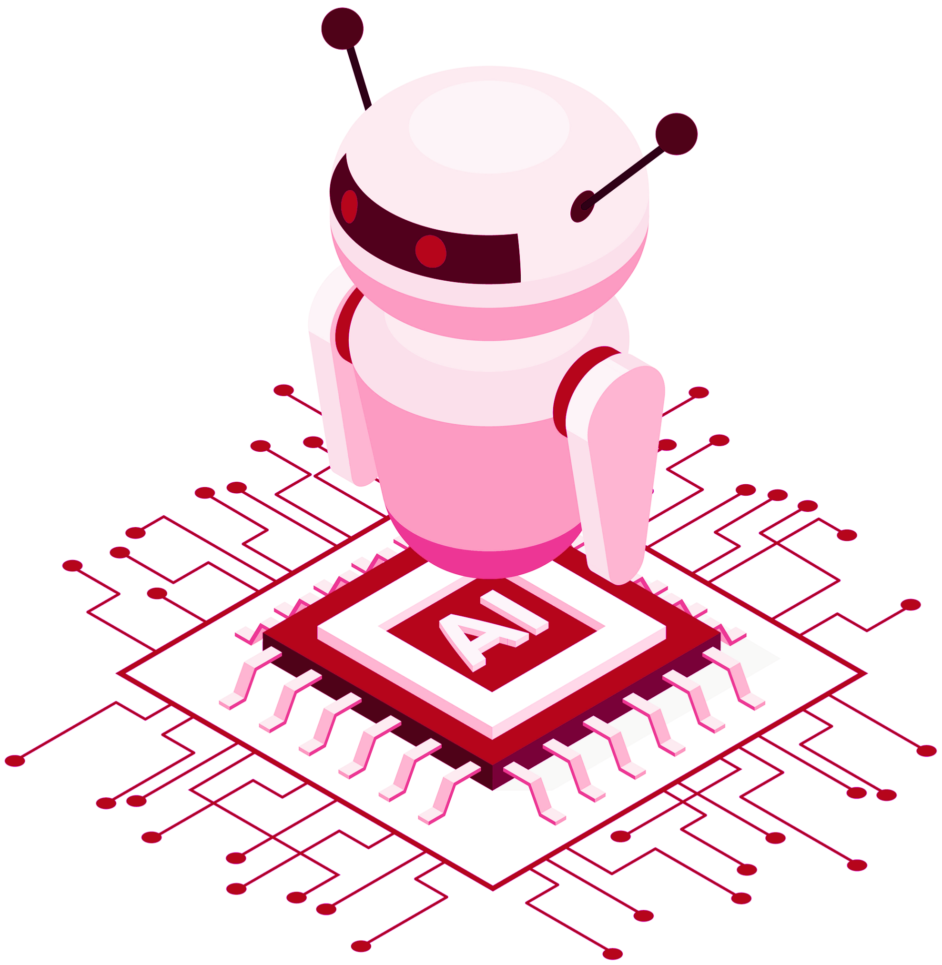

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 3:

Policy-Based Methods

By:

[Asemaneh Nafe]

[400105285]

Spring 2025

Contents

Grading

The grading will be based on the following criteria, with a total of 100 points:

Task	**Points**
Task 1: Policy Search: REINFORCE vs. GA	20
Task 2: REINFORCE: Baseline vs. No Baseline	25
Task 3: REINFORCE in a continuous action space	20
Task 4:Policy Gradient Drawbacks	25
Clarity and Quality of Code	5
Clarity and Quality of Report	5
Bonus 1: Writing your report in Latex	10

1 Task 1: Policy Search: REINFORCE vs. GA [20]

1.1 Question 1:

How do these two methods differ in terms of their effectiveness for solving reinforcement learning tasks? REINFORCE and Genetic Algorithms (GA) take very different approaches to solving reinforcement learning problems. REINFORCE updates a policy using gradient-based optimization, continuously improving it based on expected rewards. In contrast, GA evolves a population of solutions using selection, crossover, and mutation, which helps in highly stochastic or non-differentiable environments. While REINFORCE is generally more efficient in smooth, differentiable policy spaces, GA is better at exploring a broader range of solutions. However, GA may require significantly more iterations to find an optimal policy.

1.2 Question 2:

Discuss the key differences in their **performance**, **convergence rates**, and **stability**. REINFORCE typically converges faster because it follows gradient-based updates, but it is sensitive to hyperparameters like learning rate and can suffer from high variance. GA, on the other hand, is more robust to noisy rewards and doesn't rely on gradients, making it stable in challenging environments, though it converges much slower. While REINFORCE can efficiently exploit good policies once they are found, GA relies more on random mutations and selection pressure, which makes fine-tuning less precise. GA may also get stuck in local optima if diversity is lost in the population. In contrast, REINFORCE can struggle in environments with sparse rewards where GA's diverse exploration helps.

1.3 Question 3:

Additionally, explore how each method handles exploration and exploitation, and suggest situations where one might be preferred over the other. REINFORCE balances exploration and exploitation using stochastic policies and entropy regularization, making it efficient in structured, differentiable environments. GA naturally explores a wide range of solutions through random mutations but lacks a direct exploitation mechanism like policy gradient updates. If you need a method that quickly fine-tunes a policy based on known reward signals, REINFORCE is the better choice. However, if your problem involves sparse rewards, high randomness, or no clear gradients (like some evolutionary robotics tasks), GA is more suitable. In practical applications, REINFORCE shines in tasks like robotics and finance, while GA is often useful in complex search spaces or game AI. In GAs, exploitation happens implicitly by selecting individuals with the highest fitness scores (reward-based selection). The best individuals are more likely to pass their genetic material (weights, policies) to the next generation. However, unlike policy gradient methods, GA does not directly update policies in the direction of the reward gradient—instead, it relies on crossover and mutation to produce better policies over time.

In contrast, REINFORCE explicitly adjusts the policy parameters in the direction that increases expected rewards using gradient ascent. This targeted adjustment makes REINFORCE more efficient at local refinement, while GA relies on randomness and selection pressure, which can be slower in fine-tuning.

2 Task 2: REINFORCE: Baseline vs. No Baseline [25]

2.1 Question 1:

How are the observation and action spaces defined in the CartPole environment?

How are the observation and action spaces defined in the CartPole environment?

The observation space in CartPole-v1 consists of a continuous 4-dimensional vector representing:

Cart position

Cart velocity

Pole angle

Pole angular velocity

The action space is discrete, meaning there are only two possible actions:

0: Move the cart to the left

1: Move the cart to the right

2.2 Question 2:

What is the role of the discount factor (γ) in reinforcement learning, and what happens when γ=0 or γ=1?

The discount factor(λ) determines how future rewards contribute to the current value estimation. When (λ)= 1: The agent considers all future rewards equally, leading to long-term planning. When (λ) = 0: The agent only values immediate rewards, effectively making it a greedy strategy. Typical range: 0 $<$ (λ) $<$ 1 ensures that future rewards have a diminishing impact, balancing short-term and long-term rewards.

2.3 Question 3:

Why is a baseline introduced in the REINFORCE algorithm, and how does it contribute to training stability?

A baseline (typically a value function) is introduced to reduce variance in policy gradient updates without introducing bias. The advantage function $A(s_t, a_t) = G_t - V(s_t)$ helps the agent determine whether an action is better or worse than the expected return from that state. This leads to:

More stable training (reducing high variance in updates). Faster convergence since updates are less noisy. Better performance as the policy optimization is more structured.

2.4 Question 4:

What are the primary challenges associated with policy gradient methods like REINFORCE?

High Variance: Policy gradients often have high variance, leading to unstable learning and slow convergence.

Sample Inefficiency: REINFORCE discards episode data after each update, making it less data-efficient than methods like Q-learning or PPO.

Delayed Updates:Unlike Q-learning, which updates per step, REINFORCE updates parameters only after full episodes, leading to slower learning.

Credit Assignment Problem: Assigning credit to specific actions over long trajectories is challenging since updates are made at the end of episodes.

Lack of Off-Policy Learning: REINFORCE is an on-policy algorithm, meaning it cannot reuse past experiences to improve efficiency, unlike off-policy methods like DQN.

2.5 Question 5:

Based on the results, how does REINFORCE with a baseline compare to REINFORCE without a baseline in terms of performance? REINFORCE with a baseline generally performs better than REINFORCE without

a baseline because it reduces variance in gradient updates. The baseline (value function) helps stabilize training by subtracting an estimate of expected return from actual returns. This prevents the agent from making overly large updates based on high-variance returns. As a result, learning is smoother and converges faster. Without a baseline, policy updates can fluctuate more, leading to instability.

2.6 Question 6:

Explain how variance affects policy gradient methods, particularly in the context of estimating gradients from sampled trajectories. Variance in policy gradient methods refers to fluctuations in gradient estimates due to randomness in sampled trajectories. High variance makes learning unstable, causing inconsistent updates that slow down convergence. Since REINFORCE relies on Monte Carlo sampling, it suffers from noisy gradients, making optimization difficult. A baseline helps by reducing variance without introducing bias, leading to more reliable updates. Lower variance means the agent learns faster and more effectively.

3 Task 3: REINFORCE in a continuous action space [20]

3.1 Question 1:

How are the observation and action spaces defined in the MountainCarContinuous environment?

Observation Space: Box[-1.2 -0.07] x Box[0.6 0.07]

A 2D space with: Position of the car in [-1.2 -0.07] Velocity in [0.6 0.07]

Action Space: Box(-1.0, 1.0) A continuous scalar force applied to the car within [-1, 1].

3.2 Question 2:

How could an agent reach the goal in the MountainCarContinuous environment while using the least amount of energy? Explain a scenario describing the agent's behavior during an episode with most optimal policy.

The car must build momentum to reach the flag at $x \geq 0.45$. Optimal Policy: Instead of immediately applying max force, the agent should swing back and forth. It should apply controlled force at the right moments to amplify oscillations. This minimizes total energy spent while still reaching the goal efficiently. for example in our last simulation you can see that car do not spend the maximum energy in the first step it will go frward firstly and then will go back and will serve minimum energy.

3.3 Question 3:

What strategies can be employed to reduce catastrophic forgetting in continuous action space environments like MountainCarContinuous?

(Hint: experience replay or target networks) Catastrophic forgetting happens when a reinforcement learning

agent forgets previously learned behaviors as it continues training. In continuous action space environments like MountainCarContinuous, this can be a major issue, making it difficult for the agent to retain useful strategies. One effective way to tackle this problem is through experience replay, where past experiences are stored in a buffer and sampled randomly during training. This breaks the correlation between consecutive experiences and helps stabilize learning. Another useful approach is using target networks, where a separate, slowly updated copy of the network is maintained to prevent drastic updates that could destabilize learning. Additionally, methods like progressive neural networks, elastic weight consolidation, and replay-based meta-learning can help by preserving important knowledge while allowing the agent to learn new strategies effectively. In practice, experience replay and target networks are among the most commonly used techniques to ensure the agent retains useful knowledge while continuously improving.

4 Task 4: Policy Gradient Drawbacks [25]

4.1 Question 1:

Which algorithm performs better in the Frozen Lake environment? Why?
Compare the performance of Deep Q-Network (DQN) and Policy Gradient (REINFORCE) in terms of training stability, convergence speed, and overall success rate. Based on your observations, which algorithm achieves better results in this environment? Between Deep Q-Network (DQN) and Policy Gradient (REINFORCE), DQN generally performs better in the Frozen Lake environment. The main reason is that DQN is more stable because it uses experience replay and bootstrapping, which help smooth out learning and make updates more reliable. It also learns faster since it can reuse past experiences rather than relying only on new ones.

REINFORCE, on the other hand, tends to be more unstable because it directly optimizes the policy using Monte Carlo updates. Since it only updates based on complete episodes, the learning process can be slower and more unpredictable. This makes it harder for the agent to efficiently learn the best way to reach the goal, especially in an environment like Frozen Lake, where rewards are sparse, and mistakes can be costly. Sparse reward environments, like Frozen Lake, pose a significant challenge for the REINFORCE algorithm because it relies on Monte Carlo updates. This means that the agent only receives feedback at the end of an episode, making it difficult to determine which specific actions contributed to success or failure. Since rewards are only given when the agent reaches the goal (or falls into a hole), most episodes provide little to no learning signal.

In such environments, REINFORCE struggles to assign proper credit to the right actions, leading to high variance in updates. The lack of intermediate rewards also makes it hard for the agent to refine its strategy over time. Instead, it may take a long time to stumble upon a successful trajectory and even longer to generalize that experience into a reliable policy.

DQN, in contrast, can propagate reward signals backward through its value function, allowing it to learn from partial progress rather than waiting for entire episodes to complete. This makes it much more sample-efficient in sparse environments.

4.2 Question 2:

What challenges does the Frozen Lake environment introduce for reinforcement learning?
Explain the specific difficulties that arise in this environment. How do these challenges affect the learning process for both DQN and Policy Gradient methods? The Frozen Lake environment introduces several challenges for reinforcement learning, making it difficult for both DQN and Policy Gradient methods to learn efficiently. One of the biggest challenges is its sparse reward structure—agents only receive a reward upon reaching the goal, with no intermediate feedback to indicate progress. This makes it hard for algorithms like REINFORCE, which rely on Monte Carlo updates, to assign credit to the right actions, leading to slow and unstable learning.

Another challenge is the stochastic nature of movement when the environment is slippery. Actions don't always lead to the expected outcome, making it harder for the agent to learn a reliable policy. This randomness affects both DQN and REINFORCE, as it increases uncertainty in state transitions and makes it difficult to determine whether an action was actually beneficial.

For DQN, the small state space can be deceptive. While it may seem simple, the fact that states have long-term dependencies (where early moves impact the final outcome) makes learning effective Q-values difficult. Additionally, since rewards are sparse, there is little useful information for Q-learning updates, leading to slow convergence.

For Policy Gradient methods like REINFORCE, the lack of frequent rewards means that gradient updates have high variance, making learning highly unstable. Without intermediate rewards, the agent struggles to explore efficiently, often requiring many episodes to discover a successful trajectory.

Overall, Frozen Lake is challenging because of its delayed rewards, stochastic transitions, and long-term dependencies, all of which make it difficult for both DQN and Policy Gradient methods to learn effectively. However, DQN has an advantage in this environment due to its ability to estimate value functions and propagate reward information more efficiently.

4.3 Question 3:

For environments with unlimited interactions and low-cost sampling, which algorithm is more suitable?
In scenarios where the agent can sample an unlimited number of interactions without computational constraints, which approach—DQN or Policy Gradient—is more advantageous? Consider factors such as sample efficiency, function approximation, and stability of learning.

In environments where the agent can sample an unlimited number of interactions without computational constraints, Policy Gradient methods (like REINFORCE) are often more suitable than DQN.

DQN is generally more sample-efficient because it reuses past experiences through replay buffers, but it struggles with function approximation errors, especially when using deep networks. It also requires discretization of action spaces, which can be impractical in many real-world applications. Additionally, DQN's learning process can be unstable due to issues like overestimation bias and catastrophic forgetting.

Policy Gradient methods, on the other hand, directly optimize the policy and do not rely on value approximation for decision-making. This makes them well-suited for environments with high-dimensional or continuous action spaces. While Policy Gradient methods are less sample-efficient, they offer greater stability in learning because they update policies based on actual sampled trajectories rather than relying on bootstrapped value estimates. This stability is particularly beneficial when training in environments with complex dynamics or stochastic transitions.

Another key advantage of Policy Gradient methods is their ability to maintain stochastic policies throughout training, which improves exploration. In contrast, DQN tends to converge to deterministic policies, which can lead to poor exploration and suboptimal performance in some environments.

Given unlimited interactions and no computational constraints, Policy Gradient methods are the better choice due to their stability of learning, flexibility in handling continuous action spaces, and ability to optimize stochastic policies effectively

References

[1] Cover image designed by freepik. Available: https://www.freepik.com/free-vector/cute-artificial-intelligence-robot-isometric-icon_16717130.htm

[2] Policy Search. Available: https://amfarahmand.github.io/IntroRL/lectures/lec06.pdf

[3] CartPole environment from OpenAI Gym. Available: https://www.gymlibrary.dev/environments/classic_control/cart_pole/

[4] Mountain Car Continuous environment from OpenAI Gym. Available: https://www.gymlibrary.dev/environments/classic_control/cart_pole/

[5] FrozenLake environment from OpenAI Gym. Available: https://www.gymlibrary.dev/environments/toy_text/frozen_lake/