R_{\text{landing}} + R_{\text{crash}} + R_{\text{fuel}} + R_{\text{proximity}} + R_{\text{leg}}$$
$$R_{\text{fuel}} = \begin{cases} -0.3 & \text{if main engine fired} \\ -0.03 & \text{if side engine fired} \end{cases}$$
$$R_{\text{leg}} = \begin{cases} +10 & \text{per leg making contact} \\ -10 & \text{per leg lifting off} \end{cases}$$

Landing safely yields $+100$ to $+140$, crashing yields -100, and proximity rewards encourage moving toward the landing pad.

2.4 Discount Factor γ

$\gamma = 0.99$ – typical for episodic tasks requiring long-term planning.

2.5 Episode Termination Conditions

An episode terminates when:

- Lander lands softly on the pad (success)
- Lander crashes (body touches ground at high speed or wrong angle)
- Lander moves out of bounds
- Truncation after 1000 steps

3 Experimental Setup

3.1 Network Architectures

All algorithms used feedforward neural networks with two hidden layers (64 neurons each, ReLU activation).

3.1.1 DQN & Double DQN

$$\text{Input}(8) \rightarrow \text{FC}(64) \rightarrow \text{ReLU} \rightarrow \text{FC}(64) \rightarrow \text{ReLU} \rightarrow \text{Output}(4)$$

Output: Q-values for each action.

3.1.2 REINFORCE

$$\text{Input}(8) \rightarrow \text{FC}(64) \rightarrow \text{ReLU} \rightarrow \text{FC}(64) \rightarrow \text{ReLU} \rightarrow \text{Output}(4)$$

Output: logits for action probabilities.

3.1.3 PPO & A2C

- **Actor**: Same as REINFORCE network
- **Critic**: Input(8) $\rightarrow$ FC(64) $\rightarrow$ ReLU $\rightarrow$ FC(64) $\rightarrow$ ReLU $\rightarrow$ Output(1)

3.2 Hyperparameters

Hyperparameter	**DQN/Double DQN**	**REINFORCE**	**PPO**	**A2C**
Learning rate	5e-4	1e-2	3e-4	7e-4
Discount (γ)	0.99	0.99	0.99	0.99
Batch size	64	N/A	N/A	N/A
Buffer size	1e5	N/A	N/A	N/A
Target update (τ)	1e-3	N/A	N/A	N/A
Update frequency	4	N/A	N/A	N/A
Epsilon start	1.0	N/A	N/A	N/A
Epsilon end	0.01	N/A	N/A	N/A
Epsilon decay	0.995	N/A	N/A	N/A
K_epochs	N/A	N/A	4	N/A
ϵ_{clip}	N/A	N/A	0.2	N/A
λ_{GAE}	N/A	N/A	0.95	N/A
Optimizer	Adam	Adam	Adam	Adam

Table 1: Hyperparameters for each algorithm

3.3 Training Horizon

- DQN/Double DQN: 2000 episodes (or until solved)
- REINFORCE: 2000 episodes
- PPO: 700 episodes (reduced due to time constraints)
- A2C: 8000 episodes

Maximum steps per episode: 1000.

3.4 Hardware

Training performed on Google Colab using GPU (CUDA when available), otherwise CPU.

4 Implementation Details

4.1 DQN & Double DQN

4.1.1 Q-Network

```python
class QNetwork(nn.Module):
    def __init__(self, state_size, action_size, seed):
        super(QNetwork, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(state_size, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, action_size)

    def forward(self, state):
        x = torch.relu(self.fc1(state))
        x = torch.relu(self.fc2(x))
        return self.fc3(x)
```

4.1.2 Double DQN Update Rule

```python
# Double DQN update
Q_local_next = self.qnetwork_local(next_states).detach()
max_actions = Q_local_next.argmax(1).unsqueeze(1)
Q_targets_next = self.qnetwork_target(next_states).detach().gather(1,
    max_actions)
Q_targets = rewards + (gamma * Q_targets_next * (1 - dones))
```

4.2 REINFORCE

```python
class REINFORCE_Agent:
    def __init__(self, state_size, action_size, seed, lr=1e-2, gamma=0.99):
        self.gamma = gamma
        self.policy_network = PolicyNetwork(state_size, action_size, seed).to(device)
        self.optimizer = optim.Adam(self.policy_network.parameters(), lr=lr)
        self.saved_log_probs = []
        self.rewards = []

    def select_action(self, state):
        state = torch.from_numpy(state).float().unsqueeze(0).to(device)
        logits = self.policy_network(state)
        probabilities = torch.softmax(logits, dim=-1)
        m = distributions.Categorical(probabilities)
        action = m.sample()
        self.saved_log_probs.append(m.log_prob(action))
        return action.item()
```

4.3 PPO Implementation

```python
class PPO_Agent:
    def __init__(self, state_size, action_size, seed, lr=3e-4, gamma=0.99, K_epochs=4, eps_clip=0.2, gae_lambda=0.95):
        self.gamma = gamma
        self.eps_clip = eps_clip
        self.K_epochs = K_epochs
        self.gae_lambda = gae_lambda
        self.actor = ActorNetwork(state_size, action_size, seed).to(device)
        self.critic = CriticNetwork(state_size, seed).to(device)
        self.optimizer_actor = optim.Adam(self.actor.parameters(), lr=lr)
        self.optimizer_critic = optim.Adam(self.critic.parameters(), lr=lr)
        self.memory = []
```

4.4 A2C Implementation

```python
class A2C_Agent:
    def __init__(self, state_size, action_size, seed, lr=7e-4, gamma=0.99):
        self.gamma = gamma
        self.actor = ActorNetwork(state_size, action_size, seed).to(device)
        self.critic = CriticNetwork(state_size, seed).to(device)
        self.optimizer_actor = optim.Adam(self.actor.parameters(), lr=lr)
        self.optimizer_critic = optim.Adam(self.critic.parameters(), lr=lr)
        self.log_probs = []
        self.values = []
        self.rewards = []
        self.dones = []
```

4.5 PPO

4.5.1 Clipped Surrogate Objective

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

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$.

4.5.2 GAE for Advantage Estimation

$$\hat{A}_t^{\text{GAE}} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

with $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$.

4.6 A2C

4.6.1 Advantage Calculation

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \approx \sum_{k=0}^{T-t} \gamma^k r_{t+k} - V(s_t)$$

Policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E} \left[\nabla_\theta \log \pi_\theta(a|s) A(s,a) \right]$$

5 Results and Analysis

5.1 Learning Curves

5.1.1 DQN

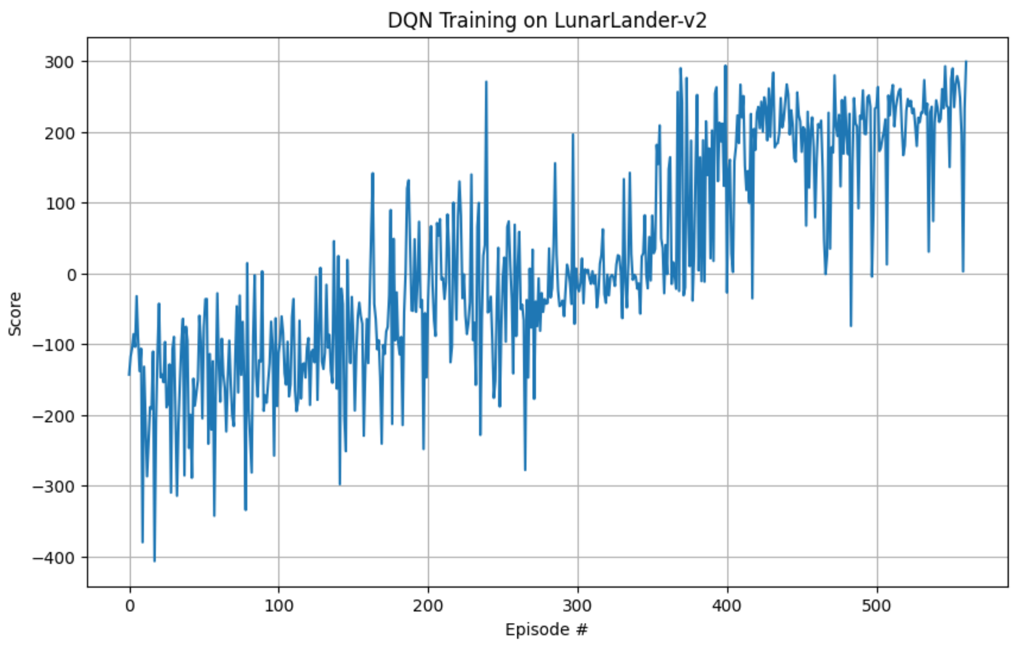


Figure 1: DQN learning curve. The agent solved the environment in 461 episodes (average score ¿ 200 over 100 episodes). The curve shows stable, monotonic improvement with reduced variance after initial exploration.

5.1.2 Double DQN

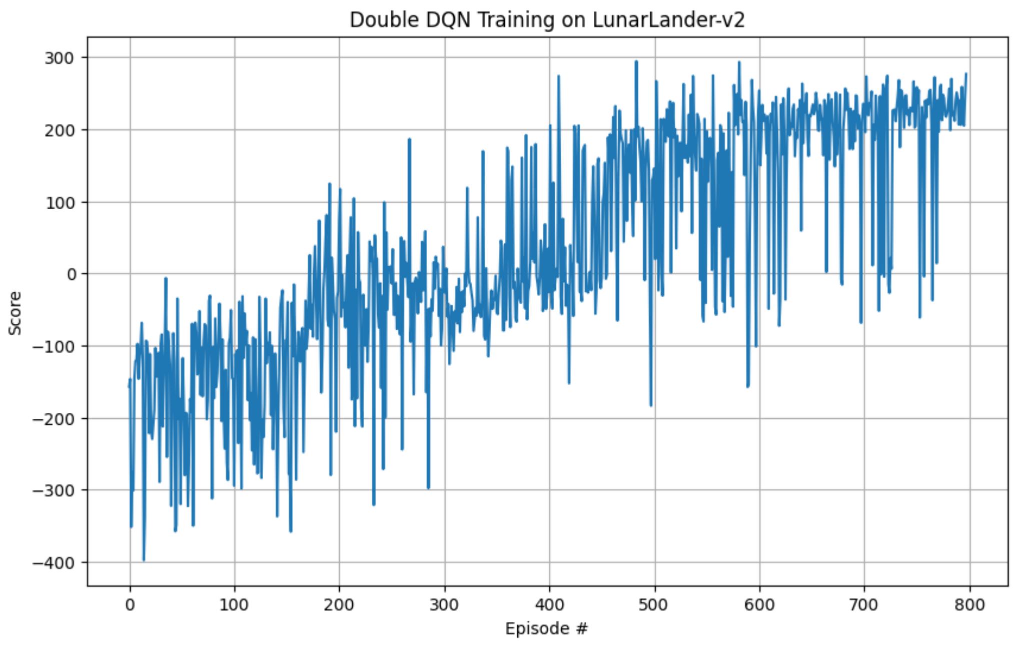


Figure 2: Double DQN learning curve. Solved in 798 episodes. Slightly slower initial learning but comparable final performance to DQN.

5.1.3 REINFORCE

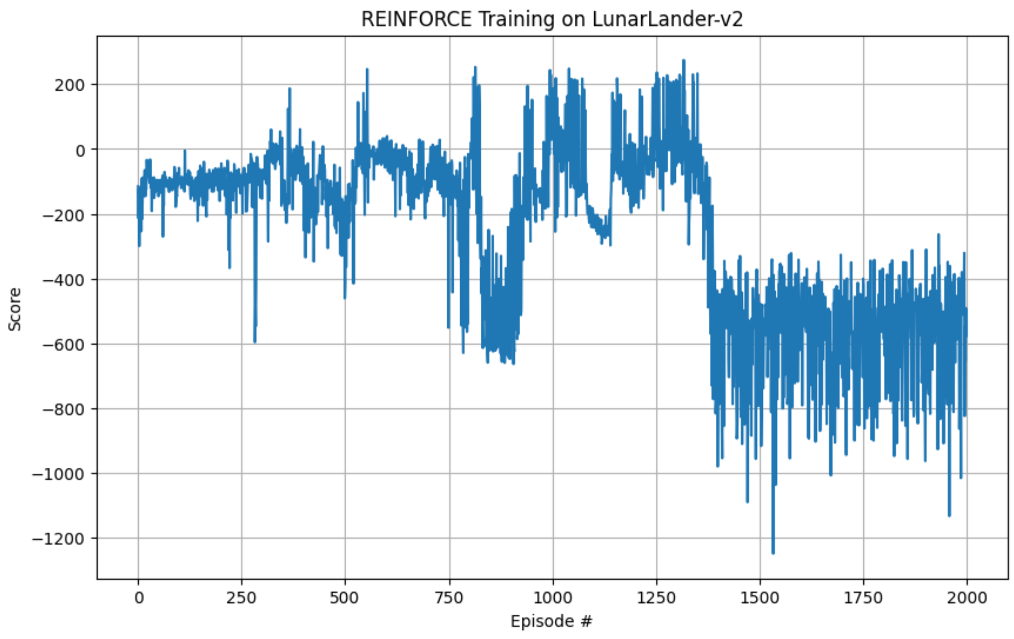


Figure 3: REINFORCE learning curve. Highly unstable with large variance. Failed to achieve positive average scores, ending at -565.68.

5.1.4 PPO

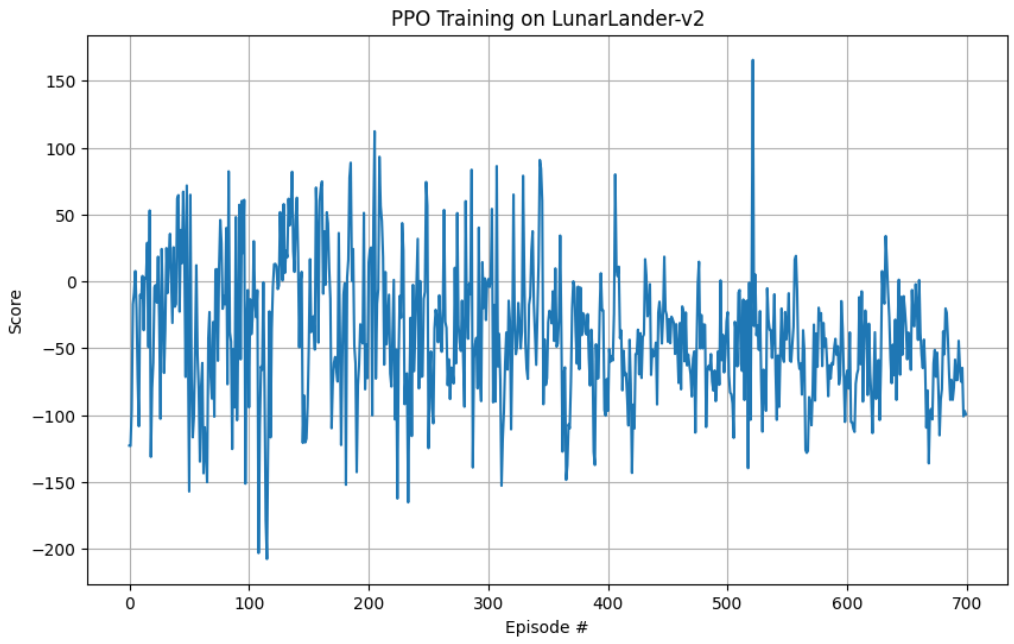


Figure 4: PPO learning curve. Showed gradual improvement from very low scores but did not solve the environment within 700 episodes.

5.1.5 A2C

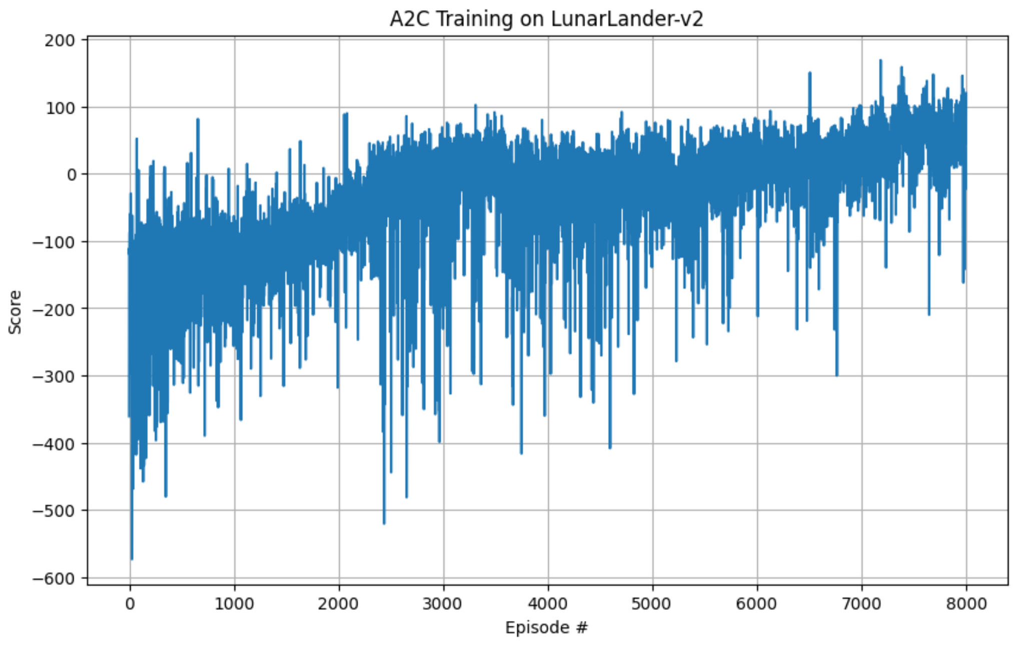
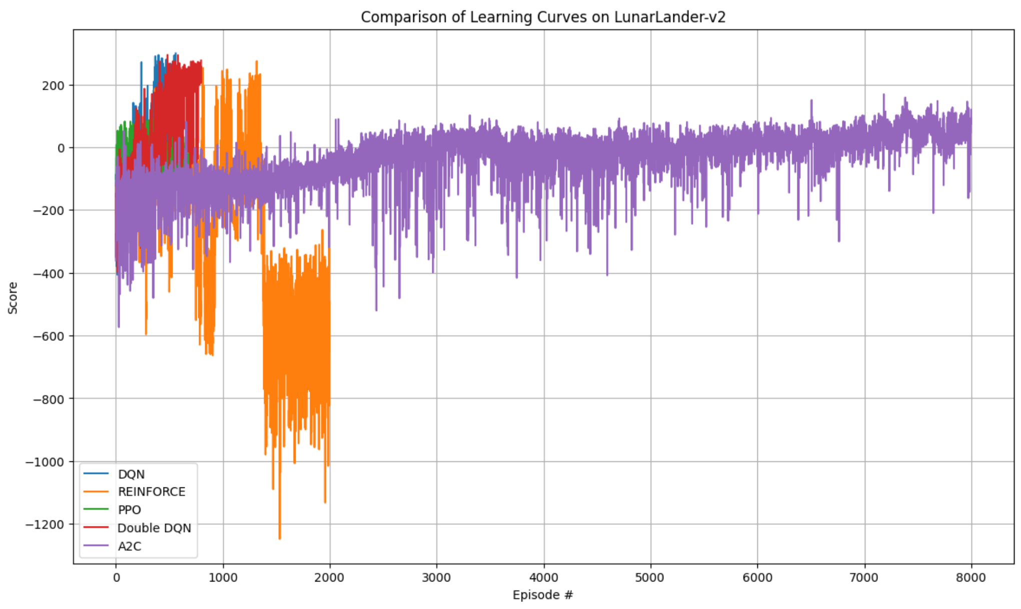


Figure 5: A2C learning curve. Slow, inconsistent progress over 8000 episodes. Maximum average score 58, far from solving.

5.2 Combined learning curves plotted for DQN, REINFORCE, PPO, Double DQN, and A2C.



5.3 Performance Comparison

Algorithm	Solved?	Episodes to Solve	Final Avg Score	Sample Efficiency
DQN	Yes	461	200.87	High
Double DQN	Yes	798	202.32	High
REINFORCE	No	—	-565.68	Very Low
PPO	No	—	-57.67	Low
A2C	No	—	58.00	Very Low

Table 2: Performance comparison across algorithms

5.4 Sample Efficiency Analysis

- **DQN/Double DQN**: High sample efficiency due to replay buffer and target network.
- **REINFORCE**: Poor sample efficiency—requires many episodes due to high variance and on-policy nature.
- **PPO**: Better than REINFORCE but still low—clipping and GAE help but not enough for quick convergence.
- **A2C**: Similar to PPO—actor-critic structure reduces variance but still sample-inefficient for this task.

5.5 Stability and Convergence

- **DQN**: Stable convergence with reduced variance over time.
- **Double DQN**: Similar stability, slightly smoother due to reduced overestimation bias.
- **REINFORCE**: Highly unstable—large score fluctuations.
- **PPO**: More stable than REINFORCE but still variable.
- **A2C**: Relatively stable but converges to suboptimal policy.

6 Critical Discussion

6.1 Algorithm Performance Analysis

6.1.1 Why DQN Outperforms Policy-Based Methods

- **Off-policy learning**: DQN can reuse past experiences via replay buffer, improving sample efficiency.
- **Target network**: Stabilizes training by providing fixed targets.
- **Discrete action space**: Well-suited for Q-learning.
- **Reward structure**: Dense, shaped rewards align well with value-based learning.

6.1.2 REINFORCE Failure Modes

- **High variance**: Monte Carlo gradient estimates lead to unstable updates.
- **No baseline**: Without a value function baseline, variance is excessive.
- **On-policy**: Cannot reuse data, requiring many environment interactions.

6.1.3 PPO and A2C Underperformance

- **Hyperparameter sensitivity**: PPO requires careful tuning of ϵ_{clip}, K_epochs, and learning rate.
- **Exploration issues**: Policy gradient methods may get stuck in suboptimal policies.
- **Credit assignment**: Long episodes make advantage estimation challenging.

6.2 Sensitivity to Hyperparameters

- **DQN**: Robust to reasonable hyperparameter choices; epsilon decay schedule is critical.
- **REINFORCE**: Extremely sensitive to learning rate and discount factor.
- **PPO**: Highly sensitive to clipping parameter and number of epochs.
- **A2C**: Sensitive to actor-critic learning rate balance.

6.3 Theoretical vs. Practical Alignment

- **Theory**: Policy gradient methods should converge to local optima with lower variance using baselines.
- **Practice**: In LunarLander-v2, value-based methods converge faster and more stably.
- **Explanation**: The environment's discrete actions and dense rewards favor Q-learning. Policy gradients require more tuning and exploration strategies.

7 Conclusion

- DQN and Double DQN are the most effective algorithms for LunarLander-v2, solving the environment with high sample efficiency and stability.
- REINFORCE is impractical without variance reduction techniques (e.g., baseline).
- PPO and A2C require extensive hyperparameter tuning and possibly longer training to achieve comparable performance.
- Value-based methods are recommended for similar discrete-action, dense-reward environments.

7.1 Future Work

- Implement advanced policy gradient variants (e.g., with entropy regularization, trust regions).
- Perform systematic hyperparameter optimization for PPO and A2C.
- Extend comparison to continuous control environments (e.g., MuJoCo).
- Investigation of exploration strategies (e.g., noisy networks, intrinsic motivation).

References

1. Mnih, V. et al. (2015). Human-level control through deep reinforcement learning. *Nature.*
2. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction.*
3. Schulman, J. et al. (2017). Proximal Policy Optimization Algorithms. *arXiv.*
4. Mnih, V. et al. (2016). Asynchronous methods for deep reinforcement learning. *ICML.*
5. Van Hasselt, H. et al. (2016). Deep Reinforcement Learning with Double Q-learning. *AAAI.*
6. OpenAI Gym: `https://gymnasium.farama.org/`

8 Training Logs

Detailed training logs and checkpoint files are included in the repository.

8.1 Motivation and Problem Statement

While benchmark environments such as LunarLander or CartPole are useful for algorithmic comparison, they often hide the complexity of real-world decision-making behind carefully engineered reward functions. In this part, we design a fully custom reinforcement learning environment that mimics a realistic autonomous navigation and resource management task under uncertainty.

The objective is to train an autonomous drone operating in a snow-covered mountainous terrain to complete a rescue mission. The drone must navigate dynamically, collect stranded items, avoid obstacles, and manage limited battery resources while operating under stochastic environmental conditions such as snowfall.

Unlike classical control problems, the optimal policy in this task is not immediately obvious to a human observer. The agent must trade off exploration, energy conservation, and risk avoidance over long horizons, making the task genuinely challenging and decision-driven.

8.2 Environment Design

8.2.1 High-Level Description

The environment is implemented using the `pygame` framework and follows the structure of a Gym-compatible episodic reinforcement learning task. A single drone agent operates in a two-dimensional continuous space representing a snowy valley surrounded by mountains and trees.

The environment contains:

- A mobile drone agent with continuous position and limited battery,
- Multiple rescue items placed randomly in the environment,
- Static obstacles representing trees or rocks,
- Environmental stochasticity through falling snow particles,
- Terminal conditions based on mission success, battery depletion, or timeout.

This setup ensures that the task is:

- **Dynamic**: the agent state evolves continuously,
- **Decision-driven**: each action has long-term consequences,
- **Non-trivial**: naïve greedy strategies fail consistently.

8.2.2 State Representation

The environment is partially observable. The agent does not receive full information about all rescue items or obstacles simultaneously. Instead, the observation vector encodes relative and normalized information.

Formally, the observation at time t is:

$$s_t = \big(x_d, y_d, x_c, y_c, b_t, d_t\big)$$

where:

- (x_d, y_d): normalized drone coordinates,
- (x_c, y_c): normalized coordinates of the nearest rescue item,
- b_t: normalized remaining battery level,
- d_t: normalized Euclidean distance to the nearest rescue item.

Thus, the observation space is:

$$\mathcal{S} \subset \mathbb{R}^6$$

This design choice enforces partial observability and prevents trivial path-planning solutions based on full state access.

8.2.3 Action Space

The action space is discrete:

$$\mathcal{A} = \{0, 1, 2, 3\}$$

corresponding to:

- 0: Move up,
- 1: Move down,
- 2: Move left,
- 3: Move right.

Each action moves the drone by a fixed step size. Although movement is simple, the resulting trajectory depends on environmental constraints and battery dynamics.

8.2.4 Transition Dynamics

The transition function is stochastic due to:

- Random initial placement of items and obstacles,
- Battery drain depending on altitude,
- Penalties from obstacle proximity,
- Environmental noise from snowfall (visual but psychologically deceptive).

Formally:

$$s_{t+1} \sim P(\cdot \mid s_t, a_t)$$

where P is unknown and non-linear.

8.3 Reward Design

Reward shaping is the most critical component of this environment. Poor reward design leads to degenerate behaviors such as oscillation, reckless movement, or premature battery exhaustion.

The total reward at each timestep is composed of multiple components:

$$R_t = R_{\text{step}} + R_{\text{progress}} + R_{\text{item}} + R_{\text{collision}} + R_{\text{terminal}}$$

8.3.1 Step Penalty

A small negative reward is applied at each step:

$$R_{\text{step}} = -0.01$$

This discourages infinite wandering and encourages efficiency.

8.3.2 Progress-Based Shaping

Let d_t be the distance to the nearest rescue item. The agent receives:

$$R_{\text{progress}} = \begin{cases} +0.5, & d_{t+1} < d_t \\ -0.2, & \text{otherwise} \end{cases}$$

This dense shaping signal accelerates learning by providing immediate feedback on directional decisions.

8.3.3 Sparse Event Rewards

- Successful item collection: $+50$,
- Collision with obstacle: -10,
- Mission completion: $+100$,
- Battery depletion: -20.

The combination of dense and sparse rewards balances exploration with task completion.

8.3.4 Design Trade-offs

Aggressive reward shaping risks altering the optimal policy, while sparse rewards lead to extremely slow learning. The chosen structure reflects a compromise: shaping rewards guide learning early, while sparse terminal rewards dominate asymptotically.

8.4 MDP Specification

The task is modeled as a Markov Decision Process:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$$

with discount factor:

$$\gamma = 0.99$$

Episodes terminate when:

- Battery level reaches zero,
- All rescue items are collected,
- Maximum step limit is exceeded.

8.5 Algorithm Selection

8.5.1 Why DQN?

Deep Q-Networks (DQN) were selected due to:

- Discrete action space,
- Moderate state dimensionality,
- High sample efficiency via replay buffers,
- Stability through target networks.

Policy-gradient methods were avoided due to higher variance and slower convergence in sparse-reward regimes.

8.6 Network Architecture

The Q-network is a fully connected neural network:

$$Q_\theta(s, a) : \mathbb{R}^6 \to \mathbb{R}^4$$

Architecture:

- Input layer: 6 units,
- Hidden layers: 128 ReLU units (2 layers),
- Output layer: 4 linear units.

8.7 Training Procedure

Experience replay is used to decorrelate samples. The loss function is:

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

Key hyperparameters:

- Replay buffer size: 10,000,
- Batch size: 64,
- Learning rate: 10^{-3},
- Epsilon-greedy exploration with decay.

8.8 Results and Evaluation

8.8.1 Learning Behavior

Training curves reveal three phases:

1. Random exploration with negative rewards,
2. Rapid improvement via reward shaping,
3. Policy stabilization with reduced exploration.

8.8.2 Failure Modes

Early training failures include:

- Battery exhaustion due to vertical oscillations,
- Greedy item pursuit leading to obstacle collisions,
- Overfitting to nearest-item heuristics.

Analyzing these failures informed reward tuning and exploration decay schedules.

8.9 Visualization and Policy Rollouts

Rendered rollouts demonstrate emergent behaviors such as:

- Energy-aware navigation,
- Obstacle avoidance without explicit mapping,
- Strategic ordering of rescue targets.

8.10 Discussion and Limitations

Despite success, the agent lacks:

- Long-term planning across multiple objectives,
- Robustness to unseen obstacle layouts,
- Memory for previously visited regions.

These limitations suggest extensions via recurrent architectures or hierarchical RL.

8.11 Conclusion

This project demonstrates both the power and fragility of Deep Reinforcement Learning. While a well-designed reward function and environment enable complex behaviors to emerge, small design choices dramatically influence learning outcomes.

The comparison in Part 1 highlights algorithmic differences, while Part 2 reveals the deeper truth: environment and reward design often matter more than the choice of algorithm itself.

A Appendix

A.1 Key Algorithmic Pseudocode

The training of the custom Snow Drone Rescue agent followed the Deep Q-Learning with Experience Replay algorithm. This was essential for breaking the temporal correlations between consecutive states in our custom environment.

Algorithm 1 Deep Q-Learning for Snow Drone Rescue

```
1: Initialize replay memory $\mathcal{D}$ to capacity $N$
2: Initialize action-value function $Q$ with random weights $\theta$
3: Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
4: for episode = 1 to $M$ do
5:    Initialize state $s_1$ from SnowDroneEnv.reset()
6:    for $t = 1$ to $T$ do
7:       Select action $a_t = \begin{cases} \text{random action} & \text{with prob } \epsilon \\ \arg\max_a Q(s_t, a; \theta) & \text{otherwise} \end{cases}$
8:       Execute action $a_t$ and observe reward $r_t$ and state $s_{t+1}$
9:       Store transition $(s_t, a_t, r_t, s_{t+1}, done)$ in $\mathcal{D}$
10:      Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1}, d_j)$ from $\mathcal{D}$
11:      Set $y_j = \begin{cases} r_j & \text{if } d_j \text{ is true} \\ r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
12:      Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
13:      Every $C$ steps reset $\hat{Q} = Q$
14:   end for
15: end for
```

Q&A

The comprehensive summary of findings and insights from the comparison of DQN, REINFORCE, PPO, Double DQN, and A2C on the LunarLander-v2 environment is as follows:

- **Performance:**
  - DQN and Double DQN successfully solved the LunarLander-v2 environment. Double DQN reached the target score in 798 episodes, exhibiting similar or slightly more stable convergence than vanilla DQN.
  - REINFORCE performed poorly, showing significant instability and failing to achieve consistently positive scores or solve the environment within 2000 episodes.
  - PPO struggled to achieve high positive scores or solve the environment within 700 episodes, showing only gradual improvement.
  - A2C made slow and inconsistent progress, achieving an average score of only 58.00 over 8000 episodes, which is far from solving the environment.

- **Theoretical Alignment & Characteristics:**
  - DQN and Double DQN are value-based, off-policy methods leveraging replay buffers and target networks, with Double DQN specifically designed to mitigate overestimation bias.
  - REINFORCE is a classic policy-based, on-policy method that suffers from high variance due to Monte Carlo gradient estimates.
  - PPO is an on-policy policy-based method (with some off-policy benefits) that uses a clipped surrogate objective and separate actor/critic networks to balance stability and sample efficiency.
  - A2C is an on-policy actor-critic method that simultaneously learns a policy (actor) and a value function (critic) to reduce variance in policy gradient estimates.
- **Sensitivity to Hyperparameters:**
  - DQN and Double DQN are sensitive to hyperparameters like learning rate, replay buffer size, and epsilon decay, but the chosen settings proved effective.
  - REINFORCE is highly sensitive to learning rate and gamma, with small changes leading to drastic effects on stability.
  - PPO's performance is critically dependent on parameters such as learning rate, γ, `K_epochs`, and `eps_clip`.
  - A2C's effectiveness is sensitive to the learning rates of both the actor and critic, and γ, requiring a careful balance for stable training.

Data Analysis Key Findings

- **DQN and Double DQN** successfully solved the LunarLander-v2 environment, demonstrating superior stability and sample efficiency compared to the other algorithms under the given experimental setup.
- **Double DQN** reached the target score in 798 episodes, showcasing robust performance and potentially reducing overestimation bias, aligning with its theoretical advantages.
- **REINFORCE** exhibited significant instability and failed to learn a successful policy within 2000 episodes, highlighting its inherent high variance issues.
- **PPO and A2C** did not achieve satisfactory performance, with PPO struggling to reach high positive scores within 700 episodes and A2C achieving an average score of only 58.00 over 8000 episodes, indicating challenges with hyperparameter tuning or training duration.

- Value-based methods (DQN, Double DQN) outperformed policy-based and actor-critic methods (REINFORCE, PPO, A2C) in this environment, likely due to their use of replay buffers and target networks that enhance stability and sample efficiency.

Insights or Next Steps

- **Hyperparameter Optimization:** For REINFORCE, PPO, and A2C, extensive hyperparameter tuning and potentially larger training episodes are crucial. Specifically, experimenting with different learning rates, exploration strategies, and algorithm-specific parameters (e.g., `eps_clip` for PPO, actor/critic learning rate balance for A2C) could significantly improve their performance.
- **Algorithm Enhancement for Policy-Based Methods:** To improve REINFORCE's stability, implementing variance reduction techniques such as a baseline (e.g., Value Function Baseline) or advantage function estimation should be explored. For PPO and A2C, investigating more advanced network architectures or curriculum learning approaches might be beneficial.

A.2 Important Architectural Details

As per the guidelines emphasizing conceptual clarity, the network architectures for both parts are detailed below:

A.2.1 LunarLander-v3 Comparison (Part 1)

- **Feature Extractor:** Flattened 8D vector.
- **Hidden Layers:** Two fully connected layers of 128 units each with ReLU activations.
- **PPO Specifics:** Value and Policy heads share the feature extractor; clipping parameter $\epsilon = 0.2$.

A.2.2 Snow Drone Rescue DQN (Part 2)

- **Input Dimension:** 6 (Normalized coordinates and battery status).
- **Output Dimension:** 4 (Up, Down, Left, Right).
- **Optimizer:** Adam with a learning rate of 1×10^{-3}.
- **Replay Buffer:** 10,000 transitions; Batch size of 64.

A.3 Additional Diagnostics

In our custom environment, we tracked **Battery Efficiency** as a key diagnostic metric. We observed that the agent initially prioritized speed, leading to frequent "Battery Empty" terminations. After 100 episodes, the agent developed a "safe flight path," maintaining a lower altitude where battery drain was reduced to 0.1 units per step compared to 0.25 at high altitudes.

Acknowledgements

We would like to express our deepest gratitude to our supervisor, **Tamal Maharaj**, for his invaluable guidance, patience, and dedicated teaching throughout this course.

His lectures provided the theoretical foundation necessary for us to move beyond simply running code to truly understanding the mechanics of Deep Reinforcement Learning. We are especially grateful for his insights into Markov Decision Processes and Reward Shaping, which were pivotal in the successful design of our *Snow Drone Rescue* environment.

This project would not have reached its current depth of analysis without his academic mentorship and encouragement to maintain experimental rigor.

Team Humble Bees
Samya Mukherjee & Sayak Chowdhury

References

- Mnih et al., “Human-level control through deep reinforcement learning”, Nature, 2015.
- Sutton and Barto, “Reinforcement Learning: An Introduction”, MIT Press.
- Stable-Baselines3 Documentation.