Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Solution for Homework 3:

Policy-Based Methods

Designed By:

Nima Shirzady

shirzady.1934@gmail.com

SeyyedAli MirGhasemi

sam717269@gmail.com

RIML

Spring 2025

Contents

1 Task 1: Policy Search: REINFORCE vs. GA

• Based on the implementation and results from comparing policy search using Genetic Algorithm (GA) and the REINFORCE algorithm:

1.1 Question 1 (10 Points)

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

Solution:

Underlying Principles

Genetic Algorithm (GA):

- Operates on a *population* of policies (parameter sets).
- Evaluates each policy by measuring its return or reward in the environment.
- Uses *selection*, *mutation*, and *crossover* to generate new candidate solutions.
- *Does not rely on gradients*; only needs a scalar fitness (reward) signal.

REINFORCE:

- A *policy-gradient* method optimizing a single parameterized policy.
- Relies on *stochastic gradient ascent* to improve the expected return.
- Iteratively *samples trajectories*, computes returns, and updates policy parameters using the policy gradient.

Effectiveness

Genetic Algorithm (GA):

- Broad, population-based exploration of the policy space.
- Potential to find globally optimal policies given sufficiently many generations and robust population diversity.
- Robust to *non-differentiable* or noisy reward signals.

REINFORCE:

- Leverages *direct gradient* information to move towards higher-reward policies more efficiently.
- Generally more *sample-efficient* if reward signals are well-shaped or dense.
- Commonly used in large-scale RL with function approximators (e.g., neural networks) for faster convergence.

1.2 Question 2 (15 Points)

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

Solution:

Performance

Genetic Algorithm (GA):

- Success depends heavily on maintaining *population diversity*.
- Can *avoid local minima* via random mutations and crossover.
- May converge to a *suboptimal* policy if the population loses diversity too early.

REINFORCE:

- Often achieves *strong final performance* when hyperparameters (learning rate, baseline, etc.) are well-tuned.
- Can get stuck in local maxima or suffer high variance if misconfigured.
- Excels in *continuous or dense* reward environments when gradient signals are stable.

Convergence Rates

Genetic Algorithm (GA):

- Requires *many policy evaluations* each generation across the population.
- May *take longer* in total wall-clock time, especially for large populations or complex tasks.

REINFORCE:

- Utilizes *gradient-based* updates for more direct optimization.
- Converges faster in terms of *number of episodes* if reward signals are adequately dense.
- Sensitive to hyperparameters: with proper tuning, it often *outperforms GA* in runtime.

Stability

Genetic Algorithm (GA):

- Policy changes can be large from generation to generation due to crossover and mutation.
- Once a strong sub-population emerges, convergence can appear stable but may *prematurely fixate* on a suboptimal solution if diversity is not maintained.

REINFORCE:

- Uses local gradient steps, so convergence is typically *smoother* with an appropriate learning rate.
- Variance-reduction techniques (e.g., baselines) improve stability.
- May *oscillate* if the learning rate is too high or if the reward signal is noisy without proper tuning.

1.3 Question 3 (15 Points)

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

Solution:

Exploration vs. Exploitation

Genetic Algorithm (GA):

- *Exploration* occurs via random mutations and crossover, which can yield novel parameter combinations.
- *Exploitation* is guided by selection pressure favoring high-fitness individuals.

REINFORCE:

- *Exploration* handled by sampling from a *stochastic policy* (e.g., softmax).
- *Exploitation* intensifies as gradient updates shift the policy toward actions with higher expected return.

When to Prefer GA

- *Sparse or deceptive rewards:* GA can discover rare rewarding trajectories that gradient-based methods might miss due to weak gradients.
- *Non-differentiable* or highly stochastic reward structures: GA only needs a scalar fitness value, rather than stable gradients.
- *Parallelization:* If computational resources allow evaluating many policies simultaneously, GA's potentially higher sampling costs can be mitigated.

When to Prefer REINFORCE

- *Dense or well-structured rewards:* Policy-gradient approaches thrive with informative reward signals.
- *Efficiency & speed:* Often converges *faster* to an optimal policy than GA once the problem is well-posed for gradient-based learning.
- *Large-scale policy networks:* REINFORCE (and related actor-critic methods) typically integrate smoothly with neural network optimization techniques.

2 Task 2: REINFORCE: Baseline vs. No Baseline

2.1 Question 1 (5 Points)

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

Solution:

- **Observation Space:**
  - The state in `CartPole-v1` is described by a 4-dimensional continuous vector:

$$[x, \quad \dot{x}, \quad \theta, \quad \dot{\theta}]$$

    where
    - x is the cart position along the track,
    - $\dot{x}$ is the cart velocity,
    - θ is the pole's angular deviation from upright,
    - $\dot{\theta}$ is the angular velocity of the pole.
- **Action Space:**
  - The agent chooses between two discrete actions:

$$\text{Action} \in \{0, 1\}$$

    where typically:
    - 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.

2.2 Question 2 (5 Points)

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

Solution:

- The discount factor $\gamma \in [0, 1]$ determines how future rewards are weighted relative to immediate rewards in the return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

- **When $\gamma = 0$:**
  - The agent cares *only* about the immediate reward R_{t+1} and ignores any future rewards.
  - This can lead to very myopic behavior, potentially sacrificing long-term performance.
- **When $\gamma = 1$:**

  - The agent *fully considers all future rewards* without any discounting, effectively maximizing the sum of rewards over an entire episode (or horizon).
  - This can lead to better long-term strategies but also *increased variance* in gradient estimates, since returns can be larger and more variable over long horizons.
  - In finite-horizon tasks (like CartPole with a time limit), $\gamma = 1$ still means each future reward counts equally until the episode ends.
- For $0 < \gamma < 1$, future rewards are discounted exponentially, providing a balance between immediate and future returns.

2.3 Question 3 (5 Points)

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

Solution:

- Vanilla REINFORCE computes a policy gradient estimate using the returns G_t:

$$\nabla J(\theta) \approx \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t|S_t)\, G_t.$$

- **High variance issue:** The returns G_t can exhibit significant variability, particularly in long or stochastic episodes.
- A **baseline**, often a learned value function $b(S_t)$ or a running average of past returns, is subtracted from G_t:

$$\nabla J(\theta) \approx \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t|S_t)\, \big[G_t - b(S_t)\big].$$

- **Effect:** Subtracting this baseline reduces the variance of the gradient estimate without biasing it, thereby *stabilizing training* and typically improving convergence speed.

2.4 Question 4 (5 Points)

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

Solution:

- **High Variance of Gradient Estimates:**
  - *Sampling-based updates:* REINFORCE relies on Monte Carlo returns from sampled episodes to estimate the policy gradient. Because each trajectory may differ substantially due to environment stochasticity or exploration, the estimated returns G_t can vary greatly from episode to episode.
  - *Long-horizon tasks:* If episodes are long (or γ is close to 1), small differences in actions early in an episode can accumulate into large differences in total returns, amplifying variance.
  - *Infrequent rewards or high noise:* If rewards are sparse or noisy, the sampled returns may not reliably reflect the true expected performance of a policy, leading to large swings in the gradient updates.

- **Sensitivity to Hyperparameters:**
  - Learning rate, discount factor, and baseline design critically affect convergence behavior.
- **Potential for Local Optima:**
  - Gradient-based methods can become stuck in local maxima or plateau regions if exploration is insufficient or if the reward signal is deceptive.

2.5 Question 5 (5 Points)

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

Solution:

- **Convergence Speed:**
  - Empirical results show that **adding a baseline** significantly accelerates learning, requiring fewer episodes to achieve comparable or higher returns.
- **Stability:**
  - Vanilla REINFORCE (no baseline) can converge but often exhibits larger performance fluctuations.
  - REINFORCE with baseline experiences *reduced variance* in gradients, yielding smoother and more stable training curves.
- **Reward Accumulation and Final Performance:**
  - Over multiple runs, the baseline-augmented algorithm typically reaches higher average returns, reflecting better policy refinement and fewer training instabilities.

2.6 Question 6 (5 Points)

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

Solution:

- **Variance from Sampling:**
  - Policy gradient methods like REINFORCE compute Monte Carlo estimates of the gradient, which inherently have high variance.
- **Negative Effects of High Variance:**
  - Can cause *unstable* or *slow* learning if updates oscillate.
  - Makes hyperparameter tuning (learning rate, batch size) more critical.
- **Mitigation Strategies:**
  - Use a **baseline**, such as a value function or average return, to reduce the magnitude of returns in the gradient calculation.
  - Employ **advantage functions** ($A_t = G_t - V(S_t)$).

- Ensure appropriate **exploration** mechanisms to avoid rapid oscillations in policy.

3 Task 3: REINFORCE in a continuous action space

3.1 Question 1 (5 Points)

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

Solution:

- **Observation Space:**
  - The state space is two-dimensional, representing:

$$[\text{position}, \quad \text{velocity}] .$$

  - Position typically ranges between $[-1.2, 0.6]$, and velocity ranges between $[-0.07, 0.07]$.
  - The agent observes these continuous values each timestep.
- **Action Space:**
  - The action space is *continuous*, typically 1-dimensional in `MountainCarContinuous`.
  - The agent chooses a force (torque) a_t in the range $[-1.0, +1.0]$ at each timestep.
  - This allows the agent to modulate the exact amount of force applied to the car, rather than selecting from discrete "push left/ push right/ no-op" actions.

3.2 Question 2 (5 Points)

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.

Solution:

- **Optimal Policy Behavior:**
  - An optimal strategy for `MountainCarContinuous` usually involves leveraging momentum: the agent often first moves slightly in the *opposite* direction to gain enough momentum before accelerating forward.
  - By carefully timing and scaling the applied force, the car can swing back and forth to climb up the rightmost hill with minimal total energy expenditure.
  - The agent applies only as much force as needed to crest the right hill, rather than constantly pushing at maximum force.
- **Scenario Example:**
  - (1) The car starts near the valley; the agent briefly applies a small negative force to roll left, increasing velocity in the negative direction.
  - (2) As the car nears the left boundary, the agent gradually switches to positive force to capitalize on the car's gained momentum.

  - (3) The car now carries extra speed going up the right hill, reducing the force required to reach the goal state (the flag at the top).
  - This cyclical approach of controlled back-and-forth motion is crucial for minimizing energy consumption.

3.3 Question 3 (10 Points)

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

(Hint: experience replay or target networks)

Solution:

- **Experience Replay:**
  - Even though pure REINFORCE traditionally computes gradients from on-policy trajectory samples, one approach to reduce forgetting is to store past transitions (s, a, r, s') in a replay buffer.
  - Periodically sampling from this buffer can stabilize learning by diversifying training examples over time, preventing the network from overfitting to the most recent experiences.
- **Target Networks (common in Actor-Critic methods):**
  - While vanilla REINFORCE does not typically use a target network, actor-critic variants (e.g., Deep Deterministic Policy Gradient, DDPG, or Soft Actor-Critic, SAC) often maintain a slowly updated target network for the critic.
  - This reduces the risk of large, destabilizing updates to the value estimates (or Q-function), which can help the policy avoid catastrophic forgetting by providing more consistent gradient signals.
- **Regularization and Ensemble Methods:**
  - Adding regularization (e.g., weight decay or dropout) or using ensemble networks can help the agent retain previously learned behaviors by smoothing parameter updates.
  - Ensembles can offer multiple estimates of state-action values, reducing the chance of drastically overwriting knowledge in a single network.

4 Task 4: Comparing Policy Gradient (REINFORCE) vs. Deep Q-Network (DQN)

4.1 Question 1 (15 Points)

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?

Solution:

In the Frozen Lake environment, Deep Q-Networks (DQN) generally outperform Policy Gradient methods like REINFORCE due to the long-horizon nature of the problem and the need for stable learning. DQN, being an off-policy algorithm, efficiently utilizes experience replay and a target network, which helps in stabilizing training and improving convergence speed. In contrast, REINFORCE, an on-policy method, suffers from high variance and slower convergence since it updates the policy based on complete episode returns, making it less effective in long-horizon problems where delayed rewards are common. Additionally, DQN learns a value-based representation that allows better generalization across state-action pairs, leading to a higher overall success rate. Therefore, in Frozen Lake, DQN achieves better results in terms of training stability, convergence speed, and overall performance compared to REINFORCE.

4.2 Question 2 (15 Points)

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?

Solution: The Frozen Lake environment introduces several challenges for reinforcement learning, primarily due to its stochastic transitions, sparse rewards, and long-horizon nature. The slippery surface makes state transitions unpredictable, adding randomness to agent actions, which complicates value estimation for DQN and policy updates for REINFORCE. Sparse rewards mean that the agent receives feedback only upon reaching the goal, making it difficult to assign credit to previous actions, leading to slow learning and high variance in policy gradient updates. Additionally, the long-horizon problem requires the agent to plan effectively over many steps, which is particularly challenging for on-policy methods like REINFORCE, as they rely on complete episode returns for updates. DQN, while more stable due to experience replay and target networks, still struggles with exploration, as it can get stuck in suboptimal policies if it fails to balance exploitation and exploration effectively. These challenges collectively slow down convergence and make optimal policy learning difficult for both algorithms.

4.3 Question 3 (20 Points)

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.

Solution:

While DQN benefits from experience replay to improve sample efficiency, it is still fundamentally a value-based method, which can struggle with convergence stability in environments where unlimited sampling is available. In contrast, policy gradient methods, despite being less sample-efficient in constrained settings, excel when interactions are abundant because they directly optimize the policy and do not suffer from issues like overestimation bias or function approximation instability that can arise in DQN. Additionally, policy gradient methods can learn stochastic policies, which are beneficial in environments requiring exploration. However, if the environment is purely discrete and deterministic, DQN may still perform well due to its stable value learning. Ultimately, when sampling is cheap and unlimited, policy gradient methods like REINFORCE approaches often become more advantageous because they can fully exploit the high interaction capacity to refine the policy without being limited by off-policy learning constraints.

5 References

[1] Cover image designed by freepik

[2] Policy Search

[3] CartPole environment from OpenAI Gym

[4] Mountain Car Continuous environment from OpenAI Gym

[5] FrozenLake environment from OpenAI Gym