

**Capstone Project Report**

**Implementing the Advantage Actor-Critic (A2C) Algorithm using a Stochastic Policy**

**Submitted By:**

**Akhil Kumar, Kartik Natarajan, Tushar Srivastava, and Nisarg Vyas**

**Date of Submission: 13.06.2025**

***Submitted in Partial Fulfillment of***

**PG Level Advanced Certificate Programme on Artificial Intelligence and Autonomous Systems**

**INDEX**

[**Introduction 3**](#_awh56n9np15q)

[**Environments 4**](#_is6yrsbkmdl)

[**Algorithms and Methods 5**](#_vjdsndqxat4r)

[**Code Implementation 8**](#_c6g1xahhbgnm)

[**Results and Conclusion 9**](#_8lhl7omkifqo)

[**Bibliography 11**](#_cqiuzhm1pifj)

## **Introduction**

* Reinforcement Learning (RL) is centered around **learning a policy that maximizes the cumulative reward** an agent receives by interacting with an environment.
* **Policy gradient methods** are **powerful for learning** in discrete, high-dimensional, or continuous action spaces. However, they suffer from the problems of **high variance and instability.**
* **Value-based methods** are typically more **sample-efficient and converge faster.** However, value-based methods often **exhibit estimation bias, particularly bootstrapping bias,** where the target value depends on the current value estimate itself
* To address these issues, **Actor-Critic methods combine the strengths** of both policy-based (actor) and value-based (critic) approaches.

**Problem Statement:**

* Design and implement an A2C (Advantage Actor-Critic) reinforcement learning agent using a stochastic Gaussian policy modeled by a feedforward neural network.
* Implement advanced variants, such as the Proximal Policy Optimization (PPO), and compare performance improvements across various environments.
* **Environments Considered:**
  + Pendulum v1(Continuous)
  + Cartpole v1 (Discrete)

**Motivation:**

* Actor-Critic algorithms form the **foundation for many advanced reinforcement learning (RL) methods**.They form the bedrock of understanding the more complex policy optimization algorithms.
* Actor-Critic models aim to **mitigate high variance and instability** by combining value-based and policy-based learning approaches.
* **Motivations behind choosing the environments:**
  + **Cartpole-v1:** Discrete Action Space with a dense reward function
  + **Pendulum-v1:** Continuous action space with dense but non-linear reward function.
  + They **cover a variety of reward structures** (dense vs sparse), helping test whether the algorithm can learn effectively with different types of feedback.
* To develop practical Reinforcement Learning skills, and use them as a basis to understand more complex frameworks like Proximal Policy Optimization (PPO).

**Objectives:**

* **To implement the Advantage Actor-Critic (A2C) algorithm** from scratch or using the PyTorch framework
* **To train and evaluate the A2C model** on two distinct environments: CartPole-v1 and Pendulum-v1
* **To analyze the performance of A2C** in terms of reward convergence, stability, and learning efficiency across different types of tasks.
* **To compare A2C with the Proximal Policy Optimization (PPO) algorithm**, a more advanced policy optimization method, on the same environments under similar training conditions.

**Approach to solve the problem:**

* **Define the environment** using OpenAI Gym (e.g., CartPole, MountainCar).
* **Implement the Advantage Actor-Critic (A2C) algorithm** with separate neural networks for the actor (policy) and critic (value function).
* **Use the policy network to sample actions**, collect rewards, and update both networks using the advantage estimate.
* **Compare performance** with a **Proximal Policy Optimization (PPO)** baseline.
* **Visualize results** using reward curves and convergence graphs for all environments.

## **Environments**

**Programming Languages:**

* **Python 3.10+:** It has a **rich ecosystem in machine learning and reinforcement learning**.

**Libraries and Frameworks:**

* **Gymnasium / OpenAI Gym –** for the CartPole and Pendulum environments.
  + **Cartpole v-1 Environment:** The objective in the CartPole-v1 environment is to balance a pole on a moving cart. This is a classic control problem with a discrete action space.
  + **Pendulum-v1 Environment:** In the Pendulum-v1 environment, the goal is to swing a pendulum up and keep it upright. This task features a continuous action space, requiring a slightly different approach to policy optimization.
* **NumPy –** for numerical operations.
* **Matplotlib and Seaborn –** for visualization of training curves.
* **Stable Baselines3 –** for using the PPO implementation and benchmarking.

**Hardware Requirements:**

* **RAM:** Minimum 8 GB.
* **GPU:** Not essential, but a basic CUDA-capable GPU (e.g., NVIDIA GTX 1650 or higher) can speed up training.

**Cloud Platforms:**

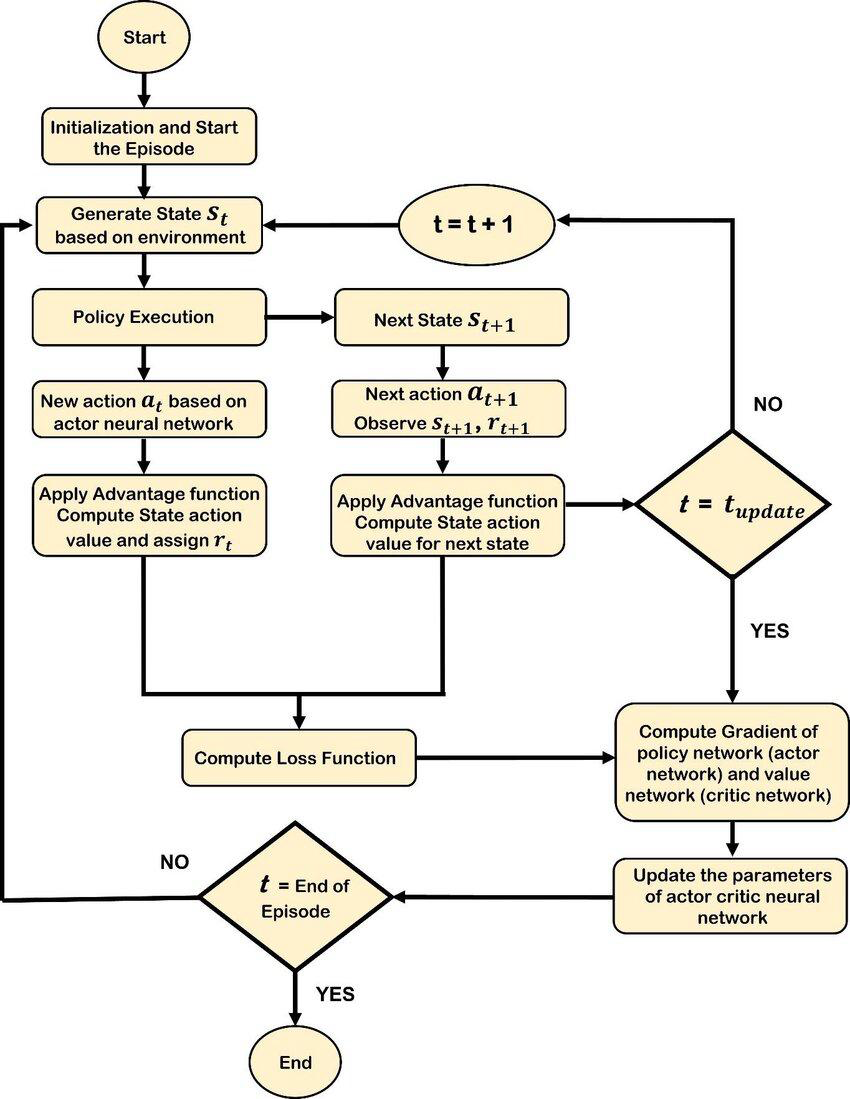
* **Google Colab** – for GPU access and running notebooks in the cloud.

## **Algorithms and Methods**

**Advantage Actor-Critic Algorithm (A2C):** The A2C architecture integrates an actor and a critic for efficient policy optimization.

* **Actor:** Selects actions based on learned policy.
* **Critic:** Evaluates actions using value function estimates.
* **TD Error:** Difference between predicted and actual returns.
* **Policy Gradient:** Optimizes policy using expected return gradients.

**Fig: A2C Flow Diagram**

****

**Approach to Solve the Problem using the A2C Algorithm:**

* **Training Loop:**
  + Initialize environment, actor and critic networks, and optimizer.
  + **For each episode:**
    - Reset the environment.
    - **For each timestep:**
      * Use the actor network to predict action probabilities.
      * Sample an action and interact with the environment.
      * Store state, action, reward, and next state.
      * Calculate TD error/advantage using the critic.
      * Accumulate gradients and update both networks.
  + Perform **policy and value updates** at the end of an episode or batch.
  + Repeat for a **fixed number of episodes or until convergence.**
* **Evaluation:**
  + Calculate the **average return** per episode
  + **Generate a simulation video.**
  + Plot **reward trends across episodes to observe variance and convergence.**
  + Sample actions using the learned policy. Save **rendered frames.**
  + Create a **visual output** of agent performance.

**Hyperparameters:**

* **Gamma (Discount Factor):**
  + Controls the importance of future rewards.
  + A **higher gamma means the agent considers long-term rewards more heavily**, while a **lower gamma focuses on immediate rewards**.
  + Typical values range from **0.9 to 0.99**.
* **Learning Rate:**
  + Determines the **step size at each iteration** while moving towards a minimum of the loss function.
  + A **small learning rate can lead to slow convergence**, while a large one might **overshoot the minimum** and fail to converge.
  + **Separate learning rates** may be used for the actor and critic.
* **Network Size:**
  + Refers to the **number of layers and neurons** in the actor and critic neural networks.
  + **Larger networks can learn more complex policies** but are prone to overfitting and require more computational resources.
  + **Smaller networks might underfit**.
* **Batch Size:**
  + The **number of samples (trajectories or experiences)** used in one gradient update step.
  + A **larger batch size provides a more accurate estimate** of the gradient but requires more memory and computational time.
  + **Smaller batch sizes can introduce more noise** but might lead to faster convergence in some cases.

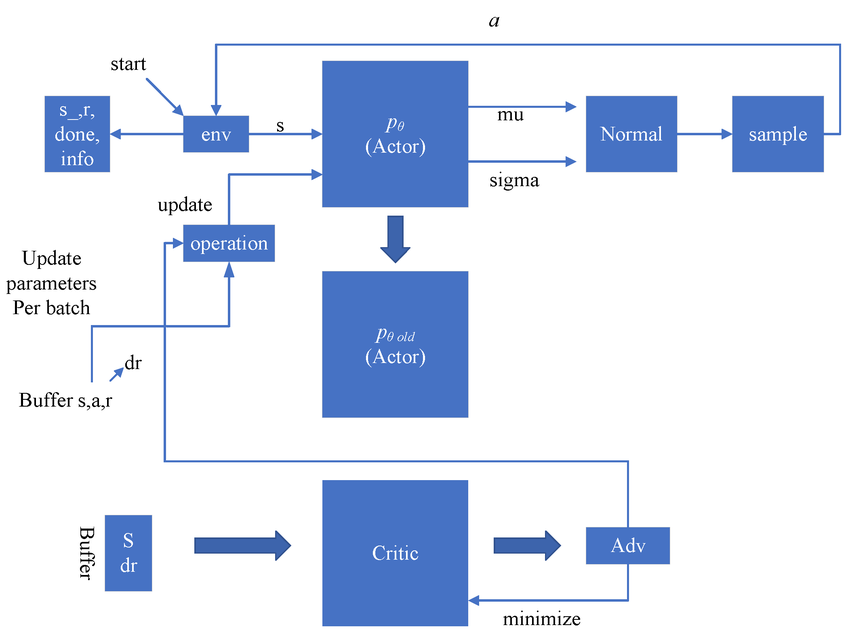
**Environment-Specific Hyperparameters:** To effectively train our A2C agent, we tailored hyperparameter configurations for each distinct environment: CartPole-v1 and Pendulum-v1. This allowed us to optimize performance based on the unique characteristics of each task.

* **CartPole-v1:**
  + Gamma (Discount Factor): 1.00
  + Learning Rate: 1e-2
  + Batch Size: Full episode
  + Entropy Bonus: 0.01
* **Pendulum-v1:**
  + Gamma (Discount Factor): 0.99
  + Learning Rate: 1e-3
  + Batch Size: Full episode
  + Entropy Bonus: 0.01

**Proximal Policy Optimization (PPO):**

* Introduced by OpenAI to strike a balance between **performance** and **stability** in policy gradient methods.
* PPO improves upon earlier policy optimization algorithms by **preventing overly large policy updates** that can destabilize learning.
* It achieves this by using a **clipped surrogate objective** that limits how much the policy is allowed to change from the previous state during training.
* **Why does it work well?**
  + Balances **exploration and exploitation** through entropy regularization.
  + More **sample-efficient and stable** than vanilla policy gradient or A2C.
  + Requires **fewer hyperparameter tricks** compared to TRPO (Trust Region Policy Optimization).
  + Works in **both discrete and continuous action spaces**, making it versatile across environments.

**Fig: PPO Flow Diagram**



**Environment-Specific Hyperparameters:** To effectively train our PPO agent, we tailored hyperparameter configurations for each distinct environment: CartPole-v1 and Pendulum-v1. This allowed us to optimize performance based on the unique characteristics of each task.

* **CartPole-v1:**
  + Gamma (Discount Factor): 0.98
  + Learning Rate: 2.5e-4
  + Clip Range: 0.2
  + Batch Size: 1
  + Entropy Bonus: 0.1
* **Pendulum-v1:**
  + Gamma (Discount Factor): 0.98
  + Learning Rate: 1e-3
  + Clip Range: 0.2
  + Batch Size: 64
  + Entropy Bonus: 0.01

## **Code Implementation**

Please refer to <https://github.com/Tusharaws/mlclasses>

## **Results and Conclusion**

**Results:**

**Table: Training Performance Analysis (PPO vs A2C)**

| **Feature / Env** | **PPO (CartPole)** | **A2C (CartPole)** | **PPO (Pendulum)** | **A2C (Pendulum)** |
| --- | --- | --- | --- | --- |
| **Learning Rate (LR)** | 2.5e-4 | 1e-2 | 1e-3 | 1e-3 |
| **Gamma** | 0.98 | 1.00 | 0.98 | 0.995 |
| **Clip Range** | 0.2 | N/A | 0.2 | N/A |
| **Epochs / Update** | 4 | 1 | 10 | 1 |
| **Batch Size** | 1 | Full episode | 64 | Full episode |
| **Sample Efficiency** | Higher | Lower | Higher | Lower |
| **Convergence Time** | Medium | Fast | Slower (but stable) | Unstable without tuning |

**Conclusion**

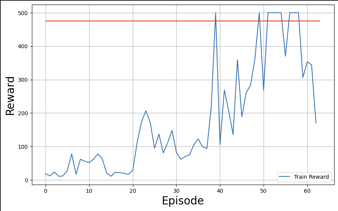
**Environment - Pendulum-V1:**

**PPO** generally works **better than A2C** for environments like Pendulum v1, which have **continuous action spaces**. PPO outperforms A2C in this case because of the following:

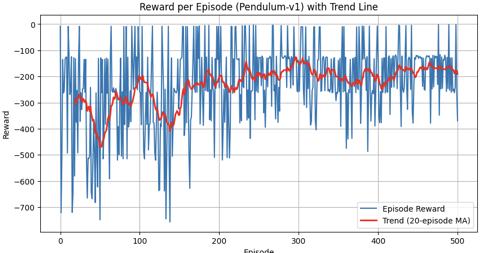
1. **Stability with Clipped Policy Updates:** PPO uses a clipped surrogate objective to prevent large, destabilizing policy updates. In continuous spaces like Pendulum-v1, policies are sensitive to small changes. PPO prevents sudden swings in policy behavior.
2. **Batch Updates:** PPO performs multiple epochs of SGD updates on mini-batches, improving sample efficiency.PPO is less sample-hungry and makes better use of each trajectory.
3. **Better Exploration Control:** PPO typically includes entropy bonuses and learnable variance in its Gaussian policy for exploration.

**Environment - Cartpole - V1:**

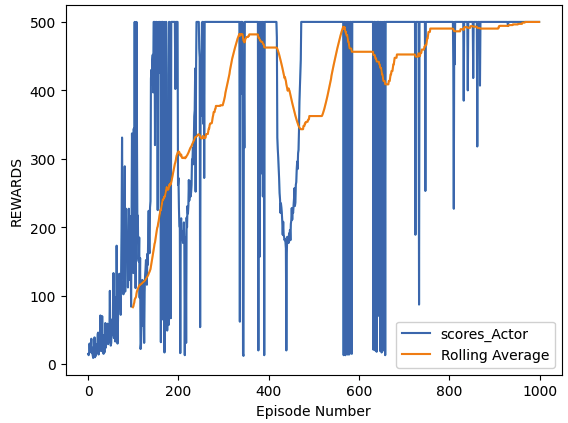
1. **Tuning Hyperparameter:** A2C performs well in a Discrete environment like CartPole-V1 if trained well with the right hyperparameter values
2. **Fast for Simple Environment:** A2C faster than PPO for a simple environment like CartPole-v1
3. **Synchronous Updates:**  This helps in the consistent policy behavior, making the training more reproducible and easier to debug.



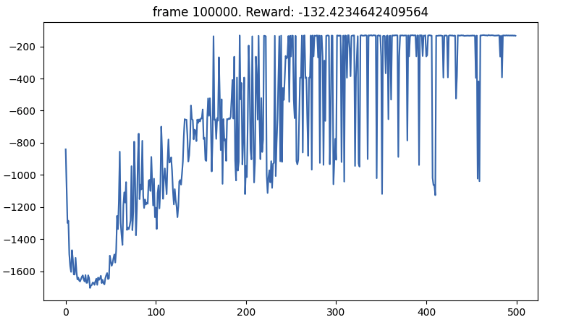
**Fig: PPO Cartpole- v1**

****

**Fig: PPO Pendulum - v1**

****

**Fig: A2C Cartpole - v1**

****

**Fig: A2C Pendulum - v1**

## **Bibliography**

1. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). MIT Press. Retrieved from <http://incompleteideas.net/book/the-book-2nd.html>
2. Mnih, V., Badia, A. P., Mirza, M., Graves, A., Harley, T., Lillicrap, T., ... & Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. arXiv preprint arXiv:1602.01783. <https://arxiv.org/abs/1602.01783>
3. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint* arXiv:1707.06347.<https://arxiv.org/abs/1707.06347>
4. Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015). Trust Region Policy Optimization. *arXiv preprint* arXiv:1502.05477.<https://arxiv.org/abs/1502.05477>
5. <https://www.researchgate.net/publication/385809725/figure/fig1/AS:11431281290366207@1731576626121/Flow-chart-of-A2C-algorithm.jpg>