A2: REINFORCE and basic Actor-Critic Methods

Kirthik Raja Kesavan(s4570170) [1] **Fernando Fejzulla(s4534719)** [1] **Ekansh Khanulia(s4336089)** [1]

Abstract

This report presents a comparative study of four reinforcement learning algorithms applied to the CartPole-v1 environment: Reinforce, a basic Actor-Critic, an Advantage Actor-Critic (A2C),and Proximal Policy Optimization(PPO). We first outline the theoretical foundations and provide concise pseudo-code for each method we applied. Experiments are conducted with consistent policy network architecture, and the performance of each algorithm is evaluated in terms of stability and learning efficiency . Our findings indicate that while REINFORCE can solve the CartPole task, actor-critic methods and PPO offer more stable and smoother convergence.

1. Introduction

Reinforcement Learning (RL) aims to learn policies that maximize total reward through interaction with an environment. The CartPole-v1 environment is a problem where the aim is to balance a pole on a moving cart. An episode terminates if the cart moves too far from the center or the pole falls beyond a certain angle, with a maximum reward of 500 per episode.The cartpole-v1 is considered solved if the agent is able to achieve an average reward above 475 or above over last 100 consecutive episodes. In this assignment, we implement and compare four Reinforcement Learning algorithms:

- **REINFORCE** — a Monte Carlo policy gradient algorithm.
- **Actor-Critic (AC)** — which uses value function baseline to reduce variance .
- **Advantage Actor-Critic (A2C)** — which is an AC algorithm using the advantage function to guide updates.
- **Proximal Policy Optimization(PPO)** — which is an advanced policy gradient algorithm that to ensure stable , small policy updates using a clipped surrogate objective, leading to efficient and reliable learning.

We compare these Algorithms against a baseline (dqn) from a previous assignment and discuss their relative strengths and weaknesses.

2. Theory

2.1. Vanilla Policy Loss and REINFORCE

The origin of the vanilla policy loss in the REINFORCE algorithm stems from the reason to maximize the expected return and that is done by directly optimizing the policy parameters. The aim in policy-based approaches, where the policy function, π is parameterized, is to adjust these parameters so we can encourage trajectories yielding high rewards.In practice,we achieve this by taking initial state value ,$V(s_0)$, into consideration , as a measure of the policy quality, which is maximized by the algorithm via gradient ascent.

Since the optimal action is unknown, estimates of action values and sampled trajectories are used instead.So naive update rule might look like:

$$\theta_{t+1} = \theta_t + \alpha\, Q(s,a)\, \nabla_\theta \pi(a \mid s).$$

However, this can over-reinforce actions with high $Q(s,a)$ and high probability. To lessen it, we normalize by $\pi(a \mid s)$ using the identity

$$\nabla_\theta \log \pi(a \mid s) = \frac{\nabla_\theta \pi(a \mid s)}{\pi(a \mid s)},$$

which gives the update as follows:

$$\theta_{t+1} = \theta_t + \alpha\, Q(s,a)\, \nabla_\theta \log \pi(a \mid s).$$

In REINFORCE ,the discounted total reward R from an episode is an estimate for $Q(s,a)$. Thus, the vanilla policy loss is implicitly defined by the gradient ascent update rule:

$$\Delta\theta \propto Q(s,a)\, \nabla_\theta \log \pi(a \mid s),$$

[1]Faculty of Science, Universiteit Leiden, Leiden, Netherlands. Correspondence to: Your Name <your.email@domain.com>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

which aims for boosting actions leading to higher returns by modifying the parameters.

REINFORCE :

Algorithm 1 Reinforce Algorithm (Pseudocode)

1: Initializing policy network π_θ with parameters θ
2: **for** each episode **do**
3: Generating trajectory (s_t, a_t, r_t) using policy π_θ
4: Computing returns R_t (Monte Carlo estimates of $Q(s_t, a_t)$)
5: Computing policy loss: $L(\theta) = -\sum_t \log \pi_\theta(a_t|s_t) R_t$ (Eq. 2)
6: Use gradient ascent to update policy parameters: $\theta \leftarrow \theta + \alpha \nabla_\theta L(\theta)$
7: **end for**
8: Repeat until convergence

2.2. Actor-Critic Methods

The term "actor" and "critic" come from their roles in reinforcement learning. The "actor" chooses action and "critic" evaluates how good the action is . We implemented Actor-Critic algorithm using two neural networks:

- **Actor (Policy Network):** A neural network $\pi_\theta(s)$ that computes a probability distribution over actions given a state. It trains a stochastically chosen actions using the policy gradient method.
- **Critic (Value Network):** A neural network $V_\phi(s)$ that estimates the value of a given state. This is often used as a baseline to calculate advantage, but in our implementation we used R_t directly for value and policy updates

Why Not Use Q-Learning Loss?

Q-learning loss includes a maximum over possible actions and is based on the Bellman optimality equation :

$$L_{\text{Q-learn}} = \Big(r + \gamma \max_a Q(s', a) - Q(s, a)\Big)^2$$

However, Q-learning assumes a **deterministic policy** and estimates the action-value function $Q(s, a)$. In contrast, our approach aligns better with policy gradient methods because it uses a policy of **stochastic** nature which approximates the state-value function $V(s)$.

What Loss is Used Instead?

We utilizes Monte Carlo to compute the return R_t from rewards and use it for both updates:

- **Policy Loss (Actor):**

$$L_{\text{policy}} = -\log \pi_\theta(a_t|s_t) \cdot R_t$$

- **Value Loss (Critic):**

$$L_{\text{value}} = \text{MSE}(V_\phi(s_t), R_t)$$

Algorithm 2 Basic Actor-Critic Algorithm (Pseudocode)

1: Initialize actor π_θ and critic V_ϕ
2: **for** each episode **do**
3: Generate trajectory (s_t, a_t, r_t) using π_θ
4: Computing returns $R_t = \sum_{l=0}^{T-t} \gamma^l r_{t+l}$
5: Normalize R_t to have zero mean and unit variance
6: Computing policy loss: $L_{\text{policy}} = -\sum_t \log \pi_\theta(a_t|s_t) \cdot R_t$ to train actor
7: Computing value loss: $L_{\text{value}} = \sum_t (V_\phi(s_t) - R_t)^2$ to train the critic
8: Update θ, ϕ via gradient descent on both value and policy losses.
9: **end for**

2.3. Advantages over Q-values

Using the advantage function, $A(s, a) = Q(s, a) - V(s)$, instead of using raw Q-values in policy gradient methods provides several benefits:

- **Variance Reduction:** Subtracting $V(s)$ from $Q(s, a)$ reduces gradient variance which stabilizes learning without biasing updates,
- **Improved Policy Evaluation:** The advantage function provides a clearer signal because it highlights actions better than the state average.
- **Relative Action Quality:** Advantage reinforces actions out performing $V(s)$, focusing learning on those that improve performance.

2.4. Advantage Actor-Critic (A2C)

The advantage function is defined as:

$$A(s, a) = R_t - V(s)$$

where R_t is the Monte Carlo return and $V(s)$ is the critic's estimate of the state value.

Loss Functions:

- **Policy Loss (Actor):**

$$L_{\text{policy}} = -\log \pi_\theta(a|s) \cdot A(s, a)$$

- **Value Loss (Critic):**

$$L_{\text{value}} = \text{MSE}(V(s), R_t)$$

Algorithm 3 Advantage Actor-Critic (A2C)(Pseudocode)

Initialize actor network parameters θ and critic network parameters ϕ.
for each episode **do**
  Reset environment and initialize lists for rewards, log-probs, and values.
  Collect trajectory (s_t, a_t, r_t) using policy π_θ and store $V_\phi(s_t)$.
  Compute discounted returns R_t.
  Compute advantages $A_t = R_t - V_\phi(s_t)$.
  Update actor: $\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) A_t$.
  Update critic with MSE: $\phi \leftarrow \phi - \beta \sum_t \nabla_\phi (V_\phi(s_t) - R_t)^2$.
end for

3. Experiments

3.1. Experimental Setup

We use the CartPole-v1 environment from Gymnasium, where the agent balances a pole using left or right actions. It receives +1 reward per step, up to 500 steps. The task is solved when the average reward exceeds 475 over 100 consecutive episodes. To ensure consistency and reproducibility, we fix random seeds and run each algorithm 5 times for 1 million steps, logging average rewards for performance tracking.We use the same policy network across all algorithms. It maps the 4-dimensional state to a probability distribution over two actions using two hidden layers (64 units, ReLU) and a softmax output, enabling stochastic action sampling for effective exploration.For smoothing the returns, we collect the average episode rewards at a fix step intervals across all 5 runs of an algorithm. These rewards are stored as lists of (step,avgreward) pairs. For each step interval, we compute the mean reward across the 5 runs, producing a single averaged value per step this help us to capture the learning curve. We consider the dqn (Figure 1)as our baseline and compared it with our results.

In our baseline dqn (Figure 1) we used a learning rate of 0.0005 and discount factor $\gamma = 0.99$ and we maintain same values throughout our algorithms to maintain fair and consistent comparison across different learning paradigms.The discount factor of 0.99 is a standard choice in CartPole that balances responsiveness immediate feedback to long-term rewards , which is necessary in an episodic setting with a 500-step cap. Similarly, the learning rate of 0.0005 is a commonly used starting point that ensures reasonably stable convergence without overwhelming the policy or value network updates. Keeping these values consistent allows us to isolate the effects of algorithmic differences.Keeping the learning rate and discount factor unchanged for Reinforce, AC and A2C resulted in effective and stable learning curve.

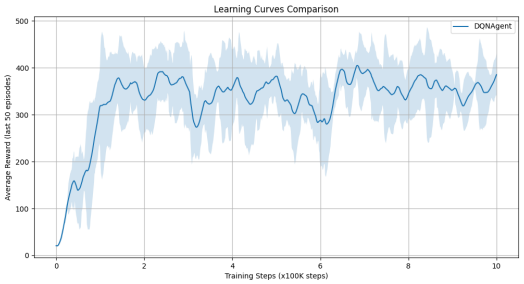

Figure 1. Learning curve for the DQN-based baseline agent in the CartPole over 1 million steps, plotting the average reward (last 50 episodes)

3.2. Approach and Results

- **Reinforce** algorithm is build upon a **single policy network π_θ** which , without using a separate value function, directly optimizes the agent's actions . At every episode, an agent collects a trajectory (s_t, a_t, r_t) until its termination and calculate discounted returns $\mathbf{R}_t = \sum_{k=t}^{T} \boldsymbol{\gamma}^{(k-t)} \mathbf{r}_k$, which approximate to $\boldsymbol{Q}(s_t, a_t)$.These returns are normalized to have unit variance within every episode and zero mean ,so that we have stabilize training throughout. Then we update the agent using the policy gradient principle, formalized as $\nabla_\theta J(\theta) \propto \sum_t R_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)$.Simultaneously we can view it as minimizing the loss $L_{\text{policy}} = -\sum_t \log \pi_\theta(a_t \mid s_t)\, R_t$.The algorithm shifts its policy towards rewarding action by increasing the logarithmic-probability of actions that gives higher returns . REINFORCE relies heavily on Monte Carlo returns,as **no value network is used**.Which results in the learning process experiencing larger fluctuation (figure 2) in gradient updates than Actor-Critic approaches but having a straightforward implementation.

- **Actor-critic** (AC) method uses two separate networks which include the actor (policy) and the critic (value function). The actor network uses $\pi_\theta(a_t|s_t)$ to choose an action a_t from the current state s_t at each time step and the critic network produces an estimate of $V_\phi(s_t)$. Both networks receive updates from the AC algorithm following the completion of every episode. There algorithm calculates returns R_t through the summation of $\gamma^{,k-t} r_k$ values from time step t until the end of the episode. The actor receives updates through policy gradient optimization using R_t as an approx-

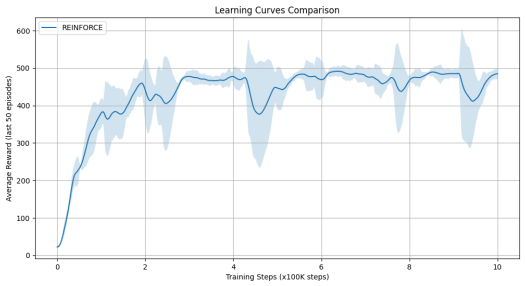

Figure 2. Reinforce learning curve on the CartPole environment over 1 million steps, plotting the average reward (last 50 episodes).

imation of $Q(s_t, a_t)$ while the critic learns to predict $V_\phi(s_t)$ based on R_t as target signals. Our implementation uses policy gradient maximization of $\log \pi_\theta(a_t|s_t) \cdot R_t$ which is equivalent to minimizing the loss function $L_{\text{policy}} = -\log \pi_\theta(a_t|s_t) \cdot R_t$. The critic network aims to minimize the mean squared error $L_{\text{value}} = \big(V_\phi(s_t) - R_t\big)^2$. Normalization of returns R_t to have zero mean and unit variance during each episode improves stability before using them for actor updates. The normalization process functions as a reward subtraction through mean based centering witch decreases the variance of the policy gradient. The learned critic V_ϕ functions as a baseline but we omitted its subtraction from R_t in this fundamental AC updates for the actor's loss.

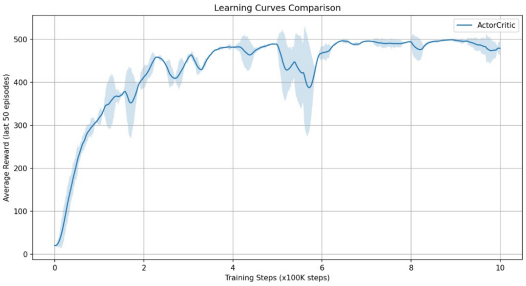

Figure 3. Learning curve for the Actor-Critic algorithm in the CartPole environment over 1 million steps, showing average reward (last 50 episodes).

- **Advantage Actor-critic** (A2C) is build upon the basic Actor critic by explicitly using the advantage function to update the policy. Instead of using raw return R_t as the feedback signals for the actor, A2C computes the advantage $A_t = R - t - V_\phi(s_t)$. This advantage represents the excess return beyond the critic's predicted value for state s_t, and it centers the policy gradient updates around zero. Intuitively, $A_t > 0$ means the action was better than expected, and $A_t < 0$ means it was worse. The actor's loss is then $L_{\text{policy}} = -\log \pi_\theta(a_t|s_t), A_t$, which pushes the policy factor to favor actions with positive advantage and void those with negative advantage. The critic is updated similarly to AC, minimizing $L_{\text{value}} = \big(V_\phi(s_t) - R_t\big)^2$ to fit the returns. In our implementation, after each episode we compute A_t for every time step using the collected R_t and the current value estimates $V_\phi(s_t)$ from the critic. We then update the actor network with gradient $\nabla_\theta \log \pi_\theta(a_t|s_t), A_t$ and the critic with $\nabla_\phi \big(V_\phi(s_t) - R_t\big)^2$. Unlike the basic AC, we did not normalize the returns A2C - the advantage term itself achieves a form of normalization by subtracting the baseline $V_\phi(s_t)$ from the return. This subtraction greatly reduces the variance of the policy gradient without introducing bias, giving A2C a more stable update signal than AC's raw returns.

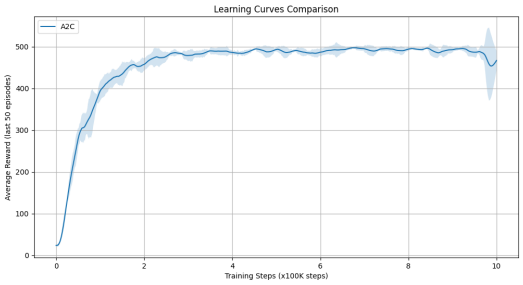

Figure 4. A2C Learning curve on CartPole environment over 1 million steps, showing the average reward (last 50 episodes)

- **Proximal Policy Optimization (Bonus)** PPO algorithm is build upon two networks: one for policy (actor) and one for value (critic). This setup helps for making stable updates in reinforcement learning. The policy network gives probability over actions, so actions are sampled randomly, not always same. The value network predicts how good a state is, which is used later for computing advantage.Training starts by collecting full episodes from environment. After that, returns are calculated with discounting, and advantages are taken by subtracting predicted values from returns. These advantage values are used to update the policy, using a clipped surrogate objective. The ratio $r_t(\theta)$ is clipped between $[1-\epsilon, 1+\epsilon]$, so the update not go too far.Policy loss is taken as minimum between clipped and non-clipped objective to avoid big update steps. Value loss is computed as mean squared error between predicted values and actual returns. Optimization is done with Adam optimizer, and different learning rates are used for policy and value networks.During training,

rewards are logged sometimes to monitor how well the model is doing. Also, PPO allows doing several epochs of update on same data batch. This approach makes PPO both efficient with data and stable for learning, and it works well in environments like CartPole-v1.

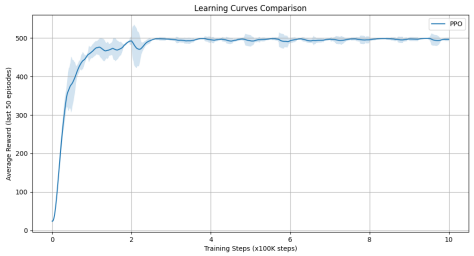

Figure 5. PPO learning curve on CartPole-v1 over 1 Million steps, showing average reward (last 50 episodes)

4. Discussion

4.1. REINFORCE vs Actor-Critic (AC)

The basic actor-critic algorithm dramatically improves learning speed and stability relatively to REINFORCE baseline. By using a learned value function as a baseline, the AC agent reduces the variance of policy gradient estimates without shifting the expected gradient. In practice, this led to much faster convergence on Cartpole. Whereas REINFORCE agent might ”take an order” of several hundred episodes to approach the solve threshold - and often exhibited unstable oscillations in reward - the AC agent reached high average rewards more consistently. For instance, one analysis is showed that AC achieved the goal nearly 400 episodes sooner than REINFORCE and maintained mean reward thereafter. The inclusion of hte critic thus provides a significant sample-efficiency gain. However, the AC algorithm still uses Monte Carlo returns, so some variances remains (we observed occasional dips in the AC learning curve), and the two-network training adds a bit of complexity. Overall, though, the actor-critic clearly demonstrates superior performance on this task, validating the benefit of baseline subtraction in policy gradient methods.

4.2. Advantage Actor-Critic (A2C) vs Actor-Critic (AC)

Both Actor-Critic (AC) and Advantage Actor-Critic (A2C) methods use two separate networks – one for the policy (actor) and one for the value estimate (critic). In the AC algorithm, the agent use Monte Carlo returns directly to compute the policy loss. This gives a useful learning signal, but because it doesn't subtract baseline, the variance stays high. In training, this was seen as more noisy reward curves and slower stability. AC agent can still learn, but often reward fluctuates a lot, and sometimes step back after progress. A2C, on other hand, adds advantage estimation: it subtracts the state value $V(s)$ from return R, to get advantage $A(s, a) = R - V(s)$. This helps the policy gradient focus more on the action's real benefit compared to average. In our training runs, A2C agent showed faster convergence, with smoother reward trend and less noisy updates. For example, after same training steps (200k), A2C usually reached higher average reward and stayed more stable compared to AC. In summary, A2C can be seen as improved version of standard AC. The use of advantage instead of full returns reduced variance and led to more stable learning. On tasks like CartPole, A2C reached high reward faster and maintained it more reliably, while AC needed more episodes and was more sensitive to randomness in training. So for most practical cases, A2C is preferred if goal is stable and sample-efficient learning.

4.3. A2C vs PPO

Both A2C (Advantage Actor-Critic) and PPO (Proximal Policy Optimization) build on the actor-critic framework, using separate networks for policy and value estimation. A2C updates the policy directly using advantage estimates from a single trajectory, making it simpler but potentially unstable, as it lacks mechanisms to constrain policy updates. PPO addresses this by introducing a clipped surrogate loss, which restricts how much the policy can change during an update, along with multiple optimization epochs over the same data. This improves training stability and sample efficiency. In the CartPole-v1 environment, PPO significantly outperformed A2C in both learning speed and final reward stability. PPO quickly converged toward the optimal reward of 500 and maintained it with minimal variance, while A2C showed more fluctuation and slower improvement. The results indicate that while A2C is lightweight and easier to implement, PPO's more conservative update rule yields better overall performance and robustness in environments requiring precise control.

4.4. REINFORCE, AC, A2C PPO vs Experience Replay Target Network

Under Identical training conditions (1 million steps and a shared network architecture), the policy gradient methods outperformed the value-based DQN variants on CartPole-v1 (as seen in figure 6 and figure 7), exhibiting faster learning, greater stability, and higher final rewards. PPO in particular showed the highest sample efficiency and stability, reaching near-maximal returns with low variance across runs. A2C also achieved robust performance and smooth convergence, clearly surpassing the simpler policy methods (REINFORCE and the basic Actor-Critic) and all value-based baselines. Among the value-based approaches, the best performance came from the DQN agent with a target network and experience replay; however, even this strongest

DQN baseline converged more slowly and less reliably than PPO or A2C. Overall, the policy-based algorithm (especially PPO and A2C) proved more robust and efficient than the most enhanced DQN, highlighting the advantages of policy gradient methods under matched conditions. Our experiments indicate

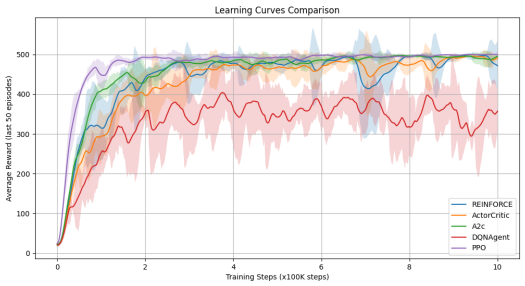

Figure 6. Comparison of REINFORCE, AC, A2C, PPO, and DQN variants.

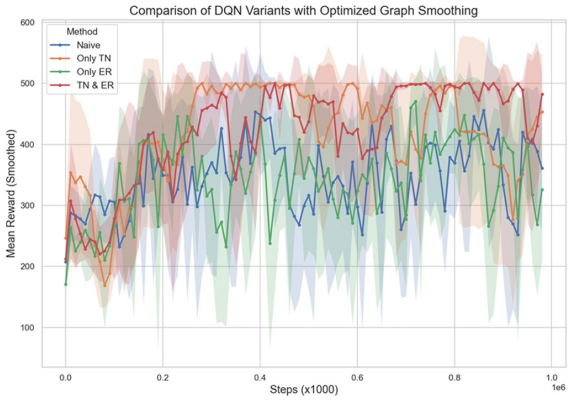

Figure 7. DQN performance comparison: Naive, TN, ER, and TN+ER.

5. Conclusion and Future Work

The assignment aimed to evaluate and compare multiple algorithms on the cartpole-v1 environment, focusing on methods like Reinforce, Actor-Critic(AC),Advantage Actor-Critic(A2C), and Proximal Policy Optimization(PPO).We choose strong value-based DQN as a baseline for our Experiment.We implemented each algorithm with consistent policy network,ran each of 5 times for 1 million steps,and tracked their performance via average episode rewards.We observed that policy gradient methods(REINFORCE, Actor-Critic, A2C)) generally show rapid early learning , but AC and A2C deferred in stability ,as they converge faster and steadily than Monte-Carlo approach of Reinforce Algorithm.Overall,our results underline the benefits of introducing an advanced policy update scheme or a critic to lessen the variance and improve learning speed. We learned that a simple algorithm like Reinforce can still achieve good performance but can suffer from high variance, whereas methods incorporating value estimation (A2C, Actor-Critic) or advanced policy regularization (PPO) which offered robust policy updates deliver more stable and faster convergence. In addition we also compared our results with naive Q-learning variants from our previous assignment- namely Naive Q-learning, Q-learning with a Target Network (TN), Experience Replay (ER), and both Target Network + Experience Replay. This lets us compare actor-critic algorithms and advanced policy-gradient against simpler,value-based baselines As of for the future work we could include Generalized Advantage Estimation for smoother advantage calculation , or clipping stratergies to ignore excessively large updates or maintain exploration pressure by using entropy regularization. For verifying the robustness of the algorithms we can try them out on different yet complex task with larger state space (e.g Atari games, LunarLander).We can also try out Actor-Critic together with an off policy replay buffer, for improving data efficiency and performance.

References

[1] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft Actor-Critic Algorithms and Applications. *arXiv preprint arXiv:1812.05905*, January 2019.

[2] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[3] V. Mnih. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[5] D. Danushidk. *PPO Algorithm*. Medium, Published February 21, 2024.

[6] Papers-100-lines. *Reinforcement Learning: PPO in Just 100 Lines of Code*. Medium, Published November 25, 2024.