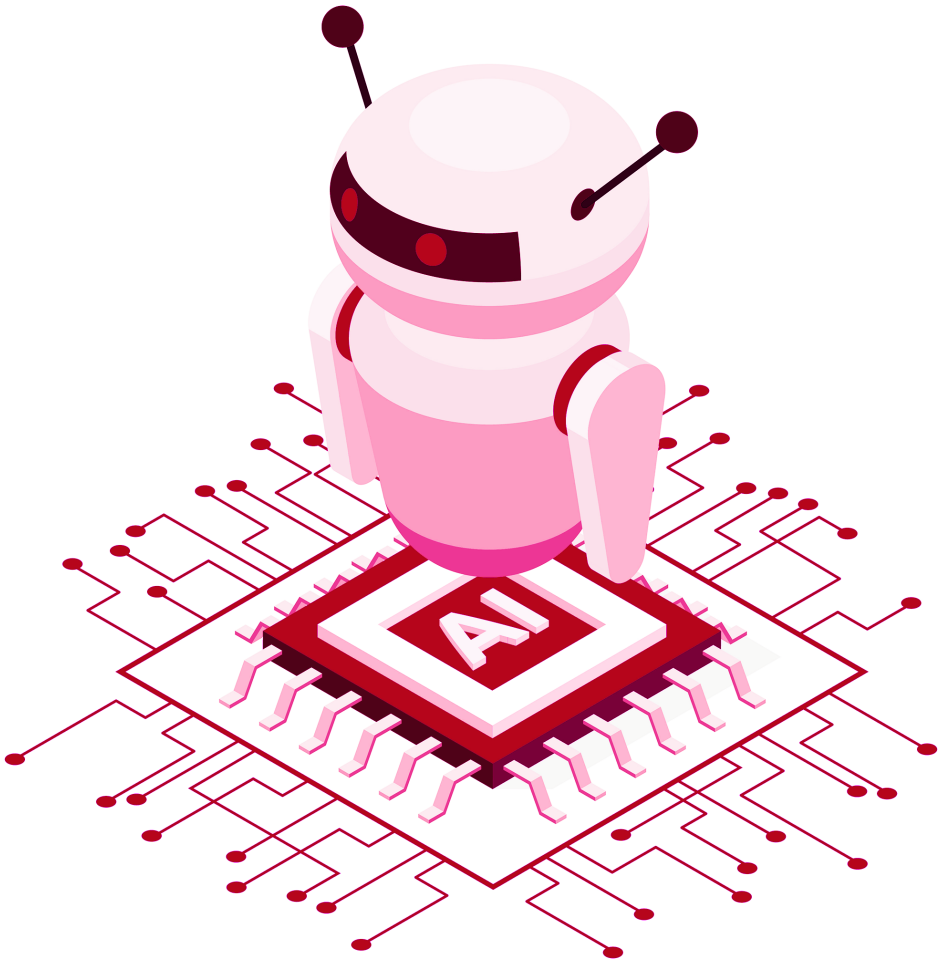

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 3:

Policy-Based Methods

By:

Ashkan Majidi

400109984

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]

1.1 Question 1:

How do these two methods differ in terms of their effectiveness for solving reinforcement learning tasks?
To evaluate the properties of each algorithm in solving reinforcement learning tasks, several factors must be considered, including sample efficiency, the balance between exploration and exploitation, and convergence behavior. Here's a breakdown of their key differences:

REINFORCE is a gradient-based policy optimization method that updates parameters using policy gradients, adjusting them in the direction of increased expected rewards. It is generally more sample-efficient and works well in structured environments where smooth gradient updates can guide policy improvement.

Genetic Algorithms (GA), on the other hand, are derivative-free methods that evolve a population of policies over generations using selection, crossover, and mutation. GA is useful when the reward function is non-differentiable or highly deceptive, where gradient-based methods struggle. It provides better global exploration but at the cost of higher computational expense and slower convergence. Unlike REINFORCE, which refines a single policy, GA explores a diverse range of policies, making it more robust against local optima but less sample-efficient.

As mentioned before, a major distinction between the two methods is that REINFORCE focuses on refining a single policy incrementally, whereas GA explores a diverse set of policies simultaneously. REINFORCE is generally more effective in structured environments where a smooth optimization landscape allows gradient-based improvement, while GA excels in problems where direct gradient information is unavailable or misleading. The choice between the two methods depends on the nature of the reinforcement learning task, with REINFORCE being preferred for fine-tuning policies and GA being advantageous for exploration-heavy or rugged reward landscapes.

In our problem, which is more structured, REINFORCE generally achieves better policies.

1.2 Question 2:

Discuss the key differences in their performance, convergence rates, and stability.
To evaluate the performance, convergence rates, and stability of REINFORCE and Genetic Algorithm (GA) in solving a grid-world problem, the following experimental configurations were used:

REINFORCE Algorithm:

- Learning Rate: 0.005
- Discount Factor (γ): 0.99
- Number of Episodes: 6000
- Initial Epsilon: 1.0
- Epsilon Decay: 0.9995

Genetic Algorithm (GA):

- Population Size: 50
- Number of Generations: 50

- Mutation Rate: 0.1
- Crossover Rate: 0.5

Following are the key differences of these methods:

- **Performance:** REINFORCE generally performs well in structured environments with differentiable policies, whereas GA excels in problems where gradients are unavailable or misleading. In our grid world problem as you saw REIFORCE performed much better than GA.
- **Convergence Rates:** In our experiments as we expected, REINFORCE converged faster, but GA needed many generations to solve the problem. Generally, REINFORCE tends to converge faster due to gradient-based updates, while GA often requires many generations to evolve a good policy.
- **Stability:** REINFORCE can suffer from high variance and instability, especially in long-horizon tasks. GA, while robust to local optima, can be unstable if mutation rates are too high or selection pressure is too strong.

1.3 Question 3:

Additionally, explore how each method handles exploration and exploitation, and suggest situations where one might be preferred over the other.
REINFORCE uses stochastic policies to balance exploration and exploitation, but it may get stuck in local optima without proper variance reduction techniques. GA, on the other hand, encourages diverse exploration through population diversity and mutation but lacks explicit exploitation mechanisms. REINFORCE is preferred when policy gradients provide meaningful updates, while GA is advantageous in deceptive landscapes or when exploration is more critical than exploitation.

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, a standard benchmark in reinforcement learning, consists of a pole attached to a moving cart. The observation space is a continuous 4-dimensional vector, including:

- Cart position (x)
- Cart velocity ($\dot{x}$)
- Pole angle (θ)
- Pole angular velocity ($\dot{\theta}$)

The action space is discrete, with two possible actions:

- Move the cart to the left (0)
- Move the cart to the right (1)

2.2 Question 2:

What is the role of the discount factor (γ) in reinforcement learning, and what happens when $\gamma = 0$ or $\gamma = 1$?
The discount factor (γ) determines the importance of future rewards in reinforcement learning. It is used in value estimation to weigh immediate rewards versus future rewards:

- When $\gamma = 0$, the agent only considers immediate rewards, ignoring future consequences. This results in a greedy policy that may not be optimal.
- When $\gamma = 1$, the agent considers all future rewards equally, leading to long-term planning. However, this can cause instability or slow convergence.

2.3 Question 3:

Why is a baseline introduced in the REINFORCE algorithm, and how does it contribute to training stability?
A baseline, often represented as a value function (e.g., $V(s)$), is introduced in REINFORCE to reduce variance in gradient estimates while keeping the expected value unchanged. It helps stabilize training by:

- Reducing fluctuations in policy updates, leading to smoother learning.
- Improving sample efficiency by minimizing unnecessary high-variance updates.
- Enhancing convergence speed by providing more reliable gradient estimates.

The most common choice for a baseline is the state-value function, where the advantage function $A(s, a) = Q(s, a) - V(s)$ is used instead of the raw return.

2.4 Question 4:

What are the primary challenges associated with policy gradient methods like REINFORCE?
Policy gradient methods, including REINFORCE, face several key challenges:

- **High Variance:** Estimating gradients from sampled trajectories can lead to noisy updates, making training unstable.
- **Sample Inefficiency:** Since updates are based on entire trajectories, many samples are required for effective learning.
- **Credit Assignment:** Determining which actions contributed most to the reward is difficult, especially in long-horizon problems.
- **Slow Convergence:** Policy updates depend on stochastic gradients, requiring many episodes to improve performance.

Note that there are many ways to mitigate these problems like using baseline, not the entire trajectory, etc.

2.5 Question 5:

Based on the results, how does REINFORCE with a baseline compare to REINFORCE without a baseline in terms of performance?
Following are the Returns of both algorithms in our problem.

```
No Baseline: mean_reward = 363.98 +/- 175.6248
With Baseline: mean_reward = 437.33 +/- 136.1609
```

As you can see REINFORCE with baseline achieves better results which are more stable but let's have look at its learning curve.

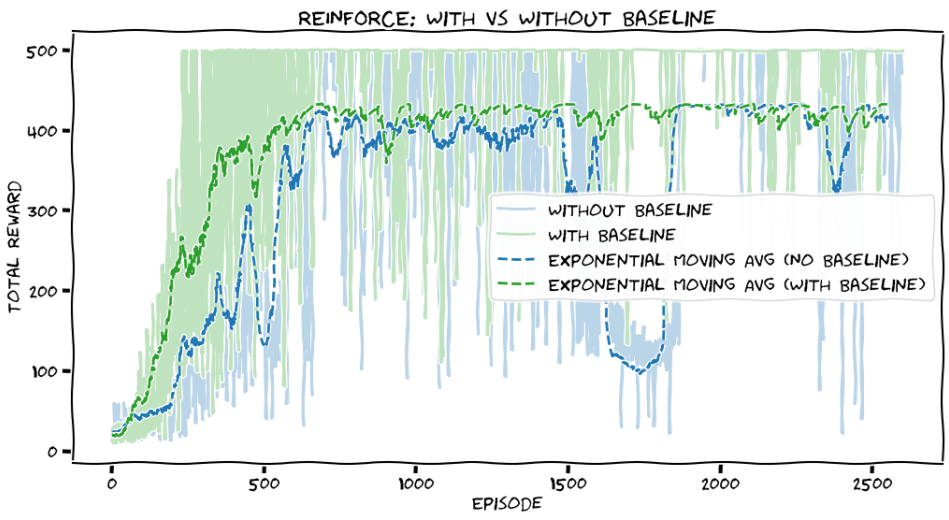

Figure 1: Learning Curve

As you can see REINFORCE with baseline have faster convergence compared with no baseline version

Experimental results show that REINFORCE with a baseline generally outperforms the version without a baseline in terms of:

- **Faster Convergence:** The variance reduction from the baseline results in smoother and more stable policy updates.
- **Higher Stability:** Training is less prone to sudden performance drops due to noisy updates.
- **Improved Sample Efficiency:** The baseline improves the effectiveness of each trajectory, reducing the number of episodes required to reach optimal performance.

2.6 Question 6:

Explain how variance affects policy gradient methods, particularly in the context of estimating gradients from sampled trajectories.
Variance plays a crucial role in policy gradient methods like REINFORCE, as it affects the reliability of gradient estimates. High variance arises from:

- The stochastic nature of trajectory sampling, leading to inconsistent policy updates.
- Reward signals being sparse or delayed, making it difficult to attribute rewards to specific actions.
- The use of Monte Carlo estimates, which inherently have high variance compared to bootstrapped methods like actor-critic.

To mitigate variance, techniques such as baselines, reward normalization, and variance reduction algorithms (e.g., Generalized Advantage Estimation) are commonly used to improve training stability and convergence speed.

3 Task 3: REINFORCE in a Continuous Action Space [20]

Before answering questions there are few notes I wanted to mention. Learning in MountainCarContinuous environment was a little tricky, Additionally I had few computational resources which made this harder (please consider this in later HWs). My model was not learning enough, I tracked the gradients and it turned out that I'm experiencing **Vanishing Gradients** so to overcome this problem I added a **Skip Connection** to my models architecture and as you see in following learning curve and gradient norme the problems sovled successfully!

```
log_std gradient norm: 1834.530517578125
fc1.weight gradient norm: 314.7116394042969
fc1.bias gradient norm: 1256.6617431640625
fc2.weight gradient norm: 694.0494995117188
fc2.bias gradient norm: 202.368408203125
mean.weight gradient norm: 4418.39794921875
mean.bias gradient norm: 394.18157958984375
```

```
Episode 1300, Reward: 77.33462370586754
```

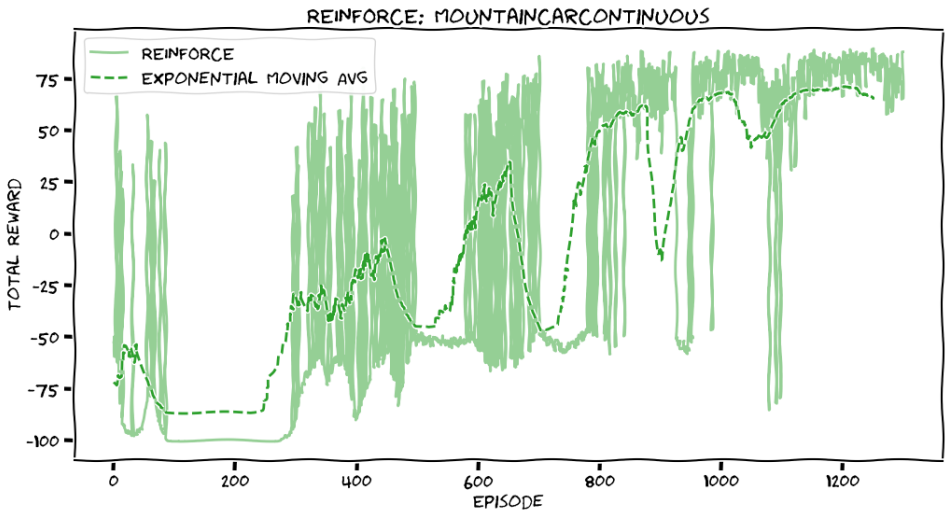

Figure 2: Learning Curve

As you saw our agent learned well and can solve the problem (see the video attached in notebook).

3.1 Question 1:

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

The MountainCarContinuous environment consists of a car that must reach the goal by building enough momentum to overcome a valley. The environment is defined as follows:

- **Observation Space:** A continuous 2-dimensional space representing:

  - Car position (x), ranging from $[-1.2, 0.6]$
  - Car velocity ($\dot{x}$), ranging from $[-0.07, 0.07]$
- **Action Space:** A continuous 1-dimensional space where the agent applies a force between $[-1, 1]$ to accelerate left or right.

The goal is to drive the car up the hill using an optimal sequence of force applications to minimize energy expenditure.

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 the most optimal policy.
To reach the goal with minimal energy usage, the agent must leverage the physics of the environment by building momentum efficiently. An optimal policy follows these principles:

- Instead of applying maximum force in one direction, the agent starts by applying small forces in alternating directions to build kinetic energy.
- It synchronizes acceleration with the car's natural oscillations, increasing force application when moving in the desired direction and minimizing it when moving against gravity.
- By gradually amplifying the swings, the car accumulates sufficient speed to reach the top without excessive energy usage.
- Once the car reaches the goal state at $x = 0.5$, the episode terminates.

This energy-efficient approach contrasts with a naive policy that applies maximum force continuously, which is inefficient and results in excessive energy consumption.

3.3 Question 3:

What strategies can be employed to reduce catastrophic forgetting in continuous action space environments like MountainCarContinuous?
Catastrophic forgetting occurs when an agent overwrites useful past knowledge due to non-stationary updates. Several strategies can mitigate this issue:

- **Experience Replay:** Storing past experiences in a replay buffer and sampling from it randomly helps break correlations between consecutive updates, leading to more stable learning.
- **Target Networks:** Maintaining a separate, slowly updating target network prevents drastic changes in the policy, reducing instability.

By employing these strategies, an agent can learn more robust policies in continuous action environments without forgetting previously successful strategies.

4 Task 4: Policy Gradient Drawbacks [25]

Experimantal settings of DQN:

```
"clip_grad_norm"        : 3.5,
"learning_rate"         : 8e-4,
"discount_factor"       : 0.935,
"batch_size"            : 64,
"update_frequency"      : 10,
"max_episodes"          : 7000
"max_steps"             : 300,
"epsilon_max"           : 0.999
"epsilon_min"           : 0.01,
"epsilon_decay"         : 0.999,
"memory_capacity"       : 10000
```

Figure 3: DQN Leraning Curve

4.1 Question 1:

Which algorithm performs better in the Frozen Lake environment? Why?
As expected the performance of DQN was better than REINFORCE in the Frozen Lake environment. In following we compare their performance based on factors such as training stability, convergence speed, and success rate.

- **Training Stability:** DQN tends to be more stable since it uses experience replay and a target network to reduce variance in updates. In contrast, REINFORCE exhibits higher variance due to policy updates based on sampled trajectories.
- **Convergence Speed:** DQN usually converges faster as it employs value-based learning with bootstrapping, enabling efficient credit assignment. REINFORCE, relying on Monte Carlo estimates, often requires more episodes for convergence.
- **Overall Success Rate:** In a stochastic environment like Frozen Lake, where slip probabilities introduce randomness, DQN generally performs better due to its value-based approach, which optimizes state-action values rather than relying purely on policy updates.

Based on these observations, DQN achieves better results in Frozen Lake due to its stability, faster convergence, and ability to generalize better in stochastic settings.

4.2 Question 2:

What challenges does the Frozen Lake environment introduce for reinforcement learning?
The Frozen Lake environment presents several challenges that make learning difficult for reinforcement learning algorithms:

- **Stochastic Transitions:** Due to the probability of slipping, actions do not always lead to expected outcomes, increasing uncertainty in learning.
- **Sparse Rewards:** The environment provides rewards only upon reaching the goal, making it difficult for algorithms to assign credit to beneficial actions.
- **Long-Term Credit Assignment:** Since the agent must navigate multiple steps without immediate feedback, it becomes harder for policy gradient methods to learn optimal trajectories efficiently.
- **Exploration vs. Exploitation:** The presence of holes in the grid requires careful exploration, as random exploration may lead to suboptimal policies or excessive failures.

These challenges make Frozen Lake particularly difficult for policy gradient methods like REINFORCE, as they rely on full trajectory rewards for updates. DQN, with value function approximation, can handle these challenges better by learning optimal Q-values over multiple episodes.

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, Policy Gradient methods like REINFORCE can be more advantageous than DQN:

- **Sample Efficiency:** DQN is more sample-efficient in environments with limited interactions due to experience replay. However, with unlimited sampling, this advantage diminishes.
- **Function Approximation:** Policy Gradient methods directly optimize the policy and can work well in high-dimensional continuous action spaces, whereas DQN struggles with such environments.
- **Learning Stability:** DQN is prone to overestimation errors and instability due to function approximation biases, while policy gradients, despite their variance, benefit from direct policy optimization.

Thus, for environments with unlimited sampling and low-cost interactions, Policy Gradient methods become more favorable, especially for complex action spaces where function approximation plays a crucial role.

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/