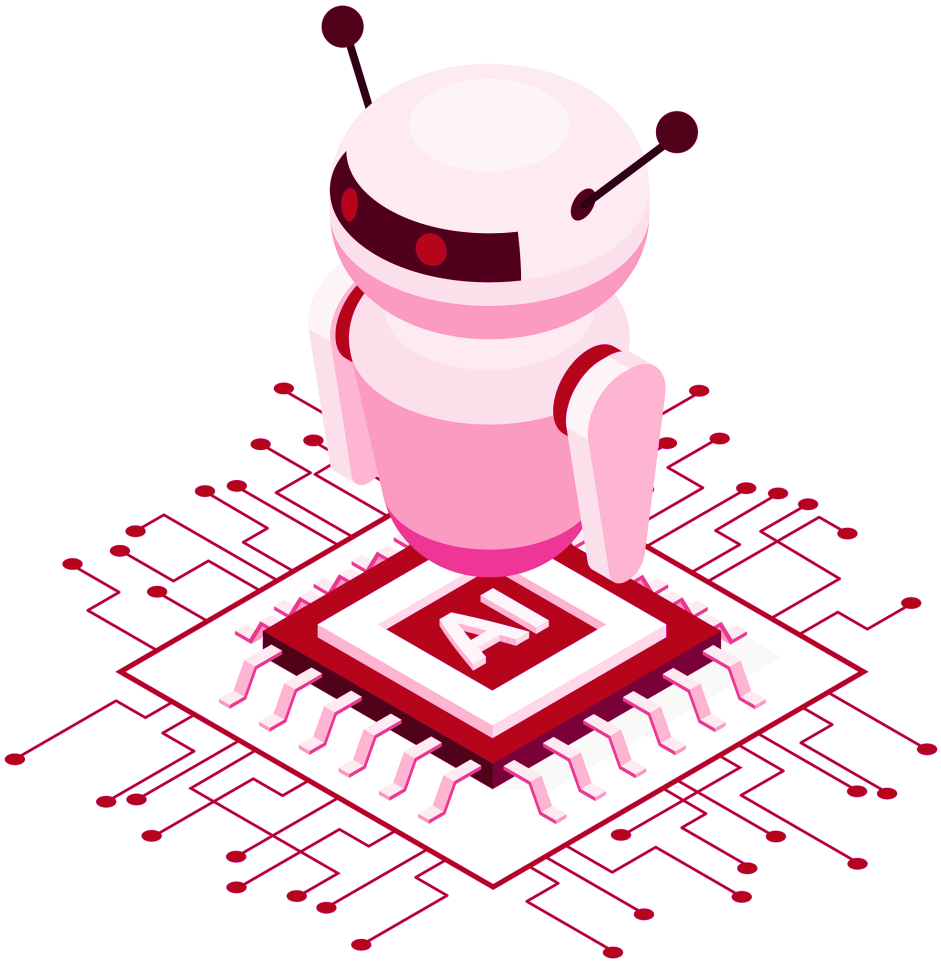

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 3:

Policy-Based Methods

By:

Ali Ghasemzadeh

401106339

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?
REINFORCE is a policy gradient method that directly adjusts a neural network's parameters by following the gradient of expected rewards. it works well when your policy is differentiable and can handle continuous actions.
GENETIC treat each policy as a "chromosome" and evolve a population over generations. They don't need gradient information and can handle weird, non-differentiable setups or highly discrete spaces.

1.2 Question 2:

Discuss the key differences in their **performance**, **convergence rates**, and **stability**.
as I run the codes i see that reinforce takes less time and converge better and genetic algorithm takes more time and it should contains a big number of populations to converge good and this cause a long time to train. but Reinforce can be unstable or got stuck if learning rates or hyperparameters are off.

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 a stochastic policy for exploration but might reduce exploration too early without extra methods (like entropy bonuses).
GAs naturally explore through mutation and crossover but risk losing diversity if selection is too strong.

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 is s: which is a vector of horizontal position of the cart, horizontal velocity of the cart, angle of the pole relative to the vertical, the angular velocity of the pole. horizontal position $\in$ [-4.8,4.8] and angle of the pole relative to the vertical $\in$ [-24, 24] degree.
Action space consists of two actions and so it is set(0, 1)
0 -> left move
1 -> right move

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?
γ determines how much the future rewards matter and if $\gamma = 0$ then only the immediate rewards matter and if the $\gamma = 1$ then all the future rewards considered equally so the high amount of γ is used for log-term strategies.

2.3 Question 3:

Why is a baseline introduced in the REINFORCE algorithm, and how does it contribute to training stability?
A baseline is essentially a reference level of expected reward used to subtract from the return before computing policy gradients.
This subtraction reduces the variance of the gradient estimates. In other words, we're more accurately judging how much better or worse an action's outcome was compared to an "average" scenario.
By lowering variance, training becomes more stable and typically converges faster. The policy updates become more consistent, rather than swinging wildly with each sampled trajectory.

2.4 Question 4:

What are the primary challenges associated with policy gradient methods like REINFORCE?
High Variance: Estimating gradients from random trajectories can be noisy, making learning unstable.
Sample Inefficiency: It often takes many episodes to gather enough data to produce stable gradient estimates.
Sensitivity to Hyperparameters: Learning rate, batch size, and discount factor can drastically affect results.
Local Optima Exploration: Pure policy gradients can get stuck if the policy doesn't explore sufficiently.

2.5 Question 5:

Based on the results, how does REINFORCE with a baseline compare to REINFORCE without a baseline in terms of performance?
With a Baseline: Usually shows faster learning and more stable performance. The updates aren't as noisy, so the agent can home in on a good policy more reliably.
Without a Baseline: Tends to have larger swings in performance and can take longer to converge because it's more affected by random fluctuations in the returns.

2.6 Question 6:

Explain how variance affects policy gradient methods, particularly in the context of estimating gradients from sampled trajectories.
Policy gradient methods like REINFORCE rely on sampled trajectories of states, actions, and rewards. Each trajectory can produce a very different return, leading to high variance in the gradient estimate.
High variance means unstable updates—the policy might change drastically from one batch of data to the next.
Reducing variance (for example, by subtracting a baseline or using variance-reduction tricks) makes the learning updates more consistent and helps the policy converge faster and more reliably.

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: A continuous 2D vector:
Car Position (where the car is on the track, typically between 1.2, 0.6)
Car Velocity (how fast the car is moving, typically between 0.07 and 0.07)
action spaec is in range [-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.
In MountainCarContinuous, the car starts at the bottom of a valley and must drive up a hill on the right to reach the goal. The trick is that the engine isn't powerful enough to drive straight up the slope from a standstill, so the agent must build momentum:
Initial "Swing": The agent might accelerate left (negative force) to roll the car backward, building up potential energy.
Momentum Build-Up: Then it switches to right (positive force) at the correct time, using the momentum gained from swinging backward to help push the car up the right hill.
Minimal Energy Usage: The agent applies just enough force to crest the hill—avoiding constant throttle to prevent energy waste.
A well-timed "back-and-forth" motion is more energy-efficient than just brute-forcing the engine.
With an optimal policy, the car typically makes one or two swings to gain momentum, then accelerates up the hill to the goal without wasting energy on unnecessary revving.

3.3 Question 3:

What strategies can be employed to reduce catastrophic forgetting in continuous action space environments like MountainCarContinuous?
Experience Replay: Store past transitions (s,a,r,s) in a replay buffer
Sample mini-batches randomly to break correlation between consecutive updates and revisit older experiences.
Target Networks: Keep a separate set of network parameters (the "target") updated more slowly than the main network.
Stabilizes learning because the target doesn't change too rapidly, reducing the chance of forgetting previously learned behaviors.

(Hint: experience replay or target networks)

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?
DQN tends to perform better on Frozen Lake (especially when is$_s lippery = False), because it learns a direct action-value function in a relatively small, deterministic environment.$
$REINFORCE can still solve the task, but it might be less stable and require more careful tuning. as I used optuna f$

Training Stability and Convergence: DQN is usually more stable in small, discrete environments. It quickly converges to the optimal policy by learning Q(s,a) values.
REINFORCE can have higher variance in its gradient updates and might take longer to converge without additional tricks (like baselines or careful hyperparameter settings).
Success Rate: In a deterministic setting (no slipperiness), DQN often finds the optimal path faster and more consistently.
REINFORCE can succeed but may fluctuate or take longer to reach high success rates.(but here it doesn't find)

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?
Sparse Rewards: The agent only gets a reward upon reaching the goal. This means many episodes end with no reward, making it hard to figure out which actions were correct.
State Space vs. Action Space: Though the grid is small, exploration can still be tricky. Without proper exploration, the agent may never find the correct route to the goal.
in summary :
DQN: Tends to handle discrete, small state spaces well, but can still struggle if exploration is insufficient.
REINFORCE: The high variance of policy gradient methods makes sparse rewards more challenging. If the agent rarely sees the goal reward, gradient updates can be noisy and slow.

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.
Unlimited Interactions and Low-Cost Sampling:
Policy Gradient methods (like REINFORCE) can shine when you can gather huge amounts of experience. The more trajectories you sample, the more accurate your gradient estimates become.
DQN is generally more sample-efficient in smaller state-action spaces, but it can also benefit from lots of

data.
DQN is used when :
Often more stable in discrete action spaces.
May converge faster when the state-action space is not too large.
Policy Gradient (REINFORCE):
With unlimited data, you can reduce gradient variance significantly.
Particularly advantageous in continuous or large action spaces, or where function approximation can be more direct via policies.
If interactions are truly unlimited and sampling is cheap, policy gradient methods can eventually do very well, because the large amount of data helps average out noisy gradients.
However, in a small, discrete environment like Frozen Lake, DQN often remains simpler and more stable—even if you can gather a lot of experience.

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/