Comparative Analysis of Deep Learning-based Actor-Critic Methods

Victor Micha, Fares Boudelaa

***Abstract*—This paper investigates deep learning-based Actor-Critic (AC) methods for reinforcement learning in continuous state, discrete action environments like CartPole-v1. We implement an AC algorithm per Sutton and Barto, using seperate neural networks for actor and critic, alongside variants like Deep Q-Learning with REINFORCE and GAE. The actor uses policy gradients, while the critic minimizes TD error (we also investigate a variant where the updates are done after full episodes MC style). Future work includes a shared network for the actor-critic and extensions to A3C and more diverse environments.**

I. Introduction

In this project, we focus on the family of Actor-Critic (AC) Methods. The intuition behind Actor-Critic Methods is to combine policy-based and value-based reinforcement learning algorithms for better performance. AC methods deploy two components: an actor, which selects actions based on the current policy, and a critic, which evaluates those actions using an action-value (or value) function. When coupled with deep learning, AC methods have been widely implemented across various industries. For example, AlphaGo, which revolutionized reinforcement learning, combines an Actor-Critic framework with Monte Carlo Tree Search (MCTS). Furthermore, AC methods are also used in training Large Language Models (LLMs) through Reinforcement Learning from Human Feedback (RLHF) to align text generation with human preferences. There exist various AC implementations, each leveraging distinct algorithms with unique trade-offs. In this project, we focus specifically on deep learning-based AC variations in a continuous state space and discrete action space. We also explore different architectures, which we will describe below, all of them using separate networks for the actor and the critic.

The link for the anonymous code on google colab is the following:

Notebook

II. Background

Actor critic methods are based on a combination of policy gradient and value-based methods. Let's recall the basics of policy gradient: Policy gradient methods aim to directly learn the policy rather than extracting an optimal policy from an estimated value function. Let π_θ be a parametric function with unknown parameters θ; this function is typically a neural network, with θ representing its weights. A key difference when comparing policy gradient to value-based methods is that policy gradient learns a stochastic policy (i.e., a probability distribution), meaning that the balance between exploration and exploitation is inherently built into the learning process.

The objective is to maximize the expected return:

$$J(\theta) = \mathbb{E}[G].$$

To learn the best θ, we use gradient ascent on this objective. By applying the policy gradient theorem (Sutton et al., 2000), we obtain a way to compute the gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[G_t \nabla_\theta \ln \pi_\theta(a|s) \right]$$

without needing to know the value function explicitly.

Actor-Critic is a RL method that combines policy-based and value-based approaches to train an agent. The "actor" learns a policy by directly optimizing the expected cumulative reward using policy gradients, allowing it to explore and adapt in complex environments. The "critic" estimates the value function (e.g., how good a state is under the current policy) and provides feedback to the actor by a temporal difference (TD) error. Essentially, the critic is acting as a guide to improve the decision-making of the actor. The teamwork of the actor and critic reduces the variance in gradient updates (compared to pure policy methods) and speeds up the learning (compared to pure value methods). Overall, Actor-Critic takes the advantages of both policy methods and value methods, making it a powerful RL method.

Let's now recall deep learning value-based methods. Value-based deep learning methods use a regressor to approximate the value function (or action-value function, which is conceptually similar). From this approximation, one can extract a policy; however, within the context of actor-critic methods, this estimation guides the actor. Let V_w be a function approximator (with parameters w) estimating the true value function V. We define the loss as an ordinary least squares (OLS) problem:

$$L(w) = \frac{1}{2} \mathbb{E} \left[\left(V(s) - V_w(s) \right)^2 \right].$$

Since the state space is continuous, we cannot solve this problem using classical tabular methods. Thus, we employ a neural network to estimate V. By minimizing $L(w)$, we update the weights w using gradient descent.

With this foundation for value estimation established, we now turn our attention to the specific algorithms employed in our project: Deep Q-Learning and Generalized Advantage Estimation (GAE).

Deep Q-Learning uses a simple multi-layer perceptron (MLP) to estimate the action-value function $Q(s, a)$, which enables handling large state spaces. Our MLP is composed of 3 layers and utilizes the ReLU activation function. Its design is based on the 'CSC-52081-EP Lab6: Deep Reinforcement Learning'.

Before explaining the second algorithm, let us define the advantage function as:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s),$$

which represents the advantage of taking action a in state s under policy π. This can also be expressed in terms of the temporal difference (TD) error:

$$A^{\pi}(s,a) = r(s,a) + \gamma\, V(s') - V^{\pi}(s) = \delta,$$

where δ is the TD error.

Let $\hat{A}_t$ be an estimator of $A^{\pi}(s,a)$. Generalized Advantage Estimation (GAE) is defined as the exponentially weighted sum of these TD errors over k steps:

$$\hat{A}_t = \sum_{l=0}^{k-1} (\gamma\lambda)^l\, \delta_{t+l},$$

where $\lambda \in [0,1]$ is a hyperparameter that controls the bias-variance tradeoff. When $\lambda = 0$, GAE reduces to TD-learning (one-step returns), and when $\lambda = 1$, it approximates Monte Carlo (MC) learning.

Finaly we introduce Proximal Policy Optimisation. Inspired by Trust Region Policy Optimisation, Proximal Policy Optimisation (PPO) is an on-policy gradient algorithm that moderates the size of learning updates during training to improve sample efficiency and stability. PPO effectively searches for the optimal policy improvement step based on the currently available data. Although several PPO variants exist, this report focuses exclusively on PPO-Clip, which, as the name suggests, "clips" the gradient whenever updates exceed a predetermined range.

The PPO-Clip algorithm is formally defined through its Clipped Surrogate Objective function:

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

Where:

- θ represents the policy parameters
- $\hat{\mathbb{E}}_t$ denotes the empirical expectation over a batch of timesteps
- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the current policy $\pi_\theta(a_t|s_t)$ and the old policy $\pi_{\theta_{\text{old}}}(a_t|s_t)$
- $\hat{A}_t$ is the estimated advantage function at timestep t
- $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ constrains the probability ratio $r_t(\theta)$ to the range $[1-\epsilon, 1+\epsilon]$, with ϵ serving as a hyperparameter

The first term within the minimum operation corresponds to the standard TRPO loss, while the second term modifies the probability ratio by replacing $r_t(\theta)$ with $1-\epsilon$ when it falls below this threshold, or with $1+\epsilon$ when it exceeds the upper bound. This clipping mechanism effectively restricts policy updates to a trust region, preventing excessively large updates that could destabilize training. For a more comprehensive explanation of the probability ratio and additional implementation details, readers are directed to the original PPO paper.

Using what we explained previously, let's define Actor-Critic methods. As the name suggests, these methods use two distinct components to learn an optimal policy. The *actor* represents the policy; it learns to select actions based on a policy gradient, using its current knowledge of the state space. The *critic*, on the other hand, estimates the value function of the state space. After each complete episode, the critic evaluates the actor's actions and updates its value estimates accordingly, so that the actor can improve its decision-making.

III. Methodology/Approach

We will be testing our agent on the CartPole-v1 environment. The reason for this is that it has a continuous state space and a discrete action space, which is the kind of environment we are interested in researching for this project.

We use seperate NN for the actor and for the critic, learning using the methods described earlier. The algorithms and reasoning for this project were inspired by Reinforcement Learning: An Introduction by Richard Sutton and Andrew Barto [1]. We implement four different Actor-Critic algorithms:

1) **Deep Q-Learning with REINFORCE:** A classical Actor-Critic implementation where the actor is trained using the REINFORCE algorithm, and the critic is updated using Deep Q-Learning.
2) **Advantage Actor-Critic (A2C):** This method replaces the action-value estimate in the REINFORCE algorithm with an advantage function, reducing variance and improving the learning process.
3) **Generalized Advantage Estimation (GAE):** In this variant, the critic uses Generalized Advantage Estimation to compute more robust advantage estimates, balancing the trade-off between bias and variance.
4) **Vanilla Actor-Critic:** This approach utilizes an MLP and incorporates critic updates after every step (TD style) and another variation with updates after each episode (MC style).
5) **Proximal Policy Optimization:** A state-of-the-art actor-critic method that employs the Clipped Surrogate Objective to update the policy (actor) while utilizing Generalized Advantage Estimation (GAE) to compute advantage estimates.

The actor generates action probabilities through a softmax layer, trained using a policy gradient loss ($-\log \pi(a|s)\cdot\delta$). The critic estimates state values, minimizing the squared temporal difference error (δ^2).

At first, the agent fell into deterministic policies, repeatedly selecting the same action regardless of the state, which limited performance strongly. At first, we tried resolving this issue by introducing entropy into the loss, to balance exploration and exploitation. The intuition behind using the entropies of the action distributions is that if this distribution has a very low entropy, the agent will not often perform exploration. By integrating entropy into the loss and penalizing action distributions with low entropy, we force the agent to not only exploit, but explore as well. However, we found that entropy was not the best solution, which is why we did not keep it in our final version. We explore the alternative solutions we found (instead of entropy) in the next section.

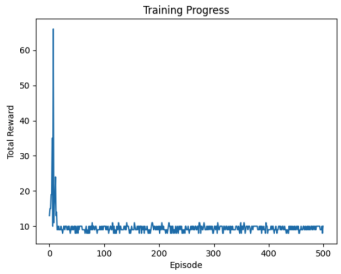


Fig. 1: Collapsing Rewards

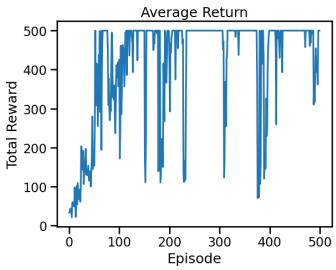


Fig. 2: AC Rewards

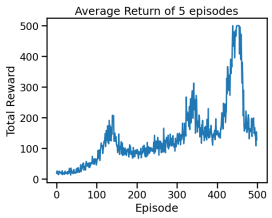


Fig. 3: A2C average rewards

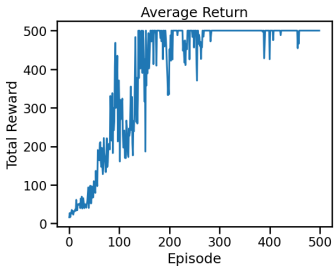


Fig. 4: GAE

IV. Results and Discussion

Our implementation builds upon the deep reinforcement learning framework introduced in Lab 6 of the CSC-52081-EP course. We extend this foundation by focusing specifically on actor-critic methods, which effectively combine policy gradient approaches with value function approximation techniques.

We evaluate model performance using the average return over five episodes per training iteration.

Starting with Deep Q-Learning combined with the REINFORCE algorithm, we construct a Value Network using a straightforward neural architecture consisting of two hidden layers, each containing 128 neurons with ReLU activation functions. The network processes state observations and produces a single scalar value estimate (This model will also be used in the A2C, GAE and PPO). We use a learning rate of 0.1 for both the actor and the critic. With this, we observe fast convergence but highly unstable training. This instability arises due to the Monte Carlo update method, which, while effective in capturing long-term dependencies, introduces high variance. As a result, the neural network learns specific scenarios well but struggles to generalize across different situations.

In contrast, the Advantage Actor-Critic (A2C) method exhibits more stable learning. We use a learning rate of 0.001 for the actor and 0.005 for the critic. This stability stems from three key factors: the advantage function, which reduces variance in policy updates by subtracting the baseline value estimate, the use of a dedicated value network, which provides more consistent learning targets, and the structured nature of baseline subtraction, which ensures that updates focus on how much better an action is compared to the expected return. However, A2C converges more slowly than REINFORCE due to the increased computation and the reliance on Temporal Difference (TD) learning which requires more iterations for policy refinement. We use

Generalized Advantage Estimation (GAE) demonstrates the expected trade-offs between convergence speed and stability. We use a custom learning rate scheduler that applies exponential decay while maintaining a minimum learning rate threshold, ensuring the model continues to learn effectively even during extended training periods. The scheduler follows:

$$\text{lr}_t = \max(\text{base\_lr} \cdot \gamma^t, \text{min\_lr})$$

Where base_lr is the initial learning rate, γ is the decay factor, t is the current epoch, and min_lr is the minimum learning rate threshold.

The method coupled with the decreasing learning rate accelerates learning by balancing bias and variance through the λ parameter. We experiment with multiple values as seen in Figure 10 and find $\lambda = 0.95$ provides the best trade-off. (This value and the scheduler will be use for the PPO implementation later).

This allows GAE to better extract reward compared to traditional TD methods while maintaining a more stable learning process. Initially, we observed instability due to poor early value estimates, which led to inaccurate advantage estimates and subsequent suboptimal policy updates. These compounding errors in multi-step returns amplified instability, and the initial high variance in advantage estimates resulted in large, unstable parameter updates. To remediate this issue, we employed batch gradient calculation. Specifically in PyTorch, rather than calculating the loss iteratively for every timestep in each episode, we calculated the loss and stepped the optimizer in batches, as seen in Figure 9. We observed that smaller batches yield faster learning at the cost of stability and vice versa. With a batch size of 32, we found the optimal trade-off between learning speed and stability. Over time, as the value network improves, these issues diminish, and GAE provides faster and more stable convergence compared to simpler actor-critic approaches.

We will now discuss the results of the vanilla AC methods using TD updates, then MC updates. Firstly, as mentioned

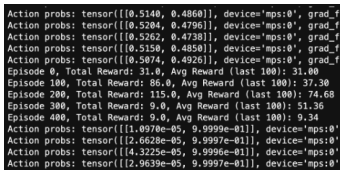


Fig. 5: Low Entropy Action Distributions

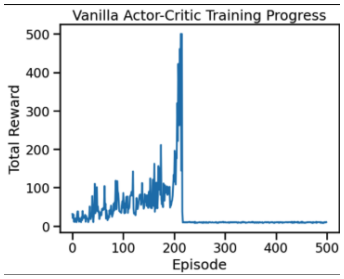


Fig. 6: Second Example Collapsing Rewards

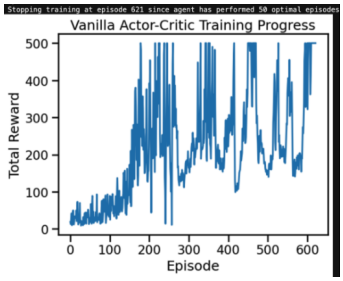


Fig. 7: AC (TD) stops training after 50 optimal episodes, at episode 621/1000

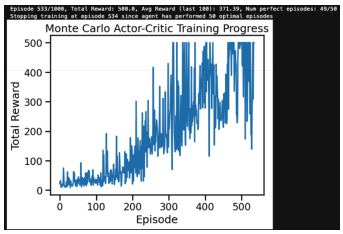


Fig. 8: Rewards AC with MC updates

above, a vanilla AC suffers from convergence to action distributions with very small entropies (Figure 5), causing the actor to consistently select the same action and never exploring. This lead to terrible rewards per episodes, as can be seen in Figure 1 (near zero with little fluctuation).

We show another example of collapsing rewards in Figure 6, where the agent had solved the cartpole game with reward 500 after training for 213 episodes. As it continued training to 500, the action distributions were pulled to extreme distributions with very little entropy where they cannot change back. This is the problem we will try to solve in the following paragraphs. We could introduce the concept of entropy to the loss, such that the action distributions with low entropies are penalized. However, we try something else, something much simpler yet suprisingly effective.

Instead of introducing the concept of entropy to the loss, to solve the problem of very low entropy action distributions, we instead recognize that we could simply stop the training once the agent has completed a certain number (hyperparameter) of perfect episodes. The reasoning behind this, is that if the agent has completed enough perfect episodes, we can assume that it has reached optimal (or close to optimal) learning, and can stop training since it has a better chance of performing well on other episodes. This does indeed assume that we know the max reward of the environment, which is not always available (but 500 for Cartpole-v1).

We also tried to lower the learning rate so the total rewards every episode during training do not fluctuate too much (less variance), so that the actor and critic do not learn too fast. This helped the total rewards to steadily reach 500 (max number of rewards). For this AC, we used $\alpha_\theta = 0.0001$ (learning rate of actor/policy) and $\alpha_w = 0.002$ (learning rate of critic /value function), so that the value function is learned as fast as possible (while remaining stable) and the policy takes very slow learning steps, waiting for the value function to be well learned.

We can see the results of the two changes to the vanilla AC described above (stopping training after certain number of perfect episodes + smalled learning rate of policy compared to critic) in the Figure 7.

We test the results of this trained agent on 1000 episodes, and it got an average of 500 reward, meaning that it completed only perfect episodes! This means that this agent sovled the Cartpole-v1 problem (with hyperparameters found in the code).

We can now discuss the results of the vanilla AC with MC updates, after each episode. This agent suffered the same problem as the TD AC, so we implemented the same solutions (stopping training after certain number of perfect episodes + smalled learning rate of policy compared to critic). Not suprisingly, this worked for the MC AC as well, and we can see from Figure 8 that the total rewards are less erratic.

We also test this agent on 1000 episodes, with average reward of 485.876, meaning that the avg reward is 97.18% of the max reward, which we consider good enough to solve the Cartpole-v1 environment.

Finally, we analyze the performance of PPO-Clip. As illustrated in Figure 11, we conducted experiments with multiple values for the clipping parameter ϵ. Our results demonstrate that $\epsilon = 0.2$ provides the optimal balance between learning stability and convergence speed. This implementation clearly outperforms our previous methods, with the agent effectively mastering the CartPole environment within 200 epochs. Notably, even with suboptimal clipping values, the model exhibits more stable learning behavior compared to our alternative implementations. For consistency in our comparative analysis, we maintained the same hyperparameters as in our GAE implementation, isolating the effect of the clipping mechanism.

V. Future Work

Building upon our current analysis of actor-critic methods, several promising directions for future research emerge.

Our investigation could be further extended to environments with continuous action spaces, which present distinct challenges compared to the discrete action spaces explored in this study. Such an extension would provide valuable insights into the generalizability of our findings across different types of reinforcement learning problems.

Additionally, we aim to implement and evaluate Asynchronous Advantage Actor-Critic (A3C). A3C employs multiple independent agents operating in parallel, each interacting with separate instances of the environment and asynchronously updating a global network. This parallel architecture potentially offers significant improvements in training efficiency and stability compared to single-agent approaches.

VI. Conclusions

The vanilla AC agents using TD and MC updates show that while the problem of low entropy action distributions hindered learning, there was no need to add entropy to the loss, we instead had to simply stop training once the agent got good enough (and have a lower learning rate for the policy than the critic). Moreover the GAE and PPO implementations show that will more state of the art algorithmes we can much more stable training with just some simple finetuning.

References

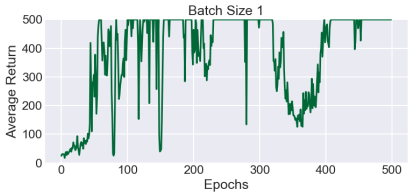
[1] Sutton and Barto. Reinforcement Learning, *MIT Press*, 2020.

[2] Read. Lecture VI - Reinforcement Learning III. In *INF581 Advanced Machine Learning and Autonomous Agents*, 2025.

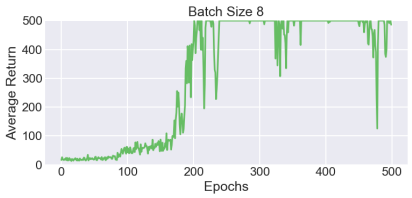
[3] Decock. Lab6: Deep Reinforcement Learning In *INF581 Advanced Machine Learning and Autonomous Agents*, 2025.

[4] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-Dimensional Continuous Control Using Generalized Advantage Estimation, *arXiv preprint arXiv:1506.02438*, 2015.

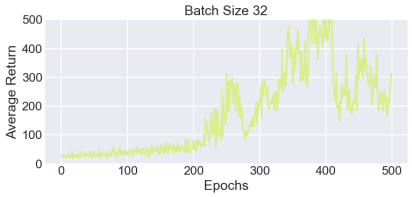
[5] Schulman, John and Wolski, Filip and Dhariwa *arXiv preprint arXiv:1707.06347*, 2017



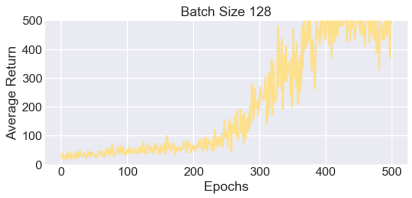
(a)



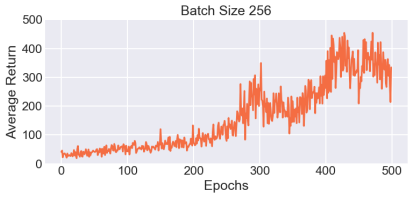
(b)



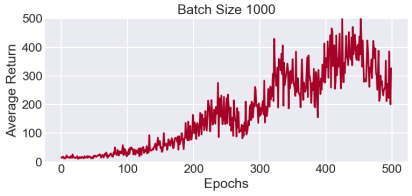
(c)



(d)

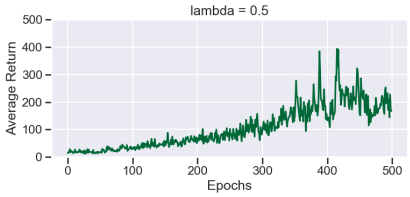


(e)

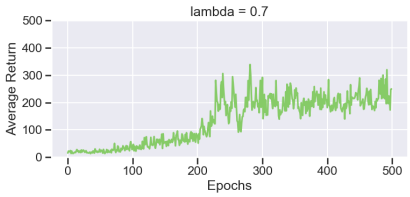


(f)

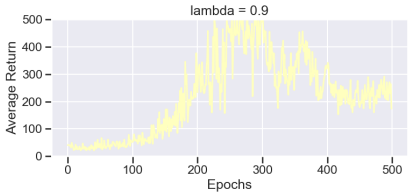
Fig. 9: Different batch sizes values for the GAE AC implementation



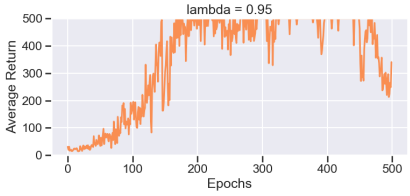
(a)



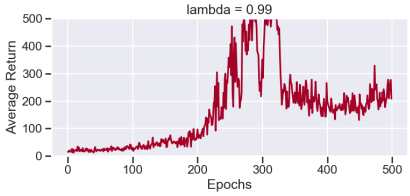
(b)



(c)

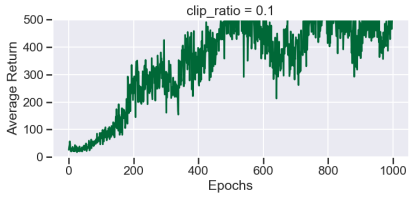


(d)

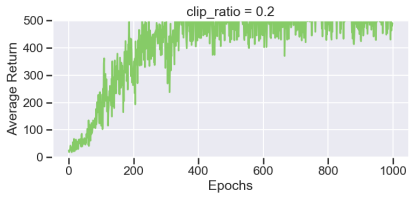


(e)

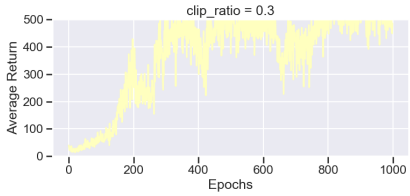
Fig. 10: Different λ values for the GAE AC implementation



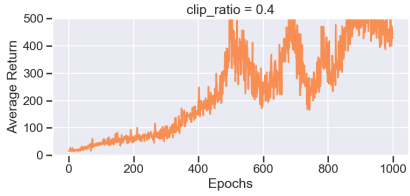
(a)



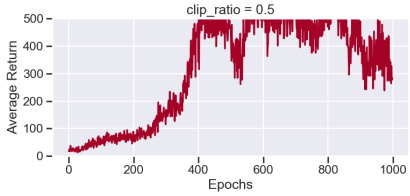
(b)



(c)



(d)



(e)

Fig. 11: Different ϵ values for the PPO implementation