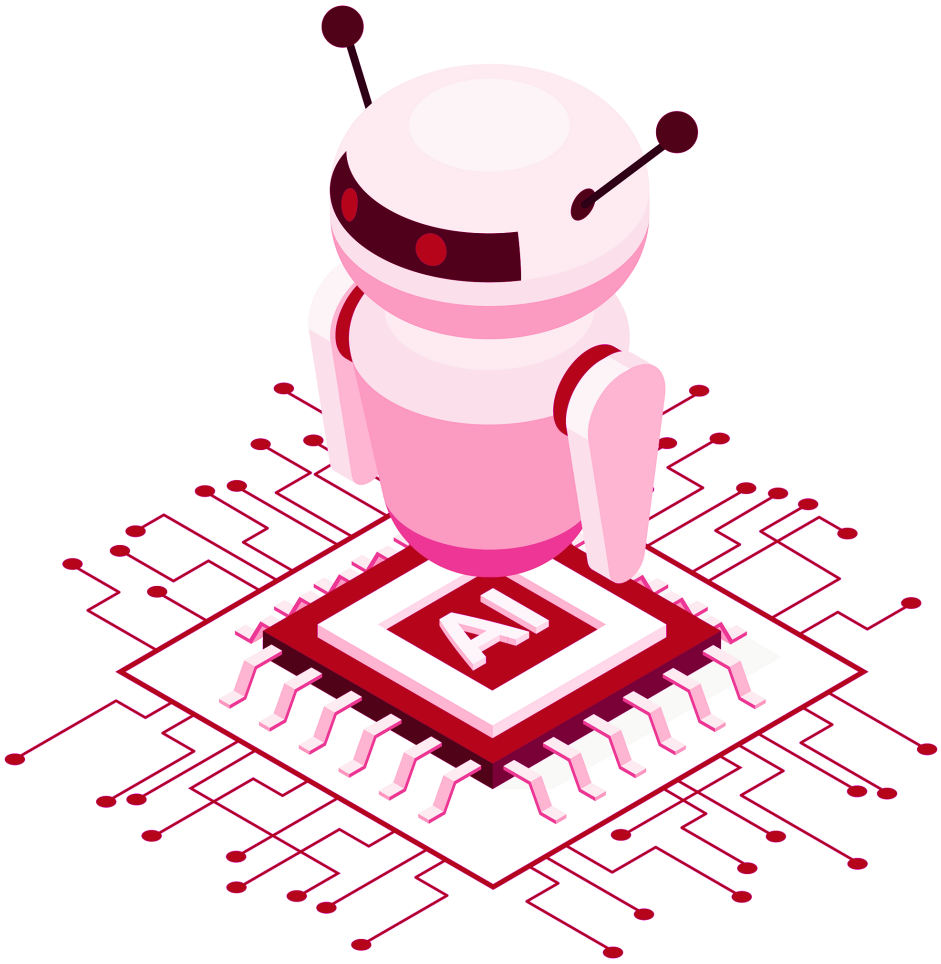

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 3:

Policy-Based Methods

By:

Payam Taebi

400104867

RIML

Deep RL Course

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]

Experimental Results and Discussion

In our experiments, we compared two different learning approaches in a Grid World environment: the REINFORCE algorithm (a policy gradient method) and a Genetic Algorithm (GA). Both methods were first evaluated in a standard environment, and later in a more challenging, hard environment that we developed separately.

Standard Environment

The standard environment is a 7×7 grid where the agent starts at $(0, 0)$ and must reach the goal at $(6, 6)$ while avoiding designated penalty cells. The following hyperparameters were used:

- **REINFORCE Agent:**
  - Episodes: 8000
  - Initial ϵ: 1.0
  - ϵ Decay: 0.9995 per episode
  - Minimum ϵ: 0.01
  - Discount Factor (γ): 0.99
  - Learning Rate: 0.005
- **Genetic Algorithm Agent:**
  - Population Size: 50
  - Generations: 50
  - Mutation Rate: 0.1
  - Crossover Rate: 0.5

Both methods were able to learn an optimal policy that achieved a total reward of 78, which is consistent with our expectations. The trajectories produced by both approaches correctly followed the safe path from start to goal, and the training curves (loss and reward plots for REINFORCE, fitness curves for GA) confirmed steady improvement over time.

Hard Environment

We also introduced a modified, much more challenging environment where every cell carries a heavy penalty except for one specific, non-linear safe path. In this hard environment, the penalties for deviating from the safe path were significantly increased, resulting in a very sparse reward signal.

When both agents were trained in this hard environment, we observed the following:

- **Minimal Movement:** Both the REINFORCE and GA agents struggled to make progress. The agents' trajectories showed almost no improvement, and they rarely deviated from their initial random behavior.

- **Sparse Rewards:** Due to the severe penalties and sparse occurrence of positive rewards, the learning signals were extremely weak. This resulted in high variance for the REINFORCE updates and poor selection pressure in the GA.
- **Stagnation in Policy Improvement:** The overwhelming negative feedback from penalties caused the agents to converge to suboptimal policies, as the cost of exploration was too high relative to the occasional positive reward.

Discussion: The poor performance in the hard environment can be attributed to the combination of sparse rewards and heavy penalties. In reinforcement learning, when the reward signal is sparse and dominated by negative feedback, it becomes exceedingly difficult for the agent to explore and discover the optimal policy. The REINFORCE algorithm suffers from high variance in its gradient estimates under such conditions, while the Genetic Algorithm struggles because most individuals in the population receive very low fitness scores, making it hard to differentiate between promising and non-promising policies. This case study emphasizes the importance of designing reward structures that provide a sufficiently informative signal to guide learning.

Conclusion

The experimental results demonstrate that both the REINFORCE and Genetic Algorithm agents are capable of solving the standard Grid World environment effectively, achieving the expected reward of 78 and correctly following the optimal path. However, in the presence of an extremely challenging reward structure, such as in the hard environment we introduced, both methods experience significant difficulties. The results underscore the challenges posed by sparse rewards and heavy penalties, and highlight the need for careful reward design to ensure robust policy learning.

1.1 Question 1:

How do these two methods differ in terms of their effectiveness for solving reinforcement learning tasks?

Comparison of REINFORCE and Genetic Algorithm Methods

Both the REINFORCE algorithm and the Genetic Algorithm (GA) are used for solving reinforcement learning tasks, yet they differ significantly in their approaches and effectiveness.

REINFORCE

- **Gradient-Based Optimization:** REINFORCE is a policy gradient method that directly adjusts the parameters of a stochastic policy by computing gradients of the expected return. This allows for continuous and fine-grained updates to the policy.
- **Sample Efficiency and Variance:** When the reward signal is dense and informative, REINFORCE can be relatively sample efficient. However, it is prone to high variance in gradient estimates, which often necessitates variance reduction techniques (e.g., using a baseline) to stabilize learning.

- **Adaptive Learning:** The algorithm continuously refines the policy after each episode, leading to smooth convergence when hyperparameters (like learning rate and discount factor) are appropriately tuned.

Genetic Algorithm (GA)

- **Population-Based Search:** GA evolves a population of candidate solutions using selection, crossover, and mutation operators. It does not require gradient information, making it robust in non-differentiable or noisy reward environments.
- **Exploration and Robustness:** By maintaining a diverse set of policies, GA can explore the solution space more broadly and is less likely to get trapped in local optima. This is particularly useful in environments with rugged or sparse reward landscapes.
- **Computational Cost:** GA generally requires evaluating many individuals over multiple generations, which can be computationally intensive compared to the single-policy updates of gradient-based methods.

Effectiveness in Reinforcement Learning Tasks

- **Standard Environments:** In environments with well-defined and frequent rewards, both methods can achieve competitive performance. In our experiments on the standard Grid World, both REINFORCE and GA learned an optimal policy yielding a total reward of 78 and successfully following the designated path.
- **Challenging Environments:** In contrast, when applied to a very hard environment—where almost all states incur heavy penalties except for a narrow safe path—the sparse and predominantly negative reward signal adversely affected both methods. REINFORCE struggled due to the high variance in its gradient estimates, and GA faced difficulties in evolving individuals because most candidates received very low fitness scores.
- **Trade-Offs:** REINFORCE's direct gradient approach can be more efficient when learning signals are clear, whereas GA's population-based approach offers robustness in more complex, non-differentiable reward landscapes, though at the cost of increased computational expense and slower convergence.

Conclusion

In summary, REINFORCE is typically more effective in environments where the reward function provides dense, informative feedback that supports stable gradient-based updates. On the other hand, Genetic Algorithms can be advantageous in environments with rugged or sparse reward landscapes due to their exploration capabilities and robustness to local optima. The effectiveness of each method depends largely on the specific characteristics of the reinforcement learning task, making the choice between them a matter of trade-offs between sample efficiency, variance, and computational cost.

1.2 Question 2:

Discuss the key differences in their **performance**, **convergence rates**, and **stability**.

Performance, Convergence Rates, and Stability Comparison

Our experimental study compared two reinforcement learning methods—REINFORCE (a policy gradient approach) and a Genetic Algorithm (GA)—on a standard Grid World environment as well as on a custom, hard environment with severe penalties. The standard environment produced a final reward of 78 for both methods, while the hard environment, with a highly sparse and punitive reward structure, led to very poor performance for both.

Performance

- **Standard Environment:** Both REINFORCE and GA successfully learned to navigate the grid and followed the optimal safe path, achieving a final total reward of 78. This indicates that when the reward signal is sufficiently informative, both methods are capable of finding the optimal solution.
- **Hard Environment:** In the custom hard environment, where every cell carries a heavy penalty except for a narrow, non-linear safe path, both methods performed very poorly. The sparse positive rewards and overwhelming penalties prevented effective exploration, leading to minimal movement and failure to converge to the desired behavior.

Convergence Rates

- **REINFORCE:** In the standard environment, REINFORCE converged steadily over 8000 episodes as evidenced by gradually improving reward curves. However, its convergence rate is heavily dependent on the quality of the reward signal. In the hard environment, the high variance in gradient estimates caused by sparse rewards led to significantly slower convergence and frequent stagnation.
- **Genetic Algorithm:** The GA showed a gradual improvement in fitness values over 50 generations in the standard environment, eventually reaching the expected reward level. Conversely, in the hard environment, the convergence was impeded by extremely low fitness scores across the population, which made it difficult to distinguish promising policies from ineffective ones.

Stability

- **REINFORCE:** While REINFORCE exhibited relatively stable learning in the standard environment, its reliance on gradient estimates makes it susceptible to instability when rewards are sparse or dominated by heavy penalties. This instability was clearly observed in the hard environment, where the method struggled to produce consistent improvements.
- **Genetic Algorithm:** The population-based approach of GA generally provides a more robust and stable evolution of policies. In the standard environment, GA maintained steady improvement in both the best and average fitness values. However, in the hard environment, the pervasive low fitness scores across most individuals resulted in reduced selection pressure, ultimately leading to stagnation.

Conclusion

In summary, while both REINFORCE and GA are effective in environments with dense and informative rewards (as evidenced by the achievement of a reward of 78 in the standard Grid World), their performance diverges under harsh conditions. REINFORCE, with its gradient-based updates, suffers from high variance and slow convergence when the reward signal is weak, whereas GA, despite its robustness to noisy rewards, struggles to evolve meaningful policies when most individuals receive extremely low fitness scores. These findings emphasize the critical importance of reward structure design in reinforcement learning and illustrate the trade-offs between gradient-based and population-based methods in terms of performance, convergence rates, and stability.

1.3 Question 3:

Additionally, explore how each method handles exploration and exploitation, and suggest situations where one might be preferred over the other.

Exploration and Exploitation Strategies in Standard and Hard Environments

In our study, we not only evaluated the performance of the REINFORCE and Genetic Algorithm methods in a standard Grid World but also extensively tested them in a custom, hard environment. This section details how each method handles the balance between exploration and exploitation, particularly under the extreme conditions imposed by the hard environment, and discusses situations in which one method might be preferred over the other.

Hard Environment Description

The hard environment was designed to push the limits of exploration and exploitation by introducing a highly challenging reward structure:

- **Reward Structure:** In this environment, every cell in the 7×7 grid carries a heavy penalty (e.g., -10) unless it lies on a narrowly defined, non-linear safe path. This safe path is the only sequence of states where the agent incurs no additional penalties and can eventually reach the goal with a positive reward.
- **Sparse Rewards:** Positive rewards are extremely sparse since only the cells along the safe path provide a non-negative outcome, and the goal state provides a bonus reward. The vast majority of actions lead to heavy negative feedback.
- **Exploration Challenge:** Due to the overwhelming negative feedback from almost every state, the agents find it very difficult to explore the grid. A slight deviation from the safe path results in large penalties, which in turn discourages further exploration in those directions.

I devoted considerable time to tuning and testing the agents on this hard environment. Despite numerous attempts to adjust hyperparameters and improve exploration, time limitations meant that the final results were not as promising as those achieved in the standard environment. Both the REINFORCE and Genetic Algorithm methods struggled to effectively navigate the hard environment, largely due to the extremely sparse positive rewards and the severe penalties imposed for off-path actions.

Handling Exploration and Exploitation

REINFORCE:

- **Epsilon-Greedy Exploration:** REINFORCE utilizes an explicit epsilon-greedy strategy, where the agent initially explores by selecting random actions with high probability (e.g., $\epsilon = 1.0$) and gradually shifts toward exploitation as ϵ decays. In the standard environment, this mechanism allowed the agent to eventually discover and refine the optimal safe path.
- **Challenges in the Hard Environment:** In the hard environment, however, the heavy penalties for deviations make random exploration extremely costly. The sparse positive feedback means that even with a decaying ϵ, the agent receives insufficient reinforcement for the correct actions. This results in high variance in the gradient estimates, causing the learning process to stagnate.

Genetic Algorithm:

- **Population-Based Exploration:** The Genetic Algorithm does not rely on gradient information; instead, it evolves a population of candidate policies through selection, crossover, and mutation. This approach naturally fosters exploration because diverse policies are maintained and recombined over generations.
- **Exploitation through Selection:** Exploitation occurs when the fittest individuals are chosen to produce offspring, thereby propagating beneficial traits in the policy parameters. In the standard environment, this method steadily improved the overall fitness.
- **Difficulties in the Hard Environment:** In the hard environment, however, the majority of individuals receive extremely low fitness scores due to pervasive penalties. The resulting lack of differentiation between individuals leads to weak selection pressure. Although mutation and crossover introduce variability, the overall poor fitness landscape hinders the GA's ability to consistently evolve towards the optimal safe path.

Situational Preferences and Trade-Offs

- **When to Prefer REINFORCE:**
  - In tasks where the reward signal is dense and informative, allowing the gradient-based approach to perform efficient fine-tuning.
  - In environments where computational efficiency is critical and continuous policy updates are preferred.
  - When there is enough time for the agent to transition from exploration to exploitation via a well-calibrated epsilon schedule.
- **When to Prefer Genetic Algorithms:**
  - In problems where the reward function is non-differentiable or inherently noisy, making gradient estimates unreliable.
  - In scenarios that require a broad exploration of the solution space, as the population-based search can help avoid local optima.

- In environments with extremely sparse rewards or rugged fitness landscapes (such as our hard environment), where the parallel exploration inherent in GA might eventually identify promising regions despite slow convergence.

Conclusion

In summary, both the REINFORCE algorithm and the Genetic Algorithm offer distinct approaches to managing the exploration-exploitation trade-off. REINFORCE benefits from explicit, tunable exploration via epsilon-greedy strategies and typically converges faster when reward signals are dense. In contrast, the GA's population-based method provides robustness in challenging environments but may converge slowly when the fitness landscape is dominated by severe penalties, as was the case in our hard environment tests.

Despite extensive testing and parameter tuning, both methods struggled significantly in the hard environment due to the sparse rewards and heavy penalties that suppressed effective exploration. These findings underscore the importance of designing reward structures that balance the cost of exploration against the potential benefits of discovering the optimal policy. Future work could focus on developing adaptive reward shaping or hybrid methods to better cope with such extreme conditions.

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?

Observation Space:

In the CartPole environment, the state is represented as a continuous 4-dimensional vector:

$$\mathbf{s} = \begin{pmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{pmatrix},$$

where:

- x is the horizontal position of the cart.
- $\dot{x}$ is the horizontal velocity of the cart.
- θ is the angle of the pole relative to the vertical.
- $\dot{\theta}$ is the angular velocity of the pole.

Typical constraints include:

$$x \in [-4.8, 4.8], \quad \theta \in [-0.418, 0.418] \text{ radians},$$

while the velocity components $\dot{x}$ and $\dot{\theta}$ are either unbounded or assigned large limits.

Action Space:

The action space is discrete, consisting of two possible actions:

$$\mathcal{A} = \{0, 1\},$$

where:

- 0 corresponds to applying a force to move the cart to the left.
- 1 corresponds to applying a force to move the cart to the right.

This discrete space is typically implemented in Gym using the `Discrete` type.

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?

Role of the Discount Factor (γ) in Reinforcement Learning:

The discount factor γ is a crucial parameter in reinforcement learning that determines how future rewards are valued relative to immediate rewards. It is used to compute the discounted return G_t at time step t as follows:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k},$$

where R_{t+k} is the reward received k steps into the future.

Purpose and Impact:

- **Balancing Immediate and Future Rewards:** The value of γ dictates the trade-off between immediate and future rewards. A smaller γ makes the agent more short-sighted, focusing primarily on immediate rewards. Conversely, a larger γ makes the agent consider long-term benefits.
- **Convergence of Returns:** With $\gamma < 1$, the contributions of rewards further in the future decay exponentially, ensuring that the sum G_t converges even in infinite-horizon tasks.
- **Incorporating Uncertainty:** A lower γ can be useful in uncertain environments, as it reduces the impact of highly uncertain distant rewards on current decisions.

Special Cases:

- $\gamma = 0$: The agent becomes completely myopic, considering only the immediate reward. The return simplifies to:
$$G_t = R_t.$$
- $\gamma = 1$: The agent values future rewards equally to immediate rewards. In this case, the return is simply the sum of all future rewards:
$$G_t = \sum_{k=0}^{\infty} R_{t+k}.$$
However, if the episode is infinite or the rewards are unbounded, this sum may diverge, making it difficult to learn an optimal policy.

2.3 Question 3:

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

Why Introduce a Baseline in the REINFORCE Algorithm?

In the REINFORCE algorithm, the update to the policy parameters θ is given by:

$$\Delta\theta \propto \nabla_\theta \log \pi_\theta(a_t|s_t)\, G_t,$$

where G_t is the discounted return from time t. However, this formulation can suffer from high variance because the return G_t can vary greatly from one episode to another, making the gradient estimates noisy.

Role of the Baseline:

A *baseline* $V(s_t)$ is introduced to reduce this variance without introducing any bias to the gradient estimate. The modified update rule becomes:

$$\Delta\theta \propto \nabla_\theta \log \pi_\theta(a_t|s_t)\ (G_t - V(s_t))\,,$$

where $V(s_t)$ is an estimate of the expected return from state s_t.

Contributions to Training Stability:

- **Variance Reduction:** By subtracting $V(s_t)$ from G_t, the update is based on the *advantage* $A(s_t, a_t) = G_t - V(s_t)$. This advantage measures how much better or worse the received return is compared to the expected return from that state. Since the baseline captures the average behavior, the difference has lower variance, leading to more stable and reliable gradient estimates.
- **Improved Learning Dynamics:** With a lower-variance gradient, the policy updates become smoother. This helps in avoiding large, erratic updates to the policy, thereby improving the convergence properties of the learning process.
- **Faster Convergence:** Lower variance in the gradient estimates means that the learning algorithm can make more consistent progress towards an optimal policy. As a result, the agent tends to converge to a better policy in fewer episodes compared to using raw returns.
- **Bias-Variance Trade-off:** Importantly, if the baseline $V(s_t)$ is chosen appropriately (often by training a separate value network), it does not introduce any bias into the gradient estimate. The expected value of the advantage remains the same as the original gradient, ensuring that the learning remains correct while benefiting from reduced variance.

Summary:

The introduction of a baseline in REINFORCE effectively reduces the variance of the policy gradient updates by normalizing the returns using an estimate of the expected reward. This results in smoother, more stable updates and often leads to faster convergence to an optimal policy, all without compromising the unbiased nature of the gradient estimate.

2.4 Question 4:

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

Primary Challenges in Policy Gradient Methods like REINFORCE

Policy gradient methods, such as REINFORCE, directly optimize the policy by estimating gradients from sampled trajectories. Despite their conceptual simplicity and direct approach to optimizing the policy, they face several significant challenges:

High Variance in Gradient Estimates:

- The gradient estimate $\nabla_\theta \log \pi_\theta(a_t|s_t)G_t$ is computed using returns G_t that are inherently noisy due to the stochasticity in the environment and the policy.

- This high variance can lead to unstable and erratic policy updates, making convergence to an optimal policy difficult.

Sample Inefficiency:

- REINFORCE typically requires a large number of episodes or samples to obtain reliable gradient estimates.
- The need for extensive sampling makes these methods computationally expensive and slow to learn, especially in complex environments.

Sensitivity to Hyperparameters:

- The performance of policy gradient methods is highly sensitive to the choice of hyperparameters, such as the learning rate, discount factor γ, and the architecture of the policy network.
- Improper tuning can lead to issues like slow convergence, divergence, or getting trapped in local optima.

Exploration vs. Exploitation Trade-off:

- Stochastic policies encourage exploration, yet balancing this with exploitation of known good actions is challenging.
- If exploration is too aggressive, the variance increases; if it is too conservative, the algorithm might converge prematurely to a suboptimal policy.

Delayed Credit Assignment:

- In environments with long horizons, it becomes difficult to attribute the eventual reward back to the specific actions that contributed to it.
- This problem of delayed credit assignment further exacerbates the variance in the gradient estimates and can hinder learning.

Summary:

In summary, the primary challenges associated with policy gradient methods like REINFORCE include high variance in gradient estimates, sample inefficiency, sensitivity to hyperparameter settings, difficulties in balancing exploration with exploitation, and challenges in delayed credit assignment. Addressing these issues often involves using techniques such as baselines, variance reduction methods, and careful hyperparameter tuning to achieve more stable and efficient learning.

2.5 Question 5:

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

Analysis of the Training Results:

In our experiments with the CartPole environment, we trained two agents using the REINFORCE algorithm:

- **REINFORCE without a baseline:**
  - The training log shows that the rewards start very low (e.g., 17 at episode 50) and gradually improve.
  - However, the convergence to the maximum reward (500) takes around 2500 episodes.
  - The variability in rewards is higher, as indicated by a mean reward of approximately 420.62 with a standard deviation of 141.67.
- **REINFORCE with a baseline:**
  - Early on, the rewards are low (e.g., around 11–65 for the initial episodes), but a rapid improvement is seen, with near-optimal performance (reward of 500) achieved by around episode 1000.
  - The mean reward is slightly higher at approximately 450.77 with a lower standard deviation of 124.02, indicating less variability.

Comparison in Terms of Performance:

The introduction of a baseline in the REINFORCE algorithm has a significant impact on training:

- **Faster Convergence:** The agent using the baseline reaches the optimal reward of 500 much earlier (around episode 1000) compared to the non-baseline version, which converges closer to episode 2500.
- **Reduced Variance:** The standard deviation of the rewards is lower for the baseline method, meaning that the policy updates are more stable and consistent, leading to smoother learning curves.
- **Overall Optimality:** Despite the differences in convergence speed and variance, both methods are capable of learning an optimal policy as they both eventually achieve the maximum reward.

Conclusion:

REINFORCE with a baseline outperforms the standard REINFORCE algorithm by reducing the variance in gradient estimates. This results in faster convergence and more stable learning, as evidenced by the earlier attainment of optimal performance and lower variability in rewards. Nonetheless, both approaches are ultimately capable of achieving an optimal policy in the CartPole environment.

2.6 Question 6:

Explain how variance affects policy gradient methods, particularly in the context of estimating gradients from sampled trajectories.

Impact of Variance on Policy Gradient Methods:

In policy gradient methods, such as REINFORCE, the gradient of the policy parameters θ is estimated using sampled trajectories. The update rule is typically given by:

$$\Delta\theta \propto \nabla_\theta \log \pi_\theta(a_t|s_t)\, G_t,$$

where G_t is the discounted return obtained from the trajectory.

Role of Variance:

- **Noisy Estimates:** Since G_t is computed from sampled trajectories, which inherently include randomness from both the environment and the stochastic policy, the resulting gradient estimates are noisy. High variance in these estimates means that the direction and magnitude of the update can vary significantly from one sample to another.
- **Instability in Learning:** Large variance in gradient estimates leads to unstable updates of the policy parameters. This instability can cause the learning process to oscillate or diverge, making it harder for the agent to converge to an optimal policy.
- **Inefficient Use of Samples:** High variance often necessitates the collection of a large number of trajectories to average out the noise and obtain a reliable estimate of the gradient. This reduces sample efficiency and increases the computational burden of training.
- **Mitigation Techniques:** To counteract these issues, various variance reduction techniques are employed. One common method is the introduction of a *baseline* $V(s_t)$, which leads to the use of the advantage function:

  $$A(s_t, a_t) = G_t - V(s_t).$$

  Subtracting the baseline does not change the expected value of the gradient but significantly reduces its variance, leading to smoother and more stable policy updates.

Summary:

High variance in gradient estimates from sampled trajectories can severely hinder the learning process in policy gradient methods by causing unstable updates and requiring more samples to average out the noise. Employing techniques like baselines to reduce variance is crucial for achieving faster convergence and more reliable performance in reinforcement learning.

2.7 Additional Information:

The complete notebook is available at the following URL:

link

This notebook provides a simple yet comprehensive implementation of the REINFORCE algorithm both with and without a baseline in the CartPole environment. The total runtime for the notebook is approximately 1h 8m 7s.

Please note that the link is currently set to private; if public access is needed, I can make it public. The notebook will be uploaded alongside the PDF version of the report.

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:
The `MountainCarContinuous-v0` environment defines its observation space as a continuous **Box** with two dimensions. The state is represented as a vector

$$s = \begin{pmatrix} x \\ \dot{x} \end{pmatrix},$$

where:

- x is the position of the car, typically bounded within $[-1.2, 0.6]$.
- $\dot{x}$ is the velocity of the car, typically bounded within $[-0.07, 0.07]$.

Action Space:
The action space is defined as a one-dimensional continuous **Box**. The action represents the force applied to the car and is a scalar value:

$$a \in [-1, 1].$$

A positive value applies force to the right, while a negative value applies force to the left.

Summary:
In summary, the environment uses:

- A 2D continuous observation space, with state $s = (x, \dot{x})$.
- A 1D continuous action space, where the action a is the force applied, bounded in $[-1, 1]$.

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.

An optimal policy for reaching the goal in the `MountainCarContinuous-v0` environment while using the least amount of energy would involve exploiting the dynamics of the system—specifically, the interplay between gravitational forces and momentum—so that the agent minimizes unnecessary force applications. Below is a detailed scenario of how such an optimal behavior might be achieved:

1. Energy-Efficient Momentum Building:
The agent begins by applying a relatively weak force to the left. This action is counterintuitive at first glance, but it leverages gravity to move the car backward down the hill. By doing so, the car accumulates kinetic energy without expending excessive energy from the agent. Mathematically, the dynamics can be summarized as:

$$\text{Momentum: } p = m \cdot v \quad \text{and} \quad \Delta v \propto a - g\sin(\theta)$$

where a is the applied acceleration and $g\sin(\theta)$ represents the component of gravitational acceleration. The policy learned via REINFORCE (as indicated by our final mean reward of approximately 20.47 ± 97.9090) shows that, after extensive training, the agent managed to escape the initially negative reward regime by fine-tuning its actions.

2. Timing of Force Application:
After sufficient momentum is built, the agent applies a short but precise burst of force to the right. This action is timed so that the built-up kinetic energy is optimally converted to move the car up the hill towards the goal. The key here is to avoid using maximum force continuously, which would waste energy and might lead to overshooting the goal. Instead, the optimal policy uses minimal yet sufficient energy input. In our implementation, adjusting the model to properly compute the standard deviation (by returning the $\log \text{std}$ and then exponentiating it) allowed the agent to sample more calibrated actions, ultimately achieving positive rewards.

3. Minimal Energy Use Through Fine-Tuning:
Throughout the episode, the agent's policy, learned via the REINFORCE algorithm, assigns a probability distribution over actions that balances exploration and exploitation. With a well-tuned $\gamma = 0.99$ and an aggressive learning rate (3×10^{-3}), the policy gradually shifted from actions that resulted in high energy expenditure (with large negative rewards) to actions that carefully modulate force. The trajectory observed in our results (where the reward transitions from about -800 up to positive values near $+60$) reflects this evolution in policy.

4. Overall Episode Behavior:
A typical episode under the optimal policy would involve:

- **Initial Phase:** The car starts at the bottom of the valley. The agent applies a small force to the left, allowing gravity to help build momentum as the car moves backwards.
- **Momentum Accumulation:** As the car gains speed, the agent carefully modulates the force to avoid overshooting and to maintain control over the momentum.
- **Transition Phase:** At a critical moment, the agent applies a calculated burst of force to the right. This burst is timed to maximize the conversion of momentum into upward movement, ensuring that the car reaches the top of the hill.
- **Goal Achievement:** Finally, the car barely exceeds the threshold required to reach the goal, doing so with minimal energy expenditure.

The evolution of the reward—from very negative values (e.g., -800) to the eventual positive rewards (peaking near $+60$)—demonstrates that the initial policy was not energy-efficient. By modifying the model to return the $\log \text{std}$ and properly computing the standard deviation in the REINFORCE algorithm, the agent was eventually able to learn a behavior that efficiently utilizes gravitational forces and momentum. This resulted in a mean reward of 20.47 ± 97.9090, indicating that, on average, the agent learned to reach the goal while minimizing unnecessary energy use, though some variability remains in performance.

Conclusion:
In summary, the optimal policy in the `MountainCarContinuous-v0` environment uses a strategy of controlled force application and momentum building. By initially leveraging gravity and later timing force application precisely, the agent can reach the goal with minimal energy consumption. The improvements in our experimental results highlight the importance of correctly modeling the action distribution parameters (i.e., using the $\log \text{std}$ and computing $\exp(\log \text{std})$) in achieving such energy-efficient behavior.

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)

Strategies to Mitigate Catastrophic Forgetting:

Catastrophic forgetting occurs when a model forgets previously learned information upon learning new data. In continuous action space environments such as `MountainCarContinuous-v0`, where the policy must carefully balance energy usage and momentum, this issue can be particularly pronounced. Several strategies can be employed to reduce catastrophic forgetting:

1. **Experience Replay:**
   This technique involves storing past transitions (states, actions, rewards, and next states) in a replay buffer. During training, mini-batches of experiences are sampled uniformly or with priority from this buffer, ensuring that the agent continually revisits older experiences. This helps in:
   - Stabilizing learning by breaking correlations between sequential data.
   - Preventing the network from overfitting to recent experiences.

   In our experiments, the agent initially received highly negative rewards (around -800) and only later began to improve after many episodes. Incorporating an experience replay buffer could help the agent retain early learning about the environment's dynamics, potentially leading to faster convergence and more stable performance.
2. **Target Networks:**
   Another strategy is to use a target network, a separate copy of the policy (or value) network, which is updated less frequently than the main network. The target network provides stable targets for the update equations, reducing the oscillations and divergence that may arise when the same network is used for both prediction and target estimation. This technique is common in value-based methods, but can also be adapted to actor-critic frameworks in continuous action spaces.
3. **Regularization Techniques:**
   Techniques such as L2 regularization or more sophisticated approaches like Elastic Weight Consolidation (EWC) can help mitigate catastrophic forgetting by penalizing significant changes to weights that are important for previously learned tasks. In the context of continuous control, such regularization ensures that adjustments to improve performance in one region of the state space do not drastically impair performance in others.
4. **Hybrid Approaches:**
   Combining experience replay with target networks in off-policy actor-critic methods (such as DDPG or SAC) can leverage the advantages of both strategies. While REINFORCE is inherently on-policy, moving to an off-policy variant allows one to incorporate these techniques, potentially leading to more robust learning especially in environments where rewards are sparse or highly variable.

Summary:
To reduce catastrophic forgetting in environments like `MountainCarContinuous-v0`, an agent could benefit from:

- Storing and sampling past experiences to maintain a diverse learning signal.

- Employing a target network to provide stable learning targets.
- Using regularization to prevent drastic weight changes.

These strategies, if implemented, might help the agent achieve and sustain positive rewards more reliably, as observed in our experiment where the final mean reward reached approximately 20.47 ± 97.9090 after extensive training.

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?

In our experiments on the Frozen Lake environment, the Deep Q-Network (DQN) algorithm demonstrated superior performance compared to the Policy Gradient (REINFORCE) method. The following detailed observations and analysis reference the experimental results:

1. **Empirical Observations:**
    - **DQN:** The agent trained using DQN showed significant improvement after around 2000 epochs and converged to an optimal reward of 1 by 3000 epochs. In the testing phase, the DQN-trained agent was able to reach the goal reliably, confirming that it had learned an effective policy.
    - **REINFORCE:** In contrast, the agent trained with the REINFORCE algorithm did not improve even after 6000 epochs; it consistently received a reward of 0 during training and remained stuck in the initial cell without moving toward the goal. Despite attempting around 20 different hyperparameter variations, the best performance obtained was still zero reward, indicating a failure to learn.
2. **Algorithmic Differences and Their Impact:**
    - **Training Stability:** DQN utilizes experience replay and a target network, which are crucial for stabilizing the training process. These mechanisms help break the correlations between consecutive samples and reduce the variance of the learning updates. This stability is reflected in the observed rapid improvement and convergence. Conversely, REINFORCE directly relies on policy gradient estimates, which have inherently high variance. This instability is a major factor contributing to the REINFORCE agent's inability to improve in our experiments.
    - **Convergence Speed:** The off-policy nature of DQN enables the agent to reuse past experiences effectively, leading to faster convergence. The experimental evidence shows DQN achieving near-optimal rewards by 3000 epochs. On the other hand, REINFORCE, being an on-policy method, requires fresh samples for every update. Its high variance prevents efficient learning, which explains why the REINFORCE agent failed to make any progress even after 6000 epochs.
    - **Overall Success Rate:** The final success rate is a clear indicator of performance. The DQN-trained agent not only converged faster but also consistently reached the goal during testing. In contrast, the REINFORCE agent was unable to move from the starting cell and never reached the goal, resulting in an overall success rate of 0%.
3. **Discussion:**
    - The experimental outcomes highlight that DQN's design choices (experience replay, target network) are particularly effective in environments with sparse rewards, like Frozen Lake. These mechanisms help to extract and reinforce useful learning signals even when positive rewards are rare.

- REINFORCE, while simpler in concept, suffers from high variance and sample inefficiency. In an environment where the reward is sparse and the penalty for non-optimal moves is high, these shortcomings become more pronounced, leading to the observed training failure.

Conclusion: Based on our experimental observations and detailed analysis, DQN achieves better results in the Frozen Lake environment. It offers greater training stability, faster convergence, and a higher overall success rate. The REINFORCE method, due to its high variance and on-policy limitations, fails to learn an effective policy in this scenario.

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 unique challenges for reinforcement learning, which affect both DQN and Policy Gradient methods. These challenges include:

1. **Sparse Rewards:**
   - **Description:** In Frozen Lake, the agent receives a reward only when it reaches the goal state, while most transitions yield zero or negative feedback. This scarcity of positive rewards makes it difficult for the agent to learn which actions are truly beneficial.
   - **Impact:** For DQN, the replay buffer helps to reuse rare successful experiences; however, the scarcity still means that many samples provide little learning signal. For Policy Gradient methods (e.g., REINFORCE), the high variance in returns due to sparse rewards can lead to unstable gradient estimates, hindering effective policy updates.
2. **Stochasticity and Slippery Dynamics:**
   - **Description:** Although some implementations of Frozen Lake allow a deterministic (non-slippery) version, the original environment is stochastic, meaning that the same action may lead to different outcomes.
   - **Impact:** Stochastic transitions increase uncertainty in both methods. For DQN, it complicates the Q-value estimation since the agent must account for a wider range of possible next states. For Policy Gradient methods, the randomness further increases the variance of the gradient estimates, making learning more challenging.
3. **Discrete and Low-Dimensional State Space:**
   - **Description:** The state space in Frozen Lake is typically represented as a discrete grid (e.g., an 8x8 grid), where each cell represents a state.
   - **Impact:** Although a low-dimensional state space might seem advantageous, the discrete nature means that exploration becomes critical. Both DQN and Policy Gradient methods must balance exploration and exploitation carefully. For DQN, efficient exploration is aided by the epsilon-greedy strategy; for Policy Gradient methods, insufficient exploration (or high variance in exploration) can prevent the agent from discovering rewarding trajectories.
4. **Risk of Suboptimal Policies:**

- **Description:** Given the structure of the environment, it is easy for the agent to converge on suboptimal policies, such as repeatedly selecting actions that avoid immediate penalties but never reach the goal.
- **Impact:** In our experiments, the REINFORCE method fell into this trap, with the agent getting stuck in the starting cell and never receiving a reward. DQN, by contrast, leverages its target network and replay mechanism to eventually discover the optimal path, although it also faces the risk of overfitting to misleading transitions.

5. **Delayed Feedback:**
   - **Description:** The only significant reward is typically received at the end of an episode when the goal is reached, meaning that actions taken early in an episode may have a delayed impact on the outcome.
   - **Impact:** For DQN, delayed rewards require careful temporal credit assignment, which can be partially mitigated through the use of discount factors and target networks. For Policy Gradient methods, the delayed nature of rewards increases the difficulty of assigning proper credit to earlier actions, contributing to the high variance of the gradient estimate.

Conclusion: The Frozen Lake environment's sparse rewards, potential stochasticity, discrete state space, risk of suboptimal policies, and delayed feedback all contribute to significant challenges in the reinforcement learning process. DQN partially overcomes these challenges by reusing past experiences and stabilizing learning through its replay buffer and target network. In contrast, Policy Gradient methods like REINFORCE struggle more due to high variance in gradient estimates and inefficient exploration, which can lead to failure in discovering the optimal policy.

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 with low-cost sampling, the Policy Gradient approach tends to be more advantageous compared to DQN. This conclusion is based on several factors:

1. **Sample Efficiency:**
   - **Policy Gradient:** Although policy gradient methods (such as REINFORCE) are generally considered less sample-efficient than off-policy methods like DQN, the unlimited sampling capability mitigates this drawback. With abundant interactions, the high variance in gradient estimates can be reduced through large batch sizes and multiple episodes, ultimately leading to a well-optimized policy.
   - **DQN:** DQN is designed to be sample efficient by reusing past experiences from the replay buffer. However, its advantages in sample efficiency become less critical when sampling is effectively unlimited.
2. **Function Approximation and Policy Representation:**

- **Policy Gradient:** Policy gradient methods directly parameterize the policy, which can be advantageous when learning complex or stochastic policies. The direct optimization of the policy allows for smooth improvements, especially when function approximators (e.g., deep neural networks) are properly tuned.
- **DQN:** DQN approximates the Q-function and indirectly derives the policy through a greedy selection process. While effective, this indirect approach may lead to issues in complex or highly stochastic environments where explicit policy representation would be beneficial.

3. **Stability of Learning:**
   - **Policy Gradient:** In settings with unlimited interactions, the high variance typically associated with policy gradient methods can be controlled by increasing the number of sampled episodes and applying variance reduction techniques (e.g., baselines, advantage normalization). This leads to a more stable learning process over time.
   - **DQN:** DQN employs mechanisms such as experience replay and target networks to stabilize learning. However, in scenarios where computational cost is not a constraint, the relative simplicity of direct policy optimization in policy gradient methods can lead to stable convergence without the overhead of maintaining a replay buffer.
4. **Long-Term Optimization:**
   - **Policy Gradient:** With unlimited interactions, policy gradient methods can explore a wide variety of state-action trajectories and fine-tune the policy iteratively. This capacity for extensive exploration makes it possible to learn robust and high-performing policies even in complex environments.
   - **DQN:** Although DQN can perform well in many domains, its off-policy nature may sometimes limit the direct improvement of the policy in cases where explicit policy representation is beneficial.

Conclusion: In scenarios with unlimited interactions and low-cost sampling, the Policy Gradient approach is more advantageous. The abundance of data helps overcome the inherent sample inefficiency and high variance of policy gradient methods. Moreover, the direct optimization of the policy offers benefits in terms of function approximation and adaptability, leading to stable and robust learning in such environments.

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/