Investigating REINFORCE and Actor-Critic Methods in the Cartpole Environment

Pranav Srivastava [1 2]
Yousef Swed [1 2]
Kevin Bretz [1 2]

Abstract

This paper reports the implementation and comparative study of three policy-based reinforcement learning algorithms: REINFORCE, basic Actor-Critic (AC) and Advantage Actor-Critic (A2C) in the Cartpole environment. We provide an explanation of the techniques and experimental design, followed by performance results in terms of learning curves (return vs. environment steps). Our results indicate that using an actor-critic architecture with advantage functions can yield improved stability and faster convergence.

1. Introduction

In recent years, policy-based reinforcement learning methods have gained traction due to their ability to directly optimize the action-selection strategy. Unlike traditional value-based methods such as Q-learning, which estimate the value of state-action pairs, policy-based approaches optimize a policy function to handle high-dimensional or continuous action spaces more naturally (Thomas & Brunskill, 2017). This report studies three such methods: REINFORCE, a pure policy gradient method; a basic Actor-Critic approach that employs a separate value network to stabilize learning; and Advantage Actor-Critic (A2C), which uses an advantage function to reduce variance in policy updates. The primary goal of this report is to provide a detailed theoretical analysis of these methods, accompanied by empirical comparisons conducted in the CartPole environment.

2. Theory

2.1. Vanilla Policy Loss

The REINFORCE algorithm is based on the vanilla policy loss (Plaat, 2022):

$$\mathcal{L}(\theta) = \mathbb{E}_{s\sim D}\left[\mathbb{E}_{a\sim\pi_\theta(a|s)}\left[-Q^{\pi_{\theta_k}}(s,a)\right]\right]$$

This loss updates policy parameters using Monte-Carlo estimates of returns without a value network. In order to understand the loss function, it's important to look at the gradient of the loss $\nabla L(\theta)$ which is used in order to improve the parameters of the network. The gradient of the loss function is equivalent to $\mathbb{E}_{s\sim D}\left[\nabla_\theta \mathbb{E}_{a\sim\pi_\theta(a|s)}\left[-Q^{\pi_{\theta_k}}(s,a)\right]\right]$ where the gradient operator is simply pushed inside the expectation. Focusing on the inner gradient $\nabla_\theta \mathbb{E}_{a\sim\pi_\theta(a|s)}\left[-Q^{\pi_{\theta_k}}(s,a)\right]$ and using the log identity $\nabla \log f(x) = \frac{\nabla f(x)}{f(x)}$ (Mohamed, 2015), it follows that

$$\nabla_\theta \mathbb{E}_{a\sim\pi_\theta(a|s)}\left[-Q^{\pi_{\theta_k}}(s,a)\right] =$$

$$-\mathbb{E}_{a\sim\pi_\theta(a|s)}\left[Q^{\pi_{\theta_k}}(s,a)\nabla_\theta \log \pi_\theta(a|s)\right]$$

This is exactly the policy gradient which inspires the update rule in the REINFORCE algorithm.

2.2. REINFORCE Algorithm

REINFORCE is a Monte-Carlo policy gradient method that uses the full-episode return as an unbiased estimator of $Q(s,a)$. This is due to the fact that REINFORCE is a model-free algorithm that doesn't estimate $Q(s,a)$ explicitly using a neural network (Plaat, 2022). Hence, the next best estimate is to calculate the returns of one full episode and use this as an estimate of the Q-value, leading to the REINFORCE update rule: $\nabla_\theta J(\theta) = \mathbb{E}[G_t \nabla_\theta \log \pi_\theta(a_t|s_t)]$ (Ma, 2020). The algorithm works by generating a full trace, and for each time step t, calculating the return from t until the end of the episode. Next, the parameters are updated by calculating the gradient $\nabla_\theta J$ and multiplying this by the learning rate. This value is added to the existing parameters. The algorithm ends when the gradient converges below a predefined threshold, or until a specified number of episodes or time steps. The convergence of the gradient $\nabla_\theta J$ signifies fewer and fewer changes in the policy every update, and as the gradient of the policy converges to a threshold or zero, this signifies that the parameters have reached a local or global optima. Resultingly, the policy stabilizes with parameters believed to lead to optimal actions.

2.3. Actor-Critic Algorithm

The Actor-Critic algorithm improves the REINFORCE algorithm by providing a value network in order to estimate the

Algorithm 1 Basic Actor-Critic (AC) Algorithm Pseudo-code (Plaat, 2022)

Input: A policy $\pi_\theta(a|s)$, a value function $V_\phi(s)$
An estimation depth n, learning rate α, number of episodes M
Initialization: Randomly initialize θ and ϕ
repeat
for $i \in 1, ..., M$ **do**
  Sample trace $\tau = \{s_0, a_0, r_0, s_1, ..., s_T$ following $\pi_\theta(a|s)\}$
  for $t \in 0, ..., T-1$ **do**
    $\hat{Q}_n(s_t, a_t) = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V_\phi(s_{t+n})$
  end for
end for
$\phi \leftarrow \phi - \alpha \nabla_\phi \sum_t \left(\hat{Q}_n(s_t, a_t) - V_\phi(s_t)\right)^2$
$\theta \leftarrow \theta + \alpha \sum_t \hat{Q}_n(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$
until $\nabla_\theta J(\theta)$ converges below ϵ
return Parameters θ

$V(s)$ rather than relying solely on the Monte Carlo estimate by sampling full episodes.

The algorithm begins by initializing two neural networks, one being the policy as in REINFORCE, and the other a value network to estimate the $Q(s, a)$ values. After sampling a full episode, the $Q(s, a)$ values are estimated for each time step of the episode, where each $Q(s, a)$ value is calculated using the following n steps. This is a key feature of the basic actor-critic algorithm, where instead of estimating the $Q(s, a)$ value for the remainder of the episode, only the next few steps are considered to estimate the value, which has less variance between estimated $Q(s, a)$ values per time step. Contrastingly, REINFORCE has high variance as the $Q(s, a)$ values are estimated by calculating the returns of the full episode, which varies drastically due to the random roll-outs (Plaat, 2022). At the end of each episode, the parameters of the value network are updated by calculating the MSE loss between the estimated $Q(s, a)$ and the outputted $V(s)$ from the value network. As $Q(s, a)$ estimates improve each loop, the difference between $Q(s, a)$ and $V(s)$ decreases, and eventually the parameters ϕ converge. The parameters θ also converge as the gradient of the policy $\nabla_\theta \log \pi_\theta(a_t|s_t)$ converges to 0. The updates of AC are a lot more stable due to the newly introduced value network, combining the strengths of value-based networks (low variance) with the strengths of the policy-based networks (low bias) (Plaat, 2022).

2.4. Conceptual Insights into Policy-Gradient Methods

2.4.1. Q-Learning loss incompatibility

It is not possible to use a Q-Network such as for DQN, as DQN uses the non-differentiable argmax function, resulting in a deterministic policy, and the actor needs gradients to perform backpropagation. (Silver et al., 2014)

2.4.2. Model Interplay

The concept of having two models learn and improve upon each other in a loop has been widely applied across various domains. For instance, in recommender systems, one model, known as the retrieval model, generates recommendations, while another model, the rating model, assigns numerical scores to the generated recommendations. These two models iteratively refine each other by cycling through multiple sequences of retrieval and ranking until the retrieval model achieves the desired performance.

A notable example of this interplay is Neural Collaborative Filtering (NCF), an integrated recommendation model that leverages two complementary neural network branches within a single, end-to-end framework. The first branch, inspired by Generalized Matrix Factorization (GMF), is designed to learn linear interactions between user and item embeddings. In contrast, the second branch employs a Multi-Layer Perceptron (MLP) to capture complex, non-linear relationships between users and items. Rather than operating as two independent models that iteratively refine one another, NCF trains both branches simultaneously using a unified loss function. The outputs of these branches are fused often via concatenation and passed through a final prediction layer, which produces a recommendation score. This joint learning process allows the model to harness the strengths of both linear and non-linear interaction modeling, yielding more robust and expressive representations.

2.5. Advantages vs Q-Values

The traditional Q-Value policy gradient function is defined as:

$$\nabla_\theta \mathbb{E}_{s,a\sim\pi_\theta} \left[\log \pi_\theta(a|s) \cdot Q(s, a)\right]$$

In theory, a significant performance increase can be achieved by replacing the Q-Values with the so-called Advantage:

$$\nabla_\theta \mathbb{E}_{s,a\sim\pi_\theta} \left[\log \pi_\theta(a|s) \cdot A(s, a)\right]$$

wherein the advantage function is defined as:

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

$V^\pi(s)$ returns the average value of taking an action, given a state s. Using the advantage normalizes the value of each possible action in relation to the average action value in the current state (Plaat, 2022). Working with relative values

Algorithm 2 Advantage Actor-Critic (A2C) Algorithm Pseudo-code

Input: A policy $\pi_\theta(a|s)$, a value function $V_\phi(s)$
An estimation depth n, learning rate α, number of episodes M
Initialization: Randomly initialize θ and ϕ
repeat
for $i \in 1, ..., M$ **do**
Sample trace $\tau = \{s_0, a_0, r_0, s_1, ..., s_T$ following $\pi_\theta(a|s)\}$
for $t \in 0, ..., T-1$ **do**
$\hat{Q}_n(s_t, a_t) = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V_\phi(s_{t+n})$
$\hat{A}_n(s_t, a_t) = \hat{Q}_n(s_t, a_t) - V_\phi(s_t)$
end for
end for
$\phi \leftarrow \phi - \alpha \nabla_\phi \sum_t \left(\hat{A}_n(s_t, a_t)\right)^2$
$\theta \leftarrow \theta + \alpha \sum_t [\hat{A}_n(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t \mid s_t)]$
until $\nabla_\theta J(\theta)$ converges below ϵ
return Parameters θ

allows the policy to focus solely on actions that are substantially better than others, and neutralizes the effect that mediocre or bad actions would have on the policy simply because the Q-Value is positive.

2.6. Advantage Actor-Critic (A2C)

A2C improves the AC algorithm by introducing the advantage function $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$, as seen in **Algorithm 2**. As mentioned in section **2.4** the advantage allows the policy to focus solely on actions that are substantially better than others, which reduces the variance in returns, stabilizing the AC algorithm.

2.7. A2C with entropy - BONUS

In policy based reinforcement learning algorithms, entropy can be artificially added to the policy loss to encourage taking actions that may not seem like the immediate optimal, but could still lead to improvements. In essence, adding an entropy bonus keeps the policy from getting stuck in a local minima and collapsing to a deterministic one too early. (Liu et al., 2019) This can be thought of as the analog to epsilon from the DQN algorithm, balancing exploration and exploitation, but for a stochastic policy.

For the bonus agent, entropy was implemented to the basic A2C algorithm. The policy loss function with added entropy bonus is defined as

$$L_{\text{actor}} = -\mathbb{E}\left[\log \pi(a \mid s) \cdot A(s, a)\right] + \beta \cdot \mathcal{H}(\pi(\cdot \mid s))$$

where

$$\mathcal{H}(\pi(\cdot|s)) = \frac{1}{2}\log(2\pi e\sigma^2) = Entropy$$

and

$$\beta = \text{Entropy Coefficient}$$

By precisely tuning the entropy coefficient, better results can be achieved and avoid getting stuck in local minima.

3. Experiments

3.1. Experimental Setup

The four algorithms (REINFORCE, AC, A2C, and A2C with entropy) were implemented using the same neural network architecture for the policy. The experiments were conducted in the Cartpole environment. Each algorithm was ran 5 times, each with 1 million environment steps. The figures include the averages of the runs. Further key details include:

- **Environment:** OpenAI Gym Cartpole.
- **Hyperparameters:** Learning rates, discount factors, and network architectures were tuned based on A1.
  - **Hidden Size:** 128
  - **Learning Rate:** 0.0005
  - **Discount Factor (gamma):** 0.99
  - **Hidden Size:** 128
  - **Total Steps:** 1000000
  - **Number of runs:** 5
  - **Seed:** 42

  For AC, A2C, and A2C with entropy, a further hyperparameter **n-step:** 1 was used.
- **Metrics:** Learning curves (return vs. environment steps) were recorded.

4. Results

We obtain the results of the experiments by comparing four policies-gradient algorithms on CartPole-v1 environment. For the experiments, performance graphs are provided in terms of smoothed average episode reward per 50 environment steps. The policies are color-coded as follows: REINFORCE (blue), AC (red), A2C (orange), and A2C with Entropy (green).

The resulting performance curve for **REINFORCE** (blue) demonstrates relatively slow convergence with large variance, ultimately levelling off to an average reward in the range of 200–250, as seen in FIgure 1.

As explained in section **2.3**, the baseline **AC** algorithm incorporates an additional value function estimator to reduce the variance in the gradient steps. The AC variant (red line) demonstrates more rapid early learning, with average reward leveling out at approximately 250–300 early on. The AC method continues to fluctuate, however, and never quite maintains above-average rewards regularly, relative to **REINFORCE**.

Advantage Actor-Critic (A2C) improves the results of **AC** by including an advantage function, in addition to further optimizing the gradient updates using bootstrapping. The estimated advantage reduces the variance even further, resulting in smoother and more stable learning dynamics. This improvement can be observed in the A2C variant (orange curve), where the average reward steadily moves towards the mid-300 level, with some residual variance remaining.

Finally, the **A2C algorithm with entropy** regularization contains another entropy bonus promoting exploration. This process avoids premature convergence to suboptimal policies. Therefore, the A2C with Entropy (green curve) converges fastest among the four algorithms, and it achieves and sustains near-optimal performance within the range of 450–500.

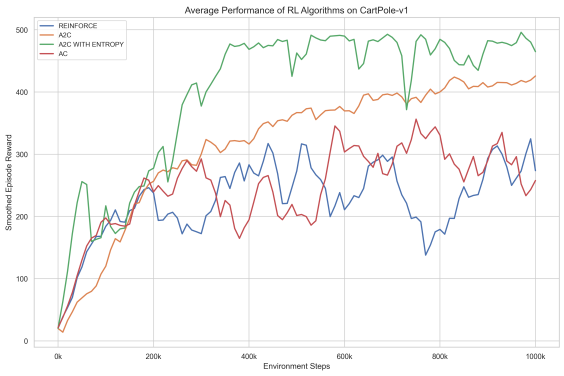

Figure 1. Average Performance (Smoothed Episode Reward) vs. Environment Steps on CartPole-v1 for REINFORCE, AC, A2C, and A2C with entropy.

4.1. Comparing Value-Based methods with Policy-Based methods

Assignment 1 used Deep Q-Learning models on CartPole, and the results using DQN with a target network and an experience replay yielded similar results to a more refined policy-based method (A2C) as seen in Figure 2. Policy-based methods thrive with continuous action spaces where the policy function represents a probability distribution over possible actions. However, environments like CartPole have a small, discrete action space, either moving left or right. This allows value-based methods such as DQN to learn rapidly through efficient Q-value estimation. In such environments, the flexibility of policy-based environments offers only a limited advantage. In an environment with a large action space - such as moving a robotic arm - the strengths of policy-based methods in modeling complex, stochastic policies would be better showcased.

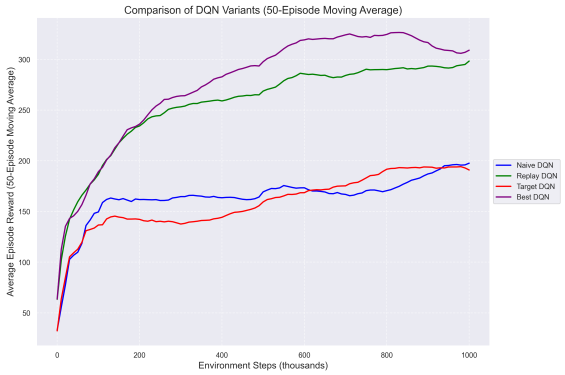

Figure 2. Average Performance (Smoothed Episode Reward) vs. Environment Steps on CartPole-v1 for Naive DQN, Replay DQN, Target DQN, and DQN with TN + ER.

5. Discussion

The variations in performance that are observed in the results of the experiments are caused by the mechanisms used by each method to estimate and utilize the expected return $Q(s, a)$.

REINFORCE relies solely on the full-episode return, which gives unbiased but high-variance estimates. As a result, learning is slow-converging and low-average level of performance. By contrast, the **AC** approach uses a critic that estimates a value function and thus reduces the variance of the gradient estimates. This enhancement allows for faster initial improvement, although the absence of advantage estimation makes the updates noisy to some extent.

A2C also enhances learning by computing an advantage function to estimate the relative value of taking an action in relation to the baseline value. This provides more stable updates as the advantage further improves the selection of optimal actions by encouraging actions that are better than the baseline value. The orange line indicates that there is better performance up to the mid-300 level of rewards when using an advantage estimator, though this option does not achieve full value of stabilization due to the lack of more exploration assistance.

The inclusion of an entropy bonus in **A2C with Entropy** is to shift the policy so that it utilizes more diverse actions so that premature convergence to sub-optimal policies cannot occur. This regularization achieves more substantial updates and, as indicated in the green line, reaches highly rapid

convergence and stable performance near the top reward range (450–500). This enhancement clearly demonstrates the benefit of having both variance reduction by using a critic and exploration rewards with an entropy term.

6. Conclusion

This report provides a comparison of four policy-gradient methods in the CartPole-v1 environment, i.e., REINFORCE, Actor-Critic (AC), Advantage Actor-Critic (A2C), and Entropy-regularized A2C. REINFORCE, using full-episode returns, is simple but suffers from high variance and slow convergence. The basic AC improves upon REINFORCE by introducing a value network and n-step bootstrapping to reduce variance. Although, AC still suffers from variance due to estimating $Q(s, a)$. The A2C algorithm, with the inclusion of an advantage function, depicts improvement in both performance and stability as shown by a smooth rise to an average reward in the mid-300s. The best performance overall is achieved via A2C with Entropy, where the entropy bonus pushes exploration and stabilizes the update process to lead to convergence at values near the optimum rewards (450–500).

These findings underscore the critical role of exploration and variance reduction in policy gradient methods. Hybrid approaches that combine off-policy learning techniques or adaptive entropy regularization might be investigated further to enhance the robustness and efficiency of these algorithms.

References

Liu, J., Gu, X., and Liu, S. Policy optimization reinforcement learning with entropy regularization. *arXiv preprint arXiv:1912.01557*, 2019. URL https://arxiv.org/abs/1912.01557.

Ma, T. Cs229 lecture notes: Reinforcement learning. https://cs229.stanford.edu/notes2020fall/notes2020fall/cs229-notes14.pdf, 2020. Stanford University, Accessed: 2025-04-15.

Mohamed, S. Machine learning trick of the day (5): Log-derivative trick, 2015. URL https://blog.shakirm.com/2015/11/machine-learning-trick-of-the-day-5-log- Accessed: 2025-04-15.

Plaat, A. *Deep Reinforcement Learning*. Springer Nature Singapore, 2022. ISBN 9789811906381. doi: 10.1007/978-981-19-0638-1. URL http://dx.doi.org/10.1007/978-981-19-0638-1.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 387–395, 2014.

Thomas, P. S. and Brunskill, E. Policy gradient methods for reinforcement learning with function approximation and action-dependent baselines, 2017. URL https://arxiv.org/abs/1706.06643.

7. Appendix

7.1. REINFORCE Performance

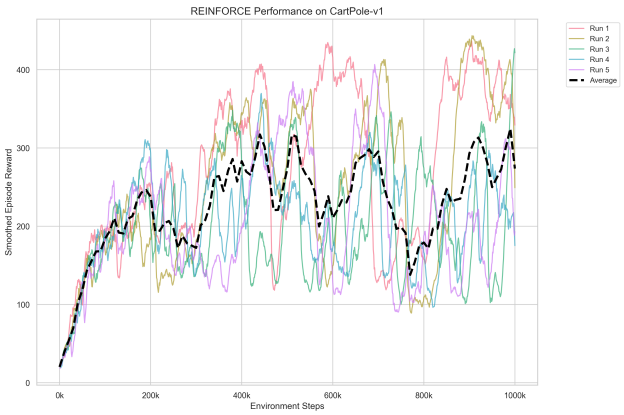

Figure 3. REINFORCE runs with averaged run

7.2. AC Performance

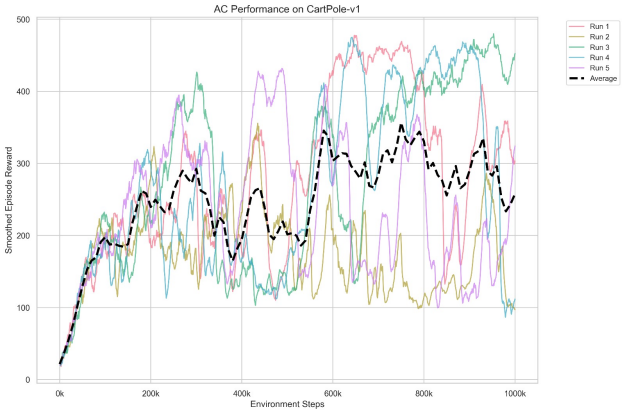

Figure 4. AC runs with averaged run

7.3. A2C Performance

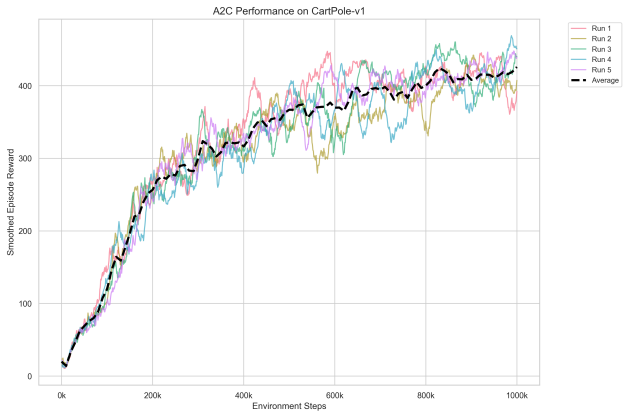

Figure 5. A2C runs with averaged run

7.4. A2C with Entropy Performance

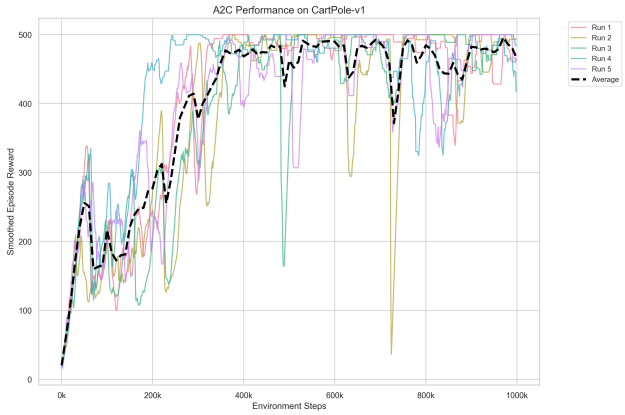

Figure 6. A2C with entropy runs with averaged run