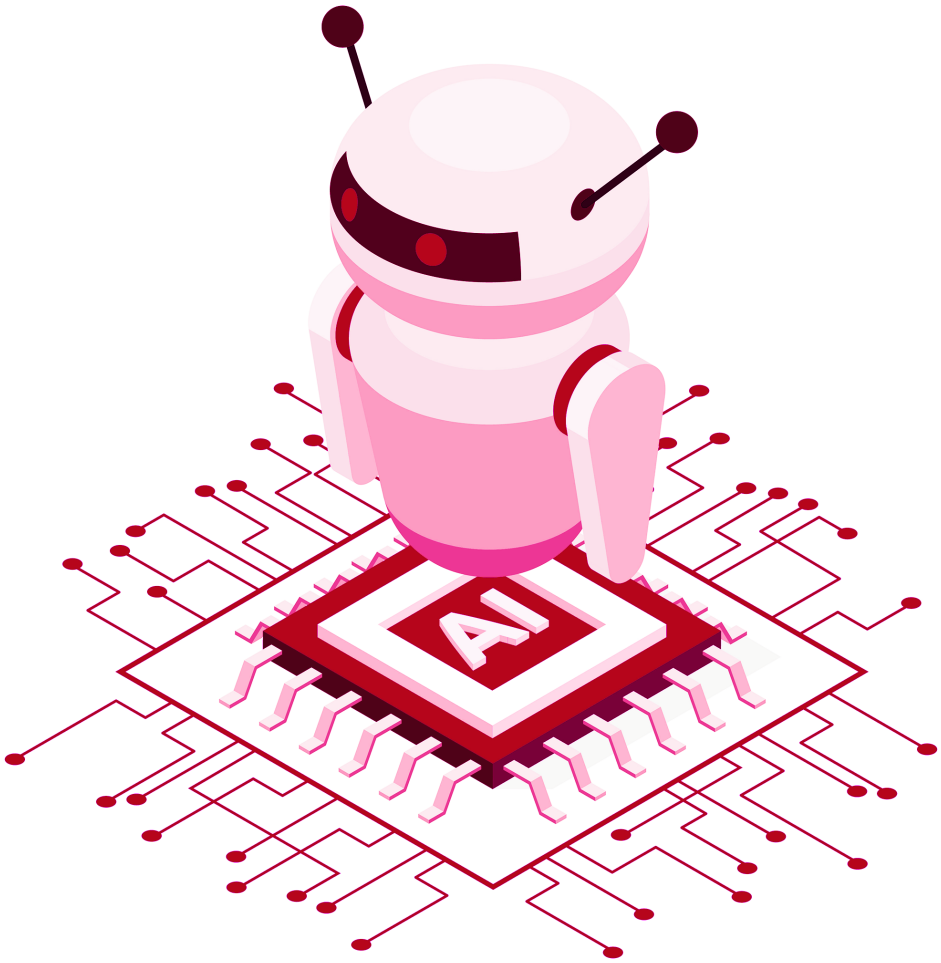

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 3:

Policy-Based Methods

By:

Ali Najar

401102701

RIML

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]

We compare two policy optimization methods for reinforcement learning (RL) in a **GridWorld** environment:

1. **REINFORCE (Policy Gradient Method)**
2. **Genetic Algorithm (GA)**

Both methods are evaluated based on their ability to optimize an agent's policy in a grid-based environment with penalties and rewards.

GridWorld Environment

The environment used in this study is a **7 × 7 GridWorld**, where an agent moves in a discrete space to reach a goal while avoiding penalty locations. The environment is defined as follows:

State Space: The state space consists of all possible (x, y) positions in a 7×7 grid:

$$S = \{(x, y) \mid x, y \in \{0, 1, 2, ..., 6\}\} \tag{1}$$

Action Space: The agent can take four possible actions:

$$A = \{\text{left}, \text{right}, \text{up}, \text{down}\} \tag{2}$$

Transition Dynamics: The agent moves in the direction of the chosen action unless blocked by a boundary:

$$s' = (x', y') = (x + \Delta x, y + \Delta y) \tag{3}$$

where $(\Delta x, \Delta y)$ depends on the chosen action:

$$\text{left} : (-1, 0) \tag{4}$$
$$\text{right} : (1, 0) \tag{5}$$
$$\text{up} : (0, 1) \tag{6}$$
$$\text{down} : (0, -1) \tag{7}$$

If the new position is outside the grid, the agent remains in the same position.

Rewards: The environment provides the following rewards:

- **Goal State** at $(6, 6)$: $+100$
- **Step Penalty** for every move: -1
- **Penalty Locations**: Specific states have a penalty of -10

The penalty locations are defined as:

$$P = \{(1, 5), (2, 2), (5, 3), (4, 4), (4, 5), (5, 1), (2, 3), (2, 0)\} \tag{8}$$

Terminal State: The episode terminates when the agent reaches the goal at $(6, 6)$.

During the experiments I have used the hyperparameters below.

REINFORCE Hyperparameters

- **Learning Rate (lr)**: 0.005
- **Discount Factor (γ)**: 0.99
- **Episodes**: 6000
- **Hidden Layer Size ($hidden\_dim$)**: 32

Genetic Algorithm Hyperparameters

- **Population Size (P)**: 30
- **Generations (G)**: 200
- **Mutation Rate (m)**: 0.05
- **Crossover Rate (c)**: 0.5
- **Hidden Layer Size ($hidden\_dim$)**: 32

1.1 Question 1:

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

Both REINFORCE and Genetic Algorithms (GA) are used for policy optimization, but they differ in effectiveness based on how they learn and adapt to reinforcement learning (RL) tasks.

1. **Learning Approach:**
   - **REINFORCE**: Uses policy gradients to directly update parameters based on action probabilities.
   - **GA**: Uses evolutionary principles such as selection, mutation, and crossover to improve policies.
2. **Sample Efficiency:**
   - **REINFORCE**: Higher sample efficiency as it updates parameters using feedback from each episode.
   - **GA**: Lower sample efficiency as it evaluates an entire population before selecting better policies.
3. **Exploration vs. Exploitation:**
   - **REINFORCE**: Uses a stochastic policy (e.g., softmax) to balance exploration and exploitation.
   - **GA**: Relies on mutations to introduce randomness.
4. **Convergence and Stability:**
   - **REINFORCE**: Can converge faster but suffers from high variance due to noisy gradient updates.
   - **GA**: More stable as it does not rely on gradients, but convergence is slower due to population-based evolution.
5. **Suitability for Different RL Problems:**
   - **REINFORCE**: Works well for problems with smooth reward landscapes.
   - **GA**: Suitable for environments with deceptive or sparse rewards (e.g., exploration-heavy tasks).

Aspect	REINFORCE	Genetic Algorithm
Learning Type	Gradient-based	Evolutionary-based
Sample Efficiency	High	Low
Exploration Mechanism	Stochastic policy	Mutation and crossover
Convergence Speed	Faster (if tuned)	Slower
Stability	High variance, requires tuning	More stable

Table 1: Comparison of REINFORCE and Genetic Algorithm in RL

1.2 Question 2:

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

1. Performance

- **REINFORCE:** achieves higher performance in structured environments where reward functions provide a clear learning signal. Since it directly updates policy parameters using gradients, it learns effective strategies relatively quickly.
- **GA:** It may struggle in high-dimensional tasks. It relies on random mutations and selection, which can slow down progress toward an optimal policy.

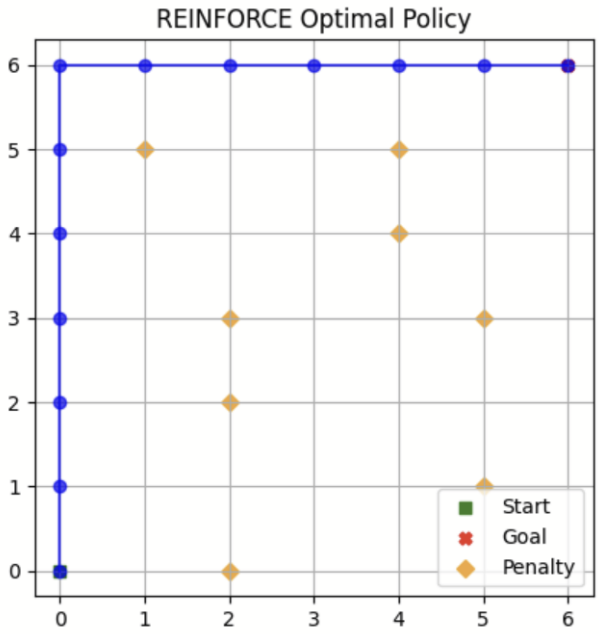

Figure 1: REINFORCE

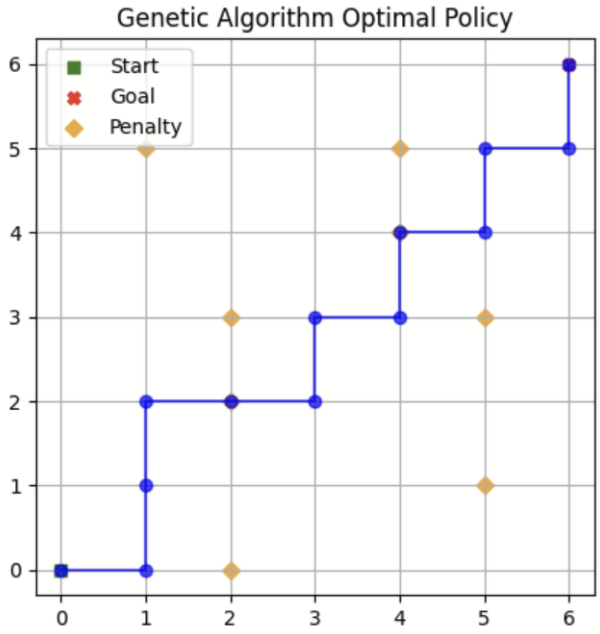

Figure 2: GA

From (1) and (2) it can be seen that both algorithms reach a good strategy (REINFORCE slightly better) but overall REINFORCE had a better performance because of better sample efficiency.

2. Convergence Rate

- **REINFORCE:** It converges faster since it updates the policy every episode. However, its convergence depends on proper tuning of the learning rate and other hyperparameters.

- **GA:** It is slower to converge as it relies on evolutionary principles rather than direct policy updates. It requires testing many policies over multiple generations before finding an optimal one.

GA convergence rate is much slower since its randomness and evolutionary principles make the process of finding an optimal policy harder. Despite all these it converges smoother but at a slower rate than REINFORCE algorithm.

A drawback of REINFORCE is its **high variance**, meaning training can be unstable. Techniques like baseline subtraction and variance reduction are often used to improve stability. These lead to REINFORCE having more sharp updates. This means that REINFORCE reaches an optimal solution faster but since the updates are noisy it can sometimes face some difficulties.

3. Stability

- **REINFORCE:** is more sensitive to noise in rewards and policy updates. Since it relies on sampled episodes and gradients, it can suffer from **high variance**, making training unstable.
- **GA:** is generally more stable because it does not rely on gradients. Instead, it maintains a population of policies that evolve steadily over time.

However, GA can suffer from **premature convergence**, where diversity in the population is lost too early, leading to suboptimal solutions. In figure (2) it can be seen that the agent moves through one of the cells with penalty and since the population does not have the diversity of other policies, this remain the best policy.

Despite all these REINFORCE algorithm has less stability since it is very sensitive to noise in rewards and updates. But GA shows more smooth updates and acts more smooth during each generation.

1.3 Question 3:

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

1. REINFORCE

REINFORCE relies on a stochastic policy, meaning it selects actions based on probability distributions. This naturally provides a mechanism for exploration, as actions with lower probabilities still have a chance of being chosen.

Early in training, exploration is encouraged because the policy is highly uncertain. Over time, as the policy learns which actions yield higher rewards, it shifts towards exploitation. A common technique used in REINFORCE is entropy regularization (which I tried to use in Notebook 3), which encourages more exploration by preventing the policy from becoming too deterministic too soon. REINFORCE smoothly transitions from exploration to exploitation by adjusting probabilities over time.

2. GA

GA approaches the problem differently by maintaining a population of policies rather than a single one. Exploration and exploitation emerge through:

- **Mutation**: Introduces randomness by modifying policies, allowing exploration of new solutions.
- **Crossover**: Combines good policies to refine strategies, reinforcing exploitation.
- **Selection**: Ensures that better policies survive, reinforcing the learning of effective strategies.

Unlike REINFORCE, GA does not directly optimize for exploitation; instead, it relies on natural selection to retain effective policies while introducing diversity through mutations.

3. When to Prefer REINFORCE vs. GA

- **REINFORCE:**
  - When the reward function is smooth and informative.
  - When the problem requires fast learning and adaptation.
  - When the policy space is high-dimensional, making gradient-based learning more efficient.
- **GA:**
  - When the environment has sparse or deceptive rewards.
  - When the problem involves misleading gradients, making policy gradient methods unreliable.
  - When broad exploration is necessary to find good solutions.

Overall, REINFORCE is often more practical due to its efficiency, but GA is valuable when extensive exploration is needed. The decision should be based on the problem's complexity, reward structure, and available computational resources.

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?

The CartPole environment represents a classic reinforcement learning control problem where the goal is to balance a pole on a moving cart. The environment defines both an observation space and an action space as follows:

Observation Space: The environment provides a 4-dimensional continuous observation space that represents the current state of the system:

$$s = (x, \dot{x}, \theta, \dot{\theta}) \in \mathbb{R}^4$$

where:

- $x \in [-4.8, 4.8]$ is the **cart position** on a 1D track.
- $\dot{x} \in (-\infty, \infty)$ is the **cart velocity**.
- $\theta \in [-0.418, 0.418]$ is the **pole angle** relative to the vertical (radians).
- $\dot{\theta} \in (-\infty, \infty)$ is the **pole angular velocity**.

Action Space: The environment has a **discrete action space** with two possible actions:

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

where:

- $a = 0$ applies a force F to move the cart **left**.
- $a = 1$ applies a force F to move the cart **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 γ plays a fundamental role in reinforcement learning by controlling how much future rewards contribute to the current decision-making process. It is a parameter in the range $0 \leq \gamma \leq 1$ and appears in the objective function:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right]$$

where R_t is the reward at time step t.

Role of γ:

- If γ is **close to 1**, the agent values long-term rewards, making it focus on optimizing performance over an extended period.
- If γ is **close to 0**, the agent prioritizes immediate rewards, ignoring future consequences.

A proper balance of γ is crucial for achieving optimal policies, especially in environments with delayed rewards.

Extreme Cases:

- **Case 1: $\gamma = 0$ (Greedy Behavior)**
  When $\gamma = 0$, the objective function simplifies to:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} 0^t R_t\right] = \mathbb{E}\left[R_0\right]$$

  This means that the agent only considers the **immediate reward** R_0 and completely ignores any future rewards. As a result:
  - The agent makes decisions based on short-term gains.
  - It may fail to learn optimal long-term strategies.
  - This is only suitable for environments where short-term rewards always lead to the best long-term outcomes.
- **Case 2: $\gamma = 1$ (Infinite Horizon Optimization)**
  When $\gamma = 1$, the objective function becomes:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} R_t\right]$$

  This means the agent gives equal importance to all future rewards. While this might seem ideal, it can cause:
  - Instability: If rewards do not diminish over time, the total return may diverge, making learning difficult.
  - Delayed Gratification: The agent may over-prioritize (gratitude) future rewards at the expense of immediate gains.

  In practical implementations, γ is usually set slightly less than 1 (e.g., $\gamma = 0.99$) to ensure that rewards further in the future still contribute but diminish gradually.

2.3 Question 3:

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

The REINFORCE algorithm is a Monte Carlo policy gradient method that updates the policy parameters

using sampled trajectories from the environment. One major challenge in REINFORCE is the **high variance** in gradient estimates, which can lead to unstable training and slow convergence. To mitigate this, a **baseline** is introduced.

Policy Gradient Without a Baseline: The standard policy gradient update in REINFORCE is given by:

$$\Delta\theta = \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) G_t$$

where G_t is the discounted return:

$$G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}.$$

Since G_t is a sampled quantity, it introduces **high variance** into the gradient estimates, making learning unstable.

Introducing a Baseline: A common variance reduction technique is to introduce a baseline function $b(s_t)$, which does not change the expectation of the policy gradient but reduces its variance:

$$\Delta\theta = \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)(G_t - b(s_t)).$$

A common choice for $b(s_t)$ is an estimate of the **state-value function**:

$$b(s_t) \approx V(s_t) = \mathbb{E}[G_t \mid s_t].$$

How the Baseline Improves Training Stability:

- **Variance Reduction:** By subtracting $b(s_t)$, the variance of the gradient estimate is reduced, leading to more stable learning.
- **More Efficient Learning:** Lower variance allows the algorithm to take more meaningful updates in the direction of optimal policies.
- **No Bias Introduced:** Since $b(s_t)$ is independent of the action, it does not change the expected value of the gradient, ensuring unbiased learning.

The introduction of a baseline in REINFORCE is an essential technique for stabilizing policy gradient updates. By reducing variance, the algorithm becomes more efficient and converges faster while maintaining an unbiased estimation of the policy gradient. The most common choice for the baseline is the state-value function $V(s)$, which provides a good estimate of expected future rewards.

2.4 Question 4:

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

Policy gradient methods, such as REINFORCE, are widely used in reinforcement learning because they directly optimize policies without requiring value function approximation. However, despite their advantages, they come with several challenges that make them difficult to train efficiently.

1. High Variance in Gradient Estimates

One of the biggest issues with REINFORCE is the high variance in its gradient estimates. The policy update rule is given by:

$$\Delta\theta = \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) G_t,$$

where G_t is the total return from time step t. Since this return is sampled from experience, it can fluctuate significantly, leading to unstable updates. As a result, training can be slow and inconsistent. A common way to reduce variance is to introduce a **baseline**, such as the state-value function $V(s_t)$, which helps smooth out updates.

2. Sample Inefficiency

REINFORCE relies on Monte Carlo sampling, meaning it needs a large number of episodes to estimate gradients accurately. Since it requires full trajectories before updating the policy, it can be slow to learn. Unlike value-based methods that update more frequently, policy gradient methods struggle to make the most of available data. To improve efficiency, methods like **actor-critic** introduce a value function to speed up learning.

3. Slow Convergence

Because REINFORCE updates policies based on entire episode returns, learning can be noisy. Small policy updates often take a long time to produce meaningful changes in behavior. Worse still, if the learning rate α is not tuned properly:

- A large α makes updates unstable.
- A small α makes learning too slow.

Adaptive learning rate methods like Adam help mitigate this, but convergence is still slower compared to Q-learning or actor-critic approaches.

4. Credit Assignment Problem

Another challenge in policy gradient methods is the difficulty of assigning credit to individual actions. Since updates depend on the total reward over an episode, it's unclear whether an action taken early on was beneficial or if later actions played a more important role. This can make it harder for the agent to learn complex strategies.

5. Lack of Off-Policy Learning

REINFORCE is an **on-policy** algorithm, meaning it must generate new trajectories using the current policy for every update. This prevents it from reusing past experiences, unlike off-policy methods like Q-learning, which can leverage stored transitions in a replay buffer. As a result, REINFORCE is less data-efficient and requires more interactions with the environment to learn effectively.

6. Balancing Exploration and Exploitation

Since policy gradient methods adjust action probabilities rather than selecting fixed actions, they must carefully balance exploration and exploitation. If exploration is too low, the agent may get stuck in a suboptimal strategy. If it is too high, the agent might keep exploring without improving performance. One way to handle this is by adding **entropy regularization**, which encourages policies to remain more stochastic in the early stages of 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?

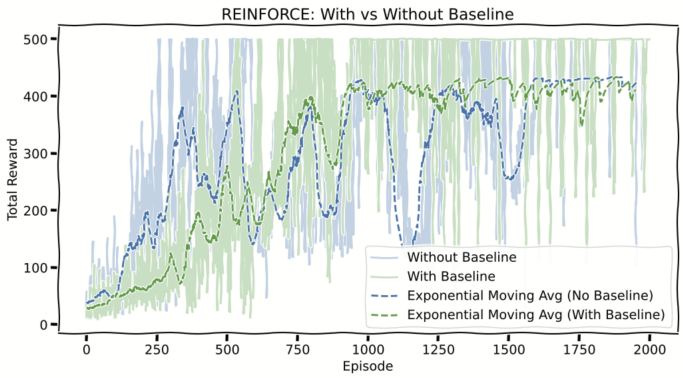

Figure 3: REINFORCE with baseline vs. without baseline

Figure (3) shows vividly that REINFORCE without baseline has many fluctuations while REINFORCE with baseline has smoother transitions. It can be seen that without baseline the algoritm gets more rewards at first but struggles to get better while with baseline the agent updates itself more smoothly.

1. Learning Stability and Convergence Speed

- REINFORCE without a baseline exhibits high variance in gradient estimates, causing unstable learning. The training curve fluctuates significantly, making convergence slower and less predictable.
- REINFORCE with a baseline significantly reduces variance, leading to smoother and more consistent policy updates.

2. Sample Efficiency

- Without a baseline, REINFORCE requires more training episodes to reach a high-performance level due to the noisy policy updates.
- With a baseline, fewer episodes are needed to achieve similar or better performance, demonstrating better sample efficiency.

2.6 Question 6:

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

Gradient Estimation and Variance

In policy gradient methods, the objective function is given by:

$$J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R_t \right],$$

and its gradient is estimated using sampled trajectories:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) G_t \right].$$

Since the return G_t is computed based on stochastic rewards and policy interactions, the gradient estimate has high variance. This variance increases with longer episodes and delayed rewards, leading to unpredictable learning dynamics.

Effects of High Variance

The presence of high variance in policy gradients negatively impacts training in several ways. First, the updates become unstable, causing the policy to oscillate rather than improve steadily. As a result, the learning process is slow, requiring a large number of samples to average out randomness. Furthermore, when variance is too high, the learning signal becomes weak, making it difficult for the agent to distinguish between good and bad actions.

Another major issue is the credit assignment problem. When rewards are delayed, it is challenging to determine which actions contributed to success or failure. This ambiguity further increases variance in the gradient updates, making it harder for the agent to learn an effective strategy.

Sources of Variance

Several factors contribute to the high variance in policy gradient estimates. One of the main causes is the stochastic nature of policies, where actions are sampled probabilistically rather than chosen deterministically. This randomness ensures exploration but also leads to inconsistent learning signals. Additionally, the environment itself introduces uncertainty, as the same action in a given state may yield different outcomes due to external factors.

Another source of variance is the reliance on complete episode returns. Since REINFORCE updates the policy based on entire trajectories, small variations in rewards can significantly affect the computed returns, leading to inconsistent policy updates.

Variance Reduction Techniques

To address the variance issue, several strategies can be employed. One of the most effective techniques is the use of a baseline. By introducing a function $b(s_t)$, typically an estimate of the value function $V(s_t)$, the variance of the gradient estimate is reduced:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)(G_t - b(s_t)) \right].$$

Since the baseline does not depend on actions, it does not introduce bias but significantly reduces variance.

Another common approach is advantage estimation, where the return G_t is replaced with the advantage function:

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

This modification helps stabilize learning by reducing noise in gradient updates.

Actor-critic methods further improve stability by introducing a separate critic network that estimates $V(s_t)$, allowing the policy updates to be guided by a more reliable learning signal. Additionally, entropy regularization can encourage exploration while preventing the policy from becoming too deterministic too early.

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?

In the `MountainCarContinuous-v0` environment an agent must drive a car up a hill by applying continuous force. Unlike the discrete version of MountainCar, this environment features both a continuous state space and a continuous action space.

Observation Space

The state of the environment is represented as a two-dimensional continuous vector:

$$s = (p, v) \in \mathbb{R}^2$$

where:

- $p \in [-1.2, 0.6]$ is the **position** of the car along the track.
- $v \in [-0.07, 0.07]$ is the **velocity** of the car.

The agent starts at a random position within the range $p \in [-0.6, -0.4]$ with velocity $v = 0$. The goal is to drive up the right hill and reach $p \geq 0.45$.

Action Space

Unlike the discrete version of MountainCar, the action space in MountainCarContinuous is **one-dimensional and continuous**:

$$a \in [-1.0, 1.0].$$

The action represents the continuous force applied to the car, where:

- $a = -1.0$ applies maximum force to the left.
- $a = 1.0$ applies maximum force to the right.
- Intermediate values apply partial force in either direction.

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 `MountainCarContinuous-v0` environment presents a classic control problem where an agent must drive a car up a steep hill. The challenge lies in overcoming the gravitational force by applying continuous

control actions. Since excessive force application is inefficient, an optimal policy must leverage the environment's natural dynamics, including momentum and gravity, to minimize energy consumption.

Equations of Motion

The movement of the car follows the update equations:

$$p_{t+1} = p_t + v_{t+1}$$

$$v_{t+1} = v_t + 0.0015a_t - 0.0025\cos(3p_t),$$

where:

- p_t is the position of the car at time step t.
- v_t is the velocity of the car at time step t.
- $a_t \in [-1, 1]$ is the continuous control action applied by the agent.
- The term $0.0025\cos(3p_t)$ represents the gravitational force acting against the motion.

The goal is to reach $p \geq 0.45$ while minimizing the energy consumption associated with force application.

Energy-Efficient Strategy

Instead of applying maximum force in a single direction, an optimal agent builds momentum gradually by oscillating between small forces in both directions. The energy cost can be defined as:

$$E = \sum_{t=0}^{T} |a_t|$$

where minimizing E ensures that the agent uses the least amount of force necessary to reach the goal.

To maximize momentum transfer efficiently, the agent follows these steps:

1. The agent first moves slightly in the negative direction ($a_t < 0$) to reach the left side of the valley, increasing the potential energy stored in the system. 2. When reaching the lowest point ($p \approx -0.6$), the agent applies a small positive force ($0 < a_t < 0.5$) to accelerate up the right slope. 3. The motion is repeated with small adjustments, where the velocity increases due to the energy gained from previous oscillations. The velocity at each peak satisfies:

$$v_{\text{max},t+1} > v_{\text{max},t}$$

ensuring that kinetic energy accumulates with each oscillation.

4. When the car's velocity reaches a sufficiently high value ($v_t \approx 0.07$), the agent applies a final force $a_t = 1.0$ at the right moment, allowing the car to clear the hill efficiently.

Optimal Policy and Control Constraints

For the most energy-efficient trajectory, the agent should follow the control law:

$$a_t = \begin{cases} -c, & p_t > 0, \quad v_t < 0 \text{ (reverse for momentum)} \\ +c, & p_t < 0, \quad v_t > 0 \text{ (accelerate with gravity)} \\ 1.0, & v_t \geq 0.07 \text{ (final push)} \end{cases}$$

where c is a small positive constant, typically $0.3 \leq c \leq 0.5$, ensuring that the agent does not apply excessive force at any given moment.

We conclude that an energy-efficient policy in MountainCarContinuous does not rely on brute-force acceleration but instead strategically builds momentum through oscillatory motion. By alternating small control forces and applying a final push when velocity is maximized, the agent reaches the goal with minimal energy expenditure. The fundamental principle is to exploit gravity and kinetic energy rather than constantly applying maximum force, making this approach optimal for energy-efficient control.

It is also notable that implementing a correct and proper policy network is really important in achieving a good policy. For example we used and independent `nn.Parameter` for `std` to make standard deviation independent of the state space. This leads the agent to learn better policies.

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)

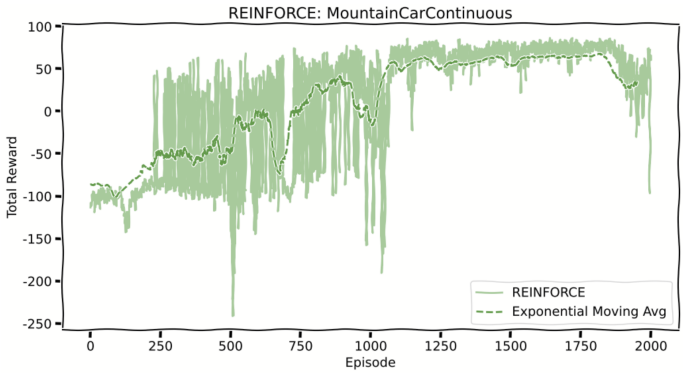

Figure 4: Reward per Episode

Catastrophic forgetting is a common issue in reinforcement learning, particularly in continuous action space environments like `MountainCarContinuous-v0`. It occurs when an agent abruptly loses previously learned knowledge due to unstable updates in the policy or value function. This problem is exacerbated in policy gradient methods, where parameter updates depend on sampled trajectories. To address this, several techniques can be employed to stabilize learning and retain past knowledge effectively which are mentioned as below:

Experience Replay

Experience replay is a widely used technique in reinforcement learning to reduce catastrophic forgetting. Instead of updating the policy using only recent experiences, the agent stores past transitions in a replay buffer and samples them randomly during training.

The replay buffer contains tuples:

$$(s_t, a_t, r_t, s_{t+1})$$

where:

- s_t is the observed state at time step t.
- a_t is the continuous action taken.
- r_t is the reward received.
- s_{t+1} is the next state.

Updating the policy using a mixture of old and new experiences prevents the network from overfitting to recent, possibly misleading, transitions. In continuous control environments like MountainCarContinuous, experience replay allows the agent to retain successful motion patterns while continuing to explore new strategies.

Target Networks

In value-based methods, using target networks helps stabilize learning by reducing the impact of sudden changes in the learned value function. Instead of updating the critic network directly, a separate target network is used to generate stable learning targets:

$$y_t = r_t + \gamma Q_{\theta'}(s_{t+1}, \mu_{\theta'}(s_{t+1}))$$

where $Q_{\theta'}$ and $\mu_{\theta'}$ are the target networks for the critic and actor, respectively. The parameters of the target networks are updated slowly using soft updates:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta',$$

where $\tau \ll 1$ (e.g., 0.005), ensuring that learning remains stable. Target networks prevent large, sudden shifts in value estimates that could destabilize policy learning.

Regularization and Adaptive Learning Rates

Another way to prevent catastrophic forgetting is by introducing weight regularization techniques such as L2 weight decay, which constrains the magnitude of policy updates. Additionally, using adaptive learning rate optimizers such as Adam helps mitigate instability by adjusting the step size based on recent gradient trends.

Entropy Regularization for Policy Stability

In policy gradient methods, entropy regularization is used to encourage exploration and prevent premature convergence to suboptimal policies. The entropy of the policy distribution is given by:

$$H(\pi) = -\sum_a \pi(a \mid s) \log \pi(a \mid s).$$

Maximizing entropy ensures that the policy remains diverse and does not forget useful actions that were previously learned.

Conclusion

Catastrophic forgetting is a significant challenge in reinforcement learning, particularly in continuous action space environments like MountainCarContinuous. Strategies such as **experience replay**, **target networks**, **adaptive learning rates**, and **entropy regularization** play a crucial role in stabilizing learning and ensuring long-term retention of knowledge. Combining these techniques allows an agent to balance exploration and exploitation while maintaining policy stability in complex reinforcement learning environments.

4 Task 4: Policy Gradient Drawbacks [25]

We start with explaining DQN hyperparameters and their effect.

DQN Hyperparameter Choices and Their Importance

- **Learning Rate (α)**:

$$\alpha = 6 \times 10^{-4}$$

  This controls the step size in updating network weights. A low value ensures stable updates, while a high value may cause divergence. This setting balances convergence speed and stability.

- **Discount Factor (γ)**:

$$\gamma = 0.95$$

  The discount factor determines how much future rewards influence current decisions. However, in Frozen Lake, rewards are sparse (mostly zero), making γ less impactful on early rewards than in environments with frequent rewards. So we choose a high γ to take the sparsity of the reward into account.

- **Exploration Strategy (ϵ-Greedy)**: Exploration is crucial in early training stages, where the agent has little knowledge of the environment.

  - **Initial Exploration Rate (ϵ_{max})**:

$$\epsilon_{\text{max}} = 1.0$$

    This ensures that the agent starts with full exploration, taking random actions to gather diverse experience.

  - **Final Exploration Rate (ϵ_{min})**:

$$\epsilon_{\text{min}} = 0.01$$

    The agent eventually transitions to mostly greedy behavior, reducing randomness while selecting the best-known actions.

  - **Exploration Decay Rate (ϵ_{decay})**:

$$\epsilon_{\text{decay}} = 0.999$$

    This determines how quickly the agent shifts from exploration to exploitation. A slow decay (close to 1) prevents premature convergence to a suboptimal policy, allowing the agent to keep exploring longer. In my opinion this was one of the most important hyperparameters along with learning rate.

- **Batch Size**:

$$\text{batch size} = 64$$

  This defines the number of samples used in each training update. A larger batch size improves gradient estimates, but also increases computational cost. A value of 64 is a good trade-off between stability and efficiency.

- **Experience Replay Capacity**:

$$\text{memory capacity} = 10,000$$

  This determines how many past experiences are stored for off-policy learning. A large replay buffer prevents the model from overfitting to recent experiences and improves sample efficiency.

- **Gradient Clipping**:

$$\text{clip grad norm} = 5$$

  To prevent exploding gradients, which destabilize training, gradient norms are clipped to a maximum value of 5. This ensures smooth training updates.

- **Target Network Update Frequency**:

$$\text{update frequency} = 10$$

  Instead of updating the Q-network directly at each step, the target network is updated every 10 steps to prevent sudden shifts in Q-value estimates. This helps stabilize learning.

- **Training Duration and Step Limits**:
  - max episodes = 3000 (Allows sufficient learning time)
  - max steps per episode = 300 (Encourages agent to explore longer trajectories)

Most Influential Hyperparameters

Among the selected hyperparameters, the following have the greatest impact:

- **Exploration Decay (ϵ_{decay})**: Since the agent gets very sparse rewards it is very important to let it explore enough before reducing the exploration rate. So it is very important to let the agent reach the goal a few times before reducing epsilon.
- **Learning Rate (α)**: Ensures stable updates towards optimal policy.
- **Experience Replay Capacity**: If set too low the model only considers recent trajectories leading to poor performance.
- **Target Network Updates**: Prevents oscillations in Q-value learning by updating less frequently.

The chosen hyperparameters ensure a balance between exploration, stability, and efficiency. In environments like Frozen Lake, where rewards are sparse and transitions are stochastic, careful tuning of **exploration rate, experience replay, and learning rate** is crucial for achieving optimal performance.

REINFORCE Hyperparameter Choices and Their Importance

- **Learning Rate (α)**:

$$\alpha = 0.02$$

  The learning rate determines how aggressively the policy updates after each gradient step. Since REINFORCE suffers from high variance, a moderate learning rate is chosen to balance stability and convergence speed.

- **Discount Factor (γ)**:

$$\gamma = 0.98$$

  For the same reason as in DQN, we have chosen a high value for γ for REINFORCE as well.

- **Hidden Layer Dimensions**:

$$\text{hidden dims} = 128$$

  The policy network is a neural network that approximates the probability distribution over actions. A hidden layer with 128 neurons provides sufficient expressiveness while keeping computational costs manageable.

- **Step Size for Learning Rate Scheduler**:

$$\text{step size} = 300$$

  This controls how frequently the learning rate is adjusted during training. Adjusting the learning rate dynamically allows for more stable convergence preventing overly aggressive updates late in training.

- **Number of Training Episodes**:

$$\text{episodes} = 1000$$

  The total number of episodes in training gives the agent enough time to explore.

Note that despite trying multiple hyperparameters the REINFORCE agent does not learn anything. This is explained in the following sections.

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?

Performance Comparison

- **Training Stability**: The reward curve for DQN demonstrates steady improvement, with fluctuations stabilizing as training progresses. This indicates consistent learning and effective policy updates. In contrast, REINFORCE failed to learn a successful policy, likely due to high variance in policy updates and the absence of experience replay. Despite trying multiple parameters for REINFORCE it did not learn and the reward is stuck at zero.
- **Convergence Speed**: The DQN agent initially exhibits random exploration but gradually improves, reaching high success rates after approximately 1000 episodes. On the other hand, REINFORCE does not show meaningful learning progress, likely due to difficulty in handling sparse rewards and stochastic transitions.
- **Overall Success Rate**: The DQN reward curve stabilizes near 1.0, indicating that the agent successfully reaches the goal in most episodes. In contrast, REINFORCE fails to learn a viable policy, leading to a low success rate.

Why DQN Performs Better in Frozen Lake

- DQN leverages experience replay, allowing the agent to reuse past experiences and improve sample efficiency.
- DQN uses Q-value estimation, which is more effective in discrete action spaces, enabling the agent to determine the best action for each state.
- Policy Gradient methods struggle with sparse rewards because they require complete trajectories to compute updates, leading to high variance in learning.
- Frozen Lake is a discrete, low-dimensional environment, which inherently favors value-based methods like DQN over policy gradient approaches.

Figure 5: DQN reward per episode

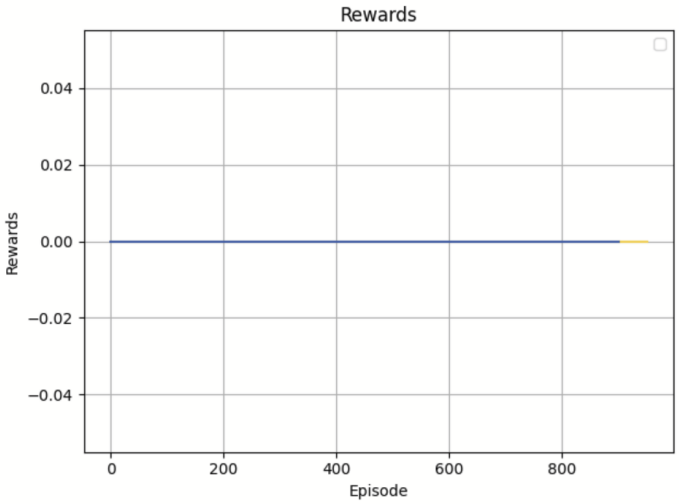

Figure 6: REINFORCE reward per episode

Conclusion

Based on the observed results (Figures (5) and (6)):

- DQN significantly outperforms REINFORCE in Frozen Lake.
- DQN achieves higher training stability, faster convergence, and a near-optimal success rate.
- REINFORCE struggles due to high variance and sparse rewards, making it unsuitable for this environment.

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 environment is particularly challenging due to **sparse rewards**, **stochastic transitions** (Although

`is_slippery` parameter is set to false here.), and the need for **long-term planning**. These factors significantly impact the learning process, affecting the performance of both Deep Q-Network (DQN) and Policy Gradient (REINFORCE) methods.

Key Challenges in the Frozen Lake Environment

- **Sparse Rewards**: The agent receives a reward only when it successfully reaches the goal, with all other transitions providing zero reward. This makes learning difficult because there is **no immediate feedback** to guide the agent in the right direction. As a result, algorithms struggle to distinguish good actions from bad ones.
- **Long-Term Planning**: Since rewards are received only upon reaching the goal, the agent must learn a sequence of actions that leads to success over multiple steps. This makes the problem more difficult for reinforcement learning algorithms, especially those relying on greedy strategies.
- **Exploration vs. Exploitation Dilemma**: Due to the large state space in the 8×8 version of Frozen Lake, the agent must explore extensively before discovering a viable policy. However, too much exploration can lead to excessive failures, while premature exploitation of a suboptimal policy can delay learning.

Impact on Deep Q-Network (DQN)

- DQN can mitigate the sparse reward problem by using **experience replay**, which allows it to learn from past successful trajectories even when rewards are rare.
- Since DQN relies on Q-value maximization, it can struggle with exploration in large state spaces, making the ϵ-greedy strategy critical for ensuring proper exploration.

Impact on Policy Gradient (REINFORCE)

- The high variance of policy gradient updates makes it extremely difficult for REINFORCE to learn in environments with sparse rewards, as policy updates depend on complete trajectories.
- Since REINFORCE is an **on-policy** method, it does not use experience replay, meaning that **every episode must provide useful learning signals**, which is rarely the case in Frozen Lake.

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.

Factors Influencing Algorithm Choice

- **Sample Efficiency**:
  - DQN is generally sample-efficient because it uses experience replay to learn from past transitions, reducing the need for new interactions. But here since we can have unlimited interactions, being able to store past trajectories is not very crucial.

  - Policy Gradient methods, such as REINFORCE, despite being on-policy does not need to worry about limited interaction with the environment. We are able to sample infinitely many times with low-cost, making on-policy methods mitigate the issue of sample efficiency.

- **Function Approximation**:
  - DQN relies on Q-value approximation using neural networks, which can suffer from instability due to overestimation biases and divergence in function approximation.
  - Policy Gradient methods, particularly those employing actor-critic architectures, provide more stable convergence by directly optimizing the policy. When interactions are unlimited, Policy Gradient methods can take advantage of more extensive training data to optimize continuous action policies effectively.
- **Stability of Learning**:
  - DQN suffers from instability issues due to bootstrapping and function approximation errors. Techniques such as target networks and experience replay mitigate these issues but do not completely eliminate them.
  - Policy Gradient methods, despite their higher variance, tend to have more stable learning dynamics when trained with sufficient data. With unlimited interactions, variance can be reduced using variance-reducing techniques like baseline functions and advantage estimation.

Suitability in Unlimited Interaction Environments

Given an environment with unlimited interactions and low-cost sampling, Policy Gradient methods are generally more advantageous. The primary reasons include:

- Policy Gradient methods can optimize stochastic policies directly, which is beneficial for environments with high-dimensional or continuous action spaces.
- With unlimited data, the high variance of Policy Gradient methods becomes less problematic as extensive sampling reduces the variance of gradient estimates.
- The lack of need for an explicit Q-value function mitigates issues related to bootstrapping and function approximation errors, leading to more stable learning in the long run.

Conclusion

In environments where interactions are unlimited and sampling is inexpensive, Policy Gradient methods are preferable due to their ability to optimize stochastic policies effectively, avoid instability issues associated with Q-learning, and leverage extensive data to reduce variance. While DQN remains a strong candidate in sample-constrained environments, Policy Gradient methods benefit significantly from unlimited data, making them the better choice in such settings.

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/