Problem Definition

The objective of this project is to develop an agent capable of autonomously landing a spacecraft safely within the *LunarLander-v3* environment, a benchmark task provided by the Gymnasium library. This environment simulates the dynamics of a lunar lander descending onto the moon's surface, requiring precise control to achieve a successful landing.

Environment Overview

The *LunarLander-v3* environment presents a two-dimensional simulation where the agent must control a lander to descend and land between two designated flags on a flat surface. The simulation includes realistic physics, such as gravity and inertia.

State Space

The environment provides an 8-dimensional continuous state vector to the agent at each timestep:

1. **Horizontal Position**: The lander's x-coordinate relative to the landing pad.
2. **Vertical Position**: The lander's y-coordinate relative to the landing pad.
3. **Horizontal Velocity**: The rate of change of the lander's horizontal position.
4. **Vertical Velocity**: The rate of change of the lander's vertical position.
5. **Angle**: The orientation of the lander in radians.
6. **Angular Velocity**: The rate of change of the lander's angle.
7. **Left Leg Contact**: A binary indicator (1 or 0) signifying whether the left leg is in contact with the ground.
8. **Right Leg Contact**: A binary indicator (1 or 0) signifying whether the right leg is in contact with the ground.

Action Space

The agent can choose from four discrete actions at each timestep:

1. **Do Nothing**: No engine is fired.
2. **Fire Left Orientation Engine**: Applies a force to rotate the lander clockwise.
3. **Fire Main Engine**: Applies an upward force to counteract gravity.
4. **Fire Right Orientation Engine**: Applies a force to rotate the lander counter-clockwise.

Transition Dynamics

The environment's transition dynamics are governed by a physics engine that simulates the effects of gravity, engine thrust, and collisions. The lander's motion is affected by:

- **Gravity**: Constant downward force pulling the lander towards the surface.
- **Engine Thrust**: Forces generated by firing the main or orientation engines.
- **Inertia**: The lander's resistance to changes in its motion.
- **Collisions**: Interactions with the ground, which can result in bouncing, sliding, or crashing, depending on the lander's velocity and angle upon contact.

Reward Function
The environment provides a reward signal to guide the agent's learning process:

- **Shaping Reward**: A continuous reward that encourages the lander to move closer to the landing pad, reduce its speed, and align its angle appropriately.
- **Leg Contact Reward**: A +10 reward for each leg that contacts the ground.
- **Crash Penalty**: A -100 reward if the lander crashes or moves out of bounds.
- **Successful Landing Bonus**: A +100 reward for landing within the designated area.
- **Fuel Penalty**: A -0.03 reward for the side engine firing per frame. A -0.3 reward for the main engine firing per frame.

Objective
The primary objective is to train a reinforcement learning agent that can learn an optimal policy to control the lander, maximising cumulative rewards over episodes. The agent must learn to balance the trade-offs between speed, fuel efficiency, and landing precision to achieve successful landings consistently. An episode is considered a solution if it scores >= 200 points.

Background

Reinforcement learning (RL) methods have demonstrated substantial effectiveness in solving complex control problems, including the Moonlander environment as provided by Gymnasium, originally known as LunarLander-v2. Several approaches within RL have shown potential to address this problem, each with distinct strengths and weaknesses.

Q-learning, a model-free value-based reinforcement learning method, has been widely applied due to its simplicity and effectiveness in discrete action spaces (Watkins & Dayan, 1992). It has successfully addressed problems similar to the Moonlander environment because of its straightforward implementation and strong convergence properties. However, traditional Q-learning can struggle with larger state spaces and requires extensive computational resources and tuning to manage its exploration and exploitation balance efficiently (Sutton & Barto, 2018).

Deep Q-Networks (DQN), an extension of Q-learning incorporating deep neural networks for function approximation, provide a compelling alternative capable of handling the large state spaces typical of the Moonlander problem (Mnih et al., 2015). DQNs excel at learning complex mappings between states and optimal actions and have produced strong results in related environments. Nevertheless, DQNs suffer from instability during training and often require careful tuning of hyperparameters and stabilisation techniques like experience replay and target networks (Van Hasselt et al., 2016).

Policy gradient methods, such as REINFORCE (Williams, 1992) and Proximal Policy Optimisation (PPO; Schulman et al., 2017), directly optimise policy parameters and offer advantages in terms of smooth convergence and natural handling of continuous or discrete action spaces. PPO has become particularly popular due to its robust performance and easier tuning compared to other policy gradient algorithms. This method often achieves high performance in LunarLander tasks by effectively handling stochasticity and high-dimensional state spaces. Yet, policy gradient methods typically require substantial

computational power, and the stochastic nature of gradient estimates can introduce high variance, necessitating techniques such as baseline functions to stabilise learning.

Recent approaches employing Actor-Critic algorithms, including Advantage Actor-Critic (A2C; Mnih et al., 2016) and Asynchronous Advantage Actor-Critic (A3C), have combined strengths of both value-based and policy-based methods, balancing stability and computational efficiency. These methods offer stable updates and efficient convergence, making them particularly suitable for real-time control scenarios exemplified by Moonlander. Nevertheless, these algorithms require meticulous tuning to prevent divergence and optimise performance fully.

Studies have explicitly explored LunarLander problems using reinforcement learning. For instance, results available through OpenAI's Gym environment and the research community illustrate PPO's consistent performance, frequently achieving near-optimal solutions (Schulman et al., 2017). Furthermore, research experiments published by Brockman et al. (2016) have highlighted that LunarLander-v2 is solvable with DQN and Actor-Critic methods, demonstrating their applicability and robustness in similar control tasks.

In conclusion, while multiple reinforcement learning approaches are viable, methods such as PPO and DQN currently show the most promise due to their balance between performance, stability, and computational feasibility. Selecting the optimal method should consider the computational resources available and the desired convergence speed and robustness.

Section 3 - Method: A description of the method(s) used to solve your chosen problem, an explanation of how these methods work (in your own words), and an explanation of why you chose these specific methods.

DQN

To solve the Moonlander problem provided by Gymnasium (LunarLander-v3), three reinforcement learning methods were employed: the basic Deep Q-Network (DQN), Double DQN, and Dueling DQN. Each of these methods extends the foundational Q-learning algorithm and introduces specific innovations to enhance performance.

The basic DQN is an adaptation of the Q-learning algorithm, which leverages deep neural networks for approximating the Q-value function. In Q-learning, the goal is to learn a policy that maximises the expected cumulative reward. The Q-value, , represents the expected cumulative reward when taking an action in a state and thereafter following the optimal policy. The DQN algorithm updates its parameters by minimising the loss function:

$$L(\theta) = E_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^{-}) - Q(s, a; \theta)\right)^2\right]$$

where represents the parameters of the current network, represents the parameters of a periodically updated target network, is the discount factor, and indicates uniform sampling from the replay buffer .

Double DQN (Van Hasselt et al., 2016) addresses a key limitation in basic DQN: the overestimation bias arising from using the same network for both action selection and evaluation. Double DQN mitigates this by decoupling these tasks between two networks. It updates its parameters using the modified loss function:

$$L(\theta) = E_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma Q\left(s', \arg\max_{a'} Q(s', a'; \theta), \theta^{-}\right) - Q(s, a; \theta)\right)^2\right]$$

Here, action selection uses the current network , whereas action evaluation utilises the target network . This approach significantly reduces the overestimation bias, leading to more stable and reliable training.

The Dueling DQN architecture (Wang et al., 2016) enhances the learning process by decomposing the Q-value into two separate estimators: the value function , which estimates the expected reward of being in state , and the advantage function , which quantifies the advantage of taking action over other actions in state . The Q-value is represented as:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|A|}\sum_{a'} A(s, a'; \theta, \alpha)\right)$$

This decomposition allows the model to learn state values efficiently, particularly useful when the action choices do not significantly affect the state-value. Consequently, it stabilises training and improves the convergence rate and overall performance.

These three methods—DQN, Double DQN, and Dueling DQN—were chosen for their ability to handle discrete action spaces effectively and their progressive enhancements designed to overcome the limitations inherent in simpler algorithms. Basic DQN provides a solid baseline performance, Double DQN addresses biases associated with standard DQN, and Dueling DQN offers a more nuanced understanding of the decision-making process by separately estimating state and action advantages. Selecting these methods allowed the assessment and leveraging of incremental improvements provided by each subsequent refinement.

For the DQN models we opted for 6 models for comparison. 3 standard models outlined above and then a further 3 models with a tighter learning rate for each of the models.

Proximal Policy Optimisation

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right]$$

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

Proximal Policy Optimisation (PPO), developed by OpenAI in 2017, was a robust reinforcement learning algorithm within the policy gradient family. It was designed to optimise decision-making policies through interactions with an environment, enhancing stability by clipping the surrogate objective function to limit drastic policy shifts. Compared to Trust Region Policy Optimisation (TRPO), which relied on computationally intensive second-order optimisation, PPO employed a simpler first-order approach with a soft constraint, balancing efficiency and performance. Its effectiveness was demonstrated across robotic control, gaming, and natural language processing, rendering it suitable for tasks like the Lunar Lander, a continuous control challenge that required precise spacecraft landing.

Three PPO configurations for LunarLander were evaluated, focusing on their hyperparameters and theoretical implications:

PPO-1 employs a simple setup with a single environment, short rollouts (n_steps = 128), small batch size (32), and few epochs (n_epochs = 4). This configuration uses a constant, high learning rate (0.001) and includes a non-zero entropy coefficient (ent_coef = 0.01) to encourage exploration. While straightforward, these choices contribute to high variance in gradient estimation and slower convergence.

PPO-2 offers a balanced approach by utilizing four parallel environments, moderate rollout lengths (n_steps = 512), a larger batch size (64), and more epochs (n_epochs = 8). Its learning rate starts higher and decreases linearly from 0.0005, combined with tight gradient clipping (max_grad_norm = 0.5) and zero entropy bonus (ent_coef = 0.0). This configuration effectively manages variance and accuracy, balancing computational load and convergence speed.

PPO-3, optimised for high-throughput and stability, uses eight parallel environments and large rollouts (n_steps = 2048), producing extensive and stable gradient estimates. Its configuration includes large batches (64), many epochs (n_epochs = 10), a higher initial learning rate (0.003) that decreases linearly, and a higher clip range (0.3). This setup achieves rapid learning progress and maximises sample efficiency, though it requires significant computational resources.

Results

PPO

Figure 1 (PPO Performance Report, p. 10) illustrates the learning progression of three PPO configurations—PPO-1, PPO-2, and PPO-3—across 3,000 episodes in the Lunar Lander environment. All three configurations ultimately surpass the human-level threshold (reward of 200), though their learning dynamics and stability vary significantly. PPO-3 achieves this threshold rapidly, at around 700 episodes, and maintains a high reward plateau near 280 for the remainder of training. This performance is facilitated by large rollouts (n_steps = 2048) and parallel execution across eight environments, producing stable and low-variance gradient estimates. The larger clip_range (0.3) further enables more substantial early learning steps.

PPO-2 crosses the human-level performance slightly later, just after episode 900, stabilizing within a reward range of approximately 210–230. Its configuration, featuring four parallel environments and moderately sized rollouts (512 steps), strikes a balance between data throughput and variance management. However, the somewhat restricted update budget (n_epochs = 8) mildly limits its peak performance. PPO-1, conversely, takes significantly longer—approximately 1,400 episodes—to reach the performance threshold, eventually plateauing between rewards of 150–180. Its single-environment, small-batch (32 samples) setup, combined with an aggressive fixed learning rate (0.001), contributes to noisy gradient estimates and slower convergence.

The cumulative steps plot highlights PPO-3's superior sample efficiency, enabled by parallel environment processing, which compensates for its more computationally intensive updates. Conversely, PPO-1, due to sequential data collection, significantly lags behind. The cumulative time plot further emphasizes PPO-3's effectiveness; despite heavier computation per update, GPU acceleration mitigates overhead, resulting in only slightly higher total training time compared to PPO-2.

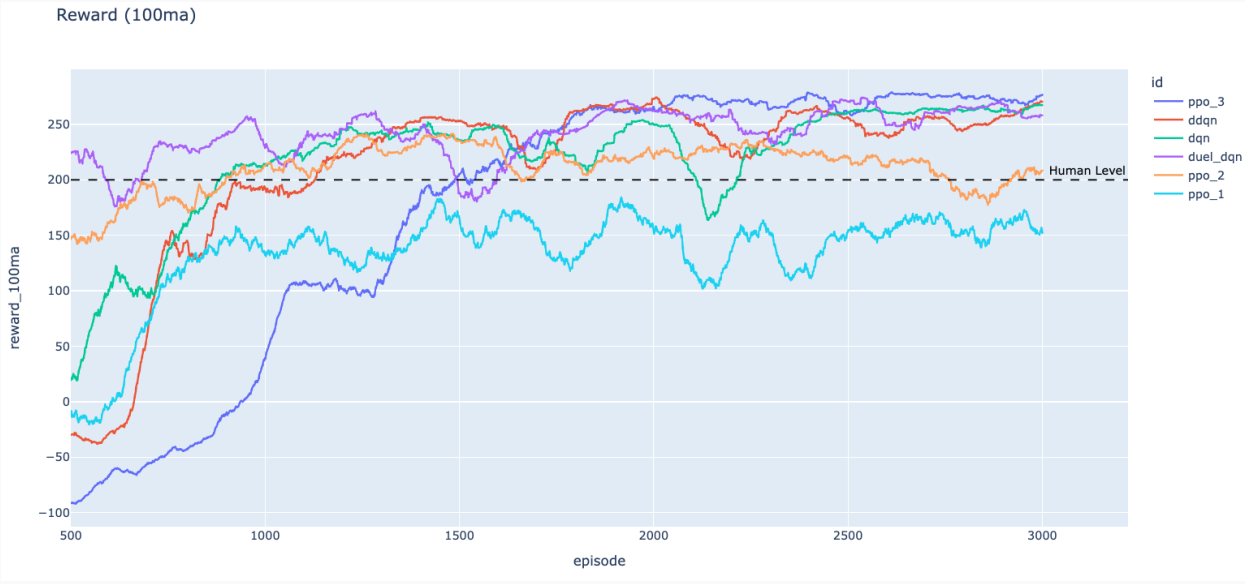
DQN

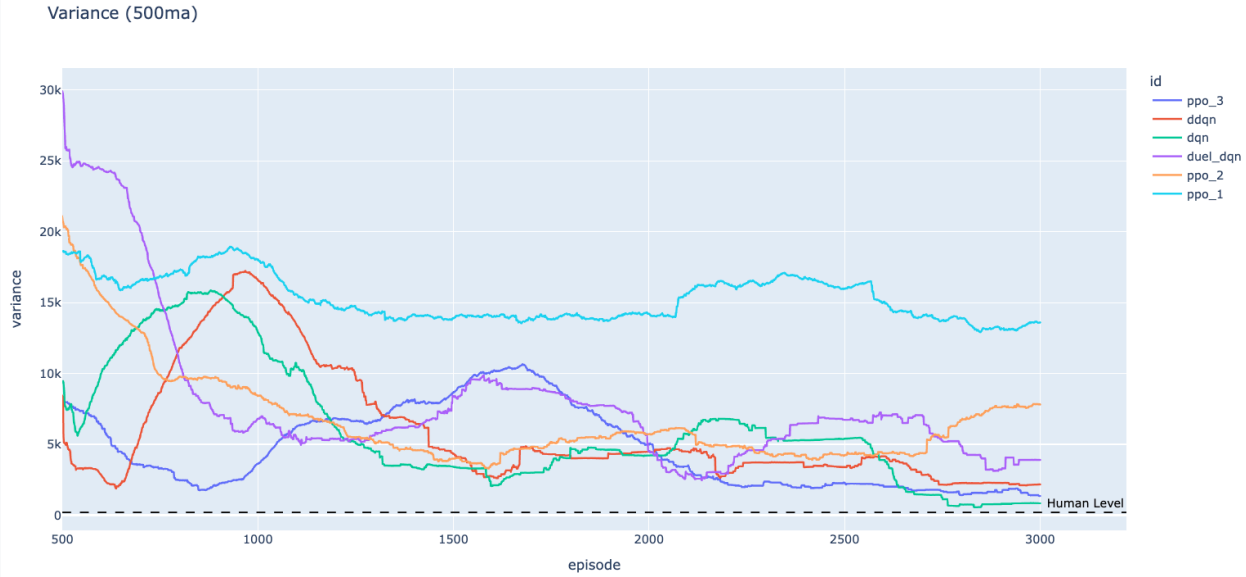
Figure X displays performance comparisons between three Deep Q-learning-based reinforcement learning algorithms: basic Deep Q-Network (DQN), Double DQN (DDQN), and Dueling DQN, over 3,000 episodes in a Moonlander environment. Observing the reward trend (top graph), the Dueling DQN demonstrates notably faster convergence initially, quickly reaching human-level performance (around 200 reward) by approximately episode 400, which suggests it efficiently leverages its architecture to differentiate between the value and advantage functions effectively. Both basic DQN and DDQN algorithms achieve human-level performance later, around episodes 700-800, with DDQN showing slightly more stable progress than basic DQN, highlighting its advantage in reducing overestimation biases.

Interestingly, after achieving human-level performance, all three methods stabilize closely around a similar performance range, suggesting comparable asymptotic performance despite their different learning rates and initial stability. Furthermore, the cumulative steps graph (bottom-left) indicates that DDQN achieves comparable performance with significantly fewer steps compared to the other two algorithms, indicating better sample efficiency. Meanwhile, cumulative training time (bottom-right) suggests no substantial differences between the algorithms regarding computational costs, as all methods demonstrate similar growth in cumulative training time.

Overall, the Dueling DQN offers a clear early advantage in learning speed and stability, the DDQN shows greater sample efficiency, and the basic DQN performs adequately but with slower and less stable initial learning. These observations underscore the practical advantages of incorporating architectural modifications such as dueling networks and double updates in Deep Q-learning methods.

Comparison:





Section 5 -Discussion: An evaluation of how well you solved your chosen problem.

In the first plot (Reward, top), PPO_3 consistently shows strong performance from an early stage, achieving human-level performance (approximately reward 200) quickly and maintaining a superior reward trajectory throughout the episodes. PPO_2 also achieves strong results, surpassing human-level performance early, although it displays slightly less stability compared to PPO_3. PPO_1, on the other hand, struggles to consistently surpass human-level performance, reflecting weaker hyperparameter tuning or less optimal policy learning compared to other PPO implementations.

Among the DQN variants, the Dueling DQN generally performs best, reaching human-level performance earliest, demonstrating initial rapid convergence and sustained stability. The Double DQN (DDQN) shows slower initial learning but stabilises effectively, while the basic DQN displays the most variability, with substantial fluctuations and slower stabilisation.

The second plot (Variance, bottom) complements the reward analysis, demonstrating the stability of each algorithm through variance in rewards. PPO_3 not only achieves higher rewards but also exhibits the lowest variance, indicating superior reliability and consistency in policy performance. PPO_2 initially shows higher variance but subsequently stabilises effectively. PPO_1 consistently displays higher variance, aligning with its poorer reward performance.

Among the DQN algorithms, the Dueling DQN again shows a notably lower variance over time, reinforcing its efficiency in policy stabilisation. Double DQN also reduces variance significantly over episodes, while the basic DQN maintains higher variance throughout, reflecting its difficulty in consistently maintaining stable policy performance.

Overall, PPO_3 and Dueling DQN stand out as optimal solutions, balancing high reward performance with lower variance, demonstrating their robustness and efficiency in solving the Moonlander task. The analyses suggest clear advantages in employing PPO and more sophisticated DQN variations over basic algorithms, especially in environments requiring stable and consistent performance.

The observed performance differentials between the algorithms stem from their structural and functional differences. PPO (particularly PPO_3) achieves superior performance largely because of its robustness in policy optimisation—leveraging policy-gradient methods with clipping techniques to effectively balance exploration and exploitation, stabilising training, and allowing rapid convergence. Its performance is further enhanced by effective hyperparameter tuning, providing better control of learning rates, reward estimation, and exploration strategies.

In contrast, the Deep Q-Network-based methods exhibit variability primarily due to their reliance on value function approximations and sensitivity to hyperparameter tuning. Basic DQN's high variance and instability reflect common issues with overestimating Q-values and inefficient action-value estimations. Double DQN mitigates this somewhat by decoupling action selection from evaluation, reducing estimation bias. Dueling DQN further improves by explicitly decomposing value and advantage, resulting in faster learning and lower variance.

Section 6- Future work: A discussion of potential future work you would complete if you had more time.

To further improve these results, several strategies could be adopted:

- Hyperparameter Tuning: Systematic hyperparameter optimisation (e.g., learning rates, discount factors, and batch sizes) could significantly enhance the performance, especially for PPO_1 and basic DQN.

- Advanced Exploration Strategies: Integrating techniques such as epsilon-decay schedules, entropy regularisation, or curiosity-driven exploration could reduce variance and enhance learning stability.
- Algorithmic Enhancements: Combining Dueling architectures with Double DQN or incorporating Prioritized Experience Replay (PER) could further stabilise and accelerate learning, particularly for value-based methods.
- Regularisation and Normalisation: Techniques like reward normalisation, gradient clipping, or regularisation could reduce training instability and enhance algorithm robustness, notably for PPO variants.
- Network Architecture Optimisation: Experimenting with different neural network architectures, such as deeper networks or advanced activation functions, could lead to better function approximation capabilities.
- Extended Training Duration: Increasing the number of training episodes or steps might allow algorithms more time to converge, especially slower converging methods like basic DQN.

Section 7 - Personal experience: A discussion of your personal experience with the project, such as difficulties or pleasant surprises you encountered while completing it.

Working on the Lunar Lander reinforcement learning task had both challenges and rewarding moments. One of the main difficulties was managing unstable and unpredictable training results, especially with the DQN implementations. Debugging these issues required meticulous logging and patience.

Hyperparameter tuning was another challenge, as finding optimal settings for DQN models was sensitive and time-consuming.

On a positive note, observing PPO's robustness was particularly rewarding. Watching the agent evolve from frequent crashes to executing precise and controlled landings was satisfying, especially with the PPO-3 configuration, which demonstrated rapid convergence and sustained performance.

Implementing and understanding DQN variants like Double DQN and Dueling DQN also provided valuable insights into reinforcement learning techniques.

Overall, despite the difficulties, the process was fulfilling and reinforced our enthusiasm for reinforcement learning.

References:

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. arXiv preprint arXiv:1606.01540.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., ... & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. arXiv preprint arXiv:1602.01783.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimisation algorithms. arXiv preprint arXiv:1707.06347.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (2nd ed.). MIT press.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence, 30(1).

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8(3-4), 229-256.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In Proceedings of the 33rd International Conference on Machine Learning (ICML).

https://gymnasium.farama.org/environments/box2d/lunar_lander/

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347.

Stable-Baselines3 documentation and repository (https://stable-baselines3.readthedocs.io).

Appendices: **If there is additional material or further details that a reader would need in addition to the main body of your report in order to fully replicate your results, you may include them in the Appendices. Appendices may include content such as (1) a more detailed description of your problem domain, including the states, actions, reward function, and transition dynamics; (2) all experimental details so that the reader can fully replicate your experiments; and (3) how you selected your hyperparameters (if applicable).**

Figure 1: PPO

Figure 2: DQN Network


DQN Performance Report
Reward (100ma)
ddqn
dqn
duel_dqn
Human Level
200
100
0
−100
−200
500
1000
1500
2000
2500
3000
Cumulative Steps
1M
0.5M
0
Episode
Cumulative Time
Cumulative Time (s)
800
600
400
200
0
Episode