Reinforcement Learning Assignment 2: A Research Study on Policy Based Reinforcement Learning

Benard Wanyande [*1] **Ataklti Kidanemariam** [*1]

Abstract

This document provides a deep dive into the world of policy-based reinforcement learning. It attempts to explain its inner workings, provides an overview of the transition from value-based reinforcement learning to actor-critic methods, and provides a comprehensive analysis of experiments performed using actor-critic methods. In addition to the base experiments based on actor-critic methods, some bonus experiments are analyzed.

1. Introduction

Reinforcement Learning (RL) has been successfully applied to a wide range of sequential decision-making problems, including game playing, robotic control, and resource optimization. In prior work, we have tackled classical problems such as the CartPole environment, which serves as a standard benchmark in RL research. Beyond academic benchmarks, RL has also demonstrated remarkable success in domains such as board games (e.g., chess and Go) (Silver et al., 2017), and Atari games (Mnih et al., 2015), largely using value-based methods such as Q-learning and Deep Q-Networks (DQN).

These value-based methods are particularly effective in environments with discrete action spaces, where the optimal policy can be derived using an `argmax` operation over the estimated value function. However, many real-world problems—such as autonomous driving, robotic manipulation, and continuous control tasks—involve continuous action spaces. In such settings, value-based approaches become less effective and often unstable, primarily due to the difficulty of computing the exact `argmax` in continuous domains (Lillicrap et al., 2015).

To address these limitations, policy-based reinforcement learning methods have been introduced. Instead of learning a value function to derive a policy, these methods aim to directly learn a parameterized policy that maps states to actions (Sutton et al., 2000). This approach provides greater flexibility and stability in continuous action spaces and naturally allows for stochastic policies, which are essential for effective exploration.

In this report, we begin by presenting the theoretical foundations of policy-based methods, focusing on the REINFORCE algorithm (Williams, 1992), a Monte Carlo policy gradient method. We then extend this discussion to include actor-critic (AC) architectures, which combine value function estimation with policy optimization to reduce variance and improve sample efficiency.

Next, we introduce the concept of the advantage function and explain how it contributes to more stable learning. We subsequently explore the Advantage Actor-Critic (A2C) algorithm, a synchronous variant of AC that strikes a balance between bias and variance while maintaining computational efficiency.

Following the theoretical exposition, we implement the REINFORCE, AC, and A2C algorithms using Python and evaluate their performance in standard control environments. A comparative analysis is conducted to highlight the differences in convergence behavior, sample efficiency, and overall performance.

Finally, we conduct additional experiments to further solidify our understanding of these algorithms and explore the effects of key hyperparameters and architectural choices on learning dynamics.

2. Theory

2.1. REINFORCE algorithm

The foundation of the vanilla policy gradient method, commonly known as the REINFORCE algorithm, originates from the work of Ronald J. Williams in 1992. This method formulates a principled way of directly optimizing stochastic policies by following the gradient of expected returns with respect to the policy parameters. The key insight is to use sampled trajectories to compute an unbiased estimate of the policy gradient, which is then used to update the

[*]Equal contribution [1]Leiden University, Leiden Institute of Advanced Computer Science, Leiden, Netherlands. Correspondence to: Benard Wanyande <b.a.wanyande@umail.leidenuniv.nl>, Firstname2 Lastname2 <first2.last2@www.uk>.

policy in the direction that maximizes expected cumulative rewards. This approach laid the groundwork for subsequent developments in policy-based reinforcement learning and remains a fundamental baseline for comparison in modern policy optimization methods (Williams, 1992).

Algorithm 1 REINFORCE algorithm/Vanilla loss algorithm

Input: A differentiable policy $\pi_\theta(a|s)$, $\alpha \in \mathbb{R}^+$, threshold $\in \mathbb{R}^+$
Initialization: Initialize parameters θ in $\mathbb{R}^d$
repeat
  Generate full trace $\tau = s_0, a_0, r_0, s_1, ..., s_T$ following $\pi_\theta(a \mid s)$
  for $t \in 0, ..., T-1$ **do**
    $R \leftarrow \sum_{k=1}^{T-1} \gamma^{k-t}.r_k$
    $\theta \leftarrow \theta + \alpha\gamma^t R \nabla_\theta \log \pi_\theta(a_t|s_t)$
  end for
until $\nabla_\theta J(\theta)$ converges below $\in$
return Parameters θ

2.2. Transition to Actor-Critic (AC) Methods

2.2.1. The terms "actor" and "critic"

In the actor-critic paradigm, learning is decomposed into two key components: the actor, responsible for updating the policy to choose actions, and the critic, which evaluates the quality of these actions via a learned value function. Although value-based techniques like Q-learning are effective in discrete action domains, their loss formulations are incompatible with actor-critic methods. Q-learning relies on a deterministic update rule that involves selecting the action with the maximum value, an operation that does not align with the stochastic and parameterized nature of policies in actor-critic algorithms.

2.2.2. Why we cannot use Q-learning loss in the Basic AC approach

Because policy-based methods optimize expected returns through gradient ascent on the policy parameters, attempting to use a Q-learning style loss would not yield valid gradients for the actor.

2.2.3. Use of Advantage Function Instead of Loss

Instead, actor-critic approaches employ policy gradient methods that leverage a measure of action quality, such as the advantage function, to guide policy updates in a way that conforms with the policy gradient theorem. This formulation ensures that the updates are both effective and stable, combining the benefits of value estimation and direct policy optimization (Sutton et al., 2000; Mnih et al., 2015).

2.2.4. The Actor-Critic (AC) algorithm (Vanilla)

The Actor-Critic (AC) algorithm merges policy gradient and value-based methods by having an actor that updates the policy and a critic that estimates the value function. The critic provides a baseline to reduce the variance of policy gradient updates, and its value estimates can be used to bootstrap the return. The advantage estimate, computed as the difference between the bootstrapped return and the baseline value, is used to scale the policy gradient, encouraging the actor to prefer actions that lead to higher-than-expected returns (Sutton & Barto, 2018).

Algorithm 2 Actor-Critic (Vanilla)

Input: Policy $\pi_\theta(a|s)$, Value function $V_\phi(s)$, learning rate α, number of episodes M
Initialize: Randomly initialize parameters θ and ϕ
for $i = 1$ to M **do**
  Sample episode $\tau = \{s_0, a_0, r_0, s_1, \ldots, s_T\}$ following policy π_θ
  for $t = 0$ to $T-1$ **do**
    Compute TD error:
      $\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$
    Update value parameters:
      $\phi \leftarrow \phi + \alpha\delta_t \nabla_\phi V_\phi(s_t)$
    Update policy parameters:
      $\theta \leftarrow \theta + \alpha\delta_t \nabla_\theta \log \pi_\theta(a_t|s_t)$
  end for
end for
Return: Parameters θ

2.2.5. The Advantage Actor-Critic (A2C) algorithm

The Advantage Actor-Critic (A2C) algorithm is a synchronous variant that simplifies the more complex asynchronous approach. In A2C, multiple agents collect data in parallel environments, but their updates are aggregated and applied synchronously, ensuring stable updates and easier implementation. A2C improves training stability compared to vanilla actor-critic by using multiple samples per update and by incorporating n-step returns and advantage estimates (Mnih et al., 2016).

2.2.6. The Asynchronous Advantage Actor-Critic (A3C)

Asynchronous Advantage Actor-Critic (A3C), explored as a bonus in this work, builds upon the actor-critic architecture by training multiple actor-learners asynchronously in separate environments. Each learner computes its own gradients and periodically updates a shared global model. This decouples the data collection process across environments and improves the stability of training while accelerating

Algorithm 3 Actor-Critic with Bootstrapping and Baseline Subtraction

Input: Policy $\pi_\theta(a|s)$, Value Function $V_\phi(s)$
Estimation depth n, learning rate α, number of episodes M
Initialization: Randomly initialize parameters θ and ϕ
while not converged **do**
  for $i = 1$ to M **do**
    Sample trajectory $\tau = \{s_0, a_0, r_0, s_1, \ldots, s_T\}$ using policy $\pi_\theta(a|s)$
    for $t = 0$ to $T-1$ **do**
      Compute n-step target:
      $\hat{Q}_n(s_t, a_t) = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V_\phi(s_{t+n})$
      Compute advantage estimate:
      $\hat{A}_n(s_t, a_t) = \hat{Q}_n(s_t, a_t) - V_\phi(s_t)$
    end for
  end for
  Update critic parameters:
  $\phi \leftarrow \phi - \alpha \sum_t \nabla_\phi \hat{A}_n(s_t, a_t)^2$
  Update actor parameters:
  $\theta \leftarrow \theta + \alpha \sum_t \hat{A}_n(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)$
end while
Return: Parameters θ

Algorithm 4 Asynchronous Advantage Actor-Critic (A3C)

Input: Global shared parameters θ, ϕ; shared counter $T = 0$
Initialize: Thread-local parameters $\theta' \leftarrow \theta$, $\phi' \leftarrow \phi$
while $T < T_{max}$ **do**
  Reset gradients $d\theta \leftarrow 0$, $d\phi \leftarrow 0$
  Sync local params: $\theta' = \theta$, $\phi' = \phi$
  $t_{start} \leftarrow t$
  Observe initial state s_t
  repeat
    Select action $a_t \sim \pi(a_t|s_t; \theta')$, observe r_t, s_{t+1}
    $t \leftarrow t+1$, $T \leftarrow T+1$
  until terminal or $t - t_{start} = t_{max}$
  Compute return R based on s_t (0 if terminal, else $V(s_t; \phi')$)
  for step i in reverse **do**
    $R \leftarrow r_i + \gamma R$
    $d\theta \mathrel{+}= \nabla_{\theta'} \log \pi(a_i|s_i)(R - V(s_i; \phi'))$
    $d\phi \mathrel{+}= \nabla_{\phi'} (R - V(s_i; \phi'))^2$
  end for
  Apply $d\theta$, $d\phi$ to update θ, ϕ
end while

convergence through parallelism (Mnih et al., 2016). The comparison between A2C and A3C illustrates how architectural variations can address sample correlation and improve scalability in policy gradient methods.

3. Experiments

3.1. Overview

The primary objective of this study was to evaluate and compare the performance of three core policy gradient algorithms — REINFORCE, Actor-Critic (AC), and Advantage Actor-Critic (A2C) — on two environments: the well-known `CartPole-v1`, and the more challenging `Acrobot-v1` as a bonus experiment.

We implemented all algorithms using a shared actor-critic neural network architecture, which was imported from a common `models.py` module. The architecture used a shared encoder for the actor, a softmax output for the action distribution (discrete), and a multilayer perceptron for the critic. The objective was to ensure fairness in architectural capacity while isolating algorithmic differences.

In `CartPole-v1`, the agent's goal is to balance a pole on a moving cart by applying left or right forces. The environment returns a reward of $+1$ for each time step the pole remains upright. Episodes terminate after 500 steps or when the pole falls. Therefore, a maximum possible episodic return is 500, and higher returns correspond to better stability and control. Good performance is typically defined as an agent consistently achieving returns close to or above 450.

`Acrobot-v1`, on the other hand, is a much harder environment due to its sparse rewards and delayed feedback. The agent controls a two-link pendulum, aiming to swing its end effector above a given height. The environment penalizes the agent with -1 at each step until the task is accomplished or 500 steps are reached, making -500 the worst possible return. A good policy will learn to solve the task efficiently and achieve returns significantly better than -500 — for example, returns above -150 indicate meaningful progress.

These benchmarks set the stage for evaluating the sample efficiency and stability of the three algorithms under comparison across both dense and sparse reward scenarios.

3.2. Initial Hypothesis

Before conducting our experiments, we hypothesized that the Advantage Actor-Critic (A2C) algorithm would outperform both the vanilla Actor-Critic (AC) and the REINFORCE algorithm in terms of learning speed, sample efficiency, and final performance across standard control environments. This assumption was based on A2C's use of advantage estimation and bootstrapped value targets, which are theoretically known to reduce gradient variance and stabilize learning.

We further expected that the Actor-Critic method would

offer a moderate improvement over REINFORCE by leveraging a value function for bootstrapped return estimates, thereby mitigating some of REINFORCE's high-variance updates.

Thus, our working hypothesis for the comparative performance hierarchy was:

$$\textbf{A2C} > \textbf{AC} > \textbf{REINFORCE}$$

Additionally, we anticipated that this ranking would hold more strongly in sparse-reward environments such as `Acrobot-v1`, where variance-reduction techniques are crucial. Conversely, in dense-reward environments like `CartPole-v1`, we expected REINFORCE to remain competitive despite its simplicity, due to the frequent reward feedback. We also expected that performance would be sensitive to hyperparameter tuning (e.g., learning rate, network size), and that uniform architectural settings might not equally favor all algorithms.

3.3. Experimental Setup

All experiments were conducted using environments from the OpenAI Gymnasium library. To ensure fair and reproducible comparisons, we fixed the random seed across all runs and used consistent training procedures. Each algorithm was implemented using PyTorch and trained on the same hardware. Experiments were run for 5 repetitions per algorithm, each using a budget of 1,000,000 environment steps. Results were logged as episode-level returns and smoothed for visualization.

Before finalizing the main experiments, we conducted extensive hyperparameter tuning and ablation studies to find the most effective configuration. Specifically, we examined: the effect of varying the global learning rate, using different learning rates for the actor and critic and the impact of increasing the critic network's capacity.

From these ablations, we found that the best performance across both environments (CartPole and Acrobot) was obtained using a learning rate of 0.0005 for both actor and critic, and a discount factor of $\gamma = 0.99$.

To ensure stable learning, we adopted a shared actor-critic neural architecture. The actor consists of a shared encoder (64 hidden units), followed by a softmax policy head. The critic network was expanded to use two hidden layers with 128 units each, as this yielded the best value estimates in our ablations. The full architecture is described below:

```
class SharedActorCritic(nn.Module):
def __init__(self, obs_dim, n_actions,
actor_hidden=64, critic_dims=[128,
128]):
super().__init__()
actor, critic definitions...
```

This architecture was used consistently for all algorithms (REINFORCE, Actor-Critic, and A2C) to isolate the effect of algorithmic differences. This tuning process significantly improved learning stability, particularly for AC and A2C on the more challenging Acrobot environment.

3.4. Results

3.4.1. CartPole

As shown in Figure 1, the REINFORCE algorithm demonstrated strong performance on CartPole, converging to near-optimal behavior despite its high-variance nature. This success is largely attributed to CartPole's dense and frequent reward structure, which allows REINFORCE's Monte Carlo updates to remain informative and stable. The algorithm benefits from the consistent feedback and relatively short episodes, which reduce the variance of return estimates.

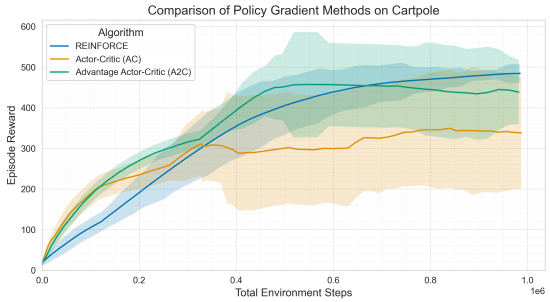

Figure 1. Comparison of REINFORCE, Actor-Critic (AC), and Advantage Actor-Critic (A2C) on the `CartPole-v1` environment over 5 training runs. The x-axis shows the total number of environment steps, and the y-axis represents episode reward. Shaded regions indicate one standard deviation across runs. A2C demonstrates the fastest and most stable convergence, while REINFORCE steadily approaches the performance ceiling with lower initial sample efficiency. Actor-Critic suffers from instability and diverges toward the end of training. This illustrates that even though REINFORCE is high-variance, its simple gradient estimate works well in dense-reward environments, whereas A2C benefits from reduced variance and faster learning via advantage estimation.

In comparison, both Actor-Critic (AC) and Advantage Actor-Critic (A2C) exhibited faster early learning, indicated by the steeper reward curves in the initial phase of training. This steepness reflects how quickly these algorithms leverage

bootstrapping (via the critic) to improve the policy. A2C consistently outperformed AC across the entire training horizon, demonstrating the benefits of using the advantage function for variance reduction.

3.4.2. ACROBOT (BONUS)

The results for Acrobot, shown in Figure 2, reveal a stark contrast. Here, REINFORCE failed to make meaningful progress, staying close to the lower bound of -500 for most of training. This is due to the environment's sparse and delayed rewards, which severely impair REINFORCE's ability to compute reliable return estimates. With long episode horizons and minimal feedback, the high variance of REINFORCE gradients becomes a significant limitation.

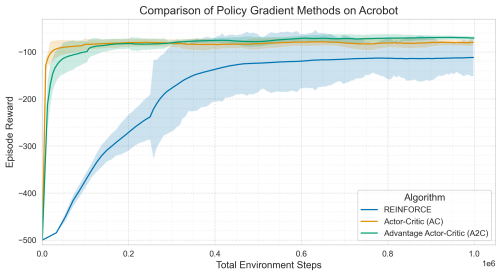

Figure 2. Performance comparison of REINFORCE, Actor-Critic (AC), and Advantage Actor-Critic (A2C) on the `Acrobot-v1` environment across 5 training repetitions. The x-axis indicates total environment steps, while the y-axis shows episodic returns, which are bounded between -500 (worst) and 0 (best). Shaded regions show standard deviation. REINFORCE struggles to improve due to Acrobot's sparse and delayed reward signal. Both AC and A2C learn effective policies, with A2C outperforming AC in both speed and final performance. These results reinforce the utility of bootstrapped and advantage-based updates in difficult learning environments.

AC and A2C, on the other hand, successfully overcame these challenges. A2C in particular showed early and sharp gains in episodic return and stabilized around -100 to -120, revealing its robustness in sparse-reward settings. The n-step advantage estimates allowed A2C to bridge the temporal gap between actions and rewards more effectively than AC's simpler TD approach.

Overall, the experiments demonstrate that while REINFORCE can perform surprisingly well in favorable conditions (like CartPole), actor-critic methods — especially A2C — exhibit broader adaptability and more sample-efficient learning, particularly in harder tasks like Acrobot.

3.4.3. ANALYSIS AND INSIGHTS

Our initial hypothesis, anticipating a hierarchy of performance where **A2C** $>$ **AC** $>$ **REINFORCE** , was largely validated by the experimental results. A2C consistently outperformed both AC and REINFORCE in terms of sample efficiency and final performance across both environments. AC occupied the middle ground, with reasonably fast learning but lower asymptotic performance. REINFORCE, while simple and stable in certain settings, lagged significantly in sparse-reward environments.

However, the results also uncovered important nuances. First, REINFORCE was surprisingly competitive on CartPole, challenging the assumption that it is categorically inferior. This demonstrates that environment dynamics and reward density play a crucial role in determining algorithm suitability. Second, hyperparameter tuning—including actor/critic layer sizes and learning rates—had a profound effect on performance. The final configuration (learning rate = 0.0005, $\gamma = 0.99$, moderate model capacity) was only discovered after careful ablation studies, emphasizing the importance of principled experimentation.

Finally, these findings underscore the broader insight that no single algorithm dominates universally. While A2C may be preferable for its efficiency and stability, simpler methods like REINFORCE can still be highly competitive in specific tasks. The choice of method should be informed by the structure of the environment, reward signal sparsity, and computational constraints.

3.4.4. COMPARISON TO VALUE-BASED METHODS

In Assignment 1, we explored value-based reinforcement learning using Deep Q-Networks (DQN) with several stabilization techniques such as experience replay and target networks. These methods showed strong performance on environments like CartPole, with stabilized variants significantly outperforming the naive DQN baseline (see Figure 4).

While DQN and its stabilized versions perform well in discrete action settings, policy-based methods offer several advantages: they support stochastic policies, are more suited to continuous action spaces, and can offer improved exploration. As seen in our policy gradient experiments, algorithms like A2C achieved comparable or superior sample efficiency to stabilized DQNs in CartPole, and also demonstrated better robustness in sparse-reward environments such as Acrobot.

Thus, while value-based approaches are highly efficient for environments with clear value signals and limited action spaces, policy-based methods provide a more flexible and often more scalable alternative for a broader range of reinforcement learning tasks.

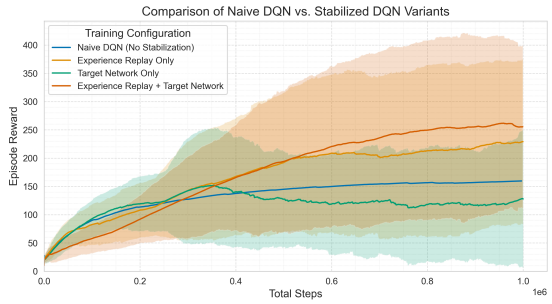

Figure 3. Comparison of Naive DQN vs. Stabilized DQN Variants on CartPole

Figure 4. Comparison of Naive DQN and stabilized DQN variants on the `CartPole-v1` environment. The plot compares four training configurations over 1 million environment steps: (1) Naive DQN without any stabilization, (2) DQN with Experience Replay only, (3) DQN with a Target Network only, and (4) DQN with both Experience Replay and Target Network. The x-axis represents total training steps, while the y-axis shows average episode return. Shaded regions represent one standard deviation over multiple runs. The results highlight the significant benefits of combining both stabilizations — the fully stabilized DQN learns faster, achieves higher rewards, and maintains consistent performance, while the naive version plateaus early with lower returns and higher variance.

4. Conclusion

In this study, we conducted an in-depth exploration of three core policy-based reinforcement learning algorithms—REINFORCE, Actor-Critic (AC), and Advantage Actor-Critic (A2C)—across two benchmark environments: `CartPole-v1` and `Acrobot-v1`. Our experiments demonstrated that A2C consistently outperformed both AC and REINFORCE in terms of learning efficiency and final reward, validating its theoretical advantages such as reduced gradient variance and more stable updates.

While our initial hypothesis—that A2C $>$ AC $>$ REINFORCE—held true overall, the results also highlighted key nuances. In `CartPole-v1`, REINFORCE achieved surprisingly strong performance when tuned appropriately, emphasizing that simple algorithms can still be highly effective in dense-reward environments. On the other hand, REINFORCE struggled to learn in the sparse-reward `Acrobot-v1`, where A2C clearly shined. These outcomes underscore the importance of environment characteristics and careful hyperparameter tuning in reinforcement learning.

Moreover, our ablation studies and architectural choices revealed that learning rate, hidden layer size, and actor-critic separation all significantly impact convergence behavior. These findings reinforce the value of controlled experimentation and rigorous reproducibility in evaluating RL methods.

Overall, this project offered valuable practical insights into policy gradient methods, demonstrating both their strengths and limitations. Future work could extend this foundation by incorporating entropy regularization, exploring continuous control settings, or benchmarking against modern actor-critic variants such as PPO and SAC.

Overall, A2C proved to be the most effective among the tested methods. The Acrobot experiment further demonstrated its advantage in learning under sparse-reward conditions.

All implementation files, including `reinforce.py`, `ac.py`, `a2c.py`, and `models.py`, followed a unified design pattern and are available in the accompanying source code repository.

References

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

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

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT press, 2018.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 2000.

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