International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Reinforcement Learning: A Foundational Analysis of Algorithm in Sequential Decision-Making

Dhruvi Prakash Karu, Rutu Pradipkumar Pansaniya, Om Pandya, Janki Tejas Patel

Computer Engineering Department, SAL College of Engineering
Computer Engineering Department, SAL College of Engineering
Computer Engineering Department, SAL College of Engineering
Assistant Professor, SAL College of Engineering

ABSTRACT:

Reinforcement Learning (RL) is the computational paradigm dedicated to solving sequential decision-making problems, where an autonomous agent learns an optimal policy to maximize cumulative reward through interaction with a dynamic environment. This paper provides a foundational analysis of RL, beginning with the Markov Decision Process (MDP) framework and reviewing core model-free algorithms, including the Value-Based (DQN) and Policy Gradient (PPO) families. We then systematically dissect critical limitations in scaling RL, focusing on hyperparameter sensitivity, the inherent sample inefficiency of on-policy methods, and the challenge of sparse rewards that render the credit assignment problem intractable.

Key words: reinforcement learning; deep reinforcement learning; Markov Decision Process; Policy Gradient (PPO) ; inverse reinforcement learning; multi-agent RL

1] Introduction

Reinforcement Learning (RL) is the computational paradigm dedicated to solving sequential decision-making problems, where an autonomous agent learns an optimal sequence of actions by interacting directly with a dynamic, often initially unknown, environment. The core complexity of RL lies in maximizing the **cumulative reward** received over time, necessitating a sophisticated mathematical framework capable of handling uncertainty across an extended temporal horizon. This learning process generates a trajectory—a sequence of states, actions, and rewards—where the agent's objective is inherently long-term reward maximization.

The RL agent examines the condition of the environment and chooses an appropriate action. If the RL agent performs the correct action, it receives a positive reward. If the agent makes the wrong move, a negative is generated. RL must balance exploration and exploitation. Exploitation occurs when an agent attempts to maximize the reward based on a previously established route. If the agent always tries to explore a new way to reach the destination, it is called exploration. RL The model does not need a large dataset and learns by trial and error.

The distinction between RL and traditional supervised learning (SL) is fundamental. SL relies entirely on pre-labeled datasets for pattern recognition, whereas RL operates in dynamic environments and receives evaluative feedback (rewards) rather than instructive feedback (labels). The agent must determine which past action, potentially many time steps removed, is responsible for a current high or low reward. This complexity prevents the direct application of supervised optimization techniques, requiring the agent to build a sophisticated understanding of the environment's temporal causality. RL inherently requires an algorithmic structure that can manage the non-stationary, interactive learning process, shifting the challenge from static classification to dynamic control.

Beyond the agent and the environment, there are four main sub-elements of a reinforcement learning system: a policy, a reward signal, a value function, and, optionally, a model of the environment.

A **policy** defines the learning agent's way of behaving at a given time. Simply putting, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. A **reward signal** defines the goal in a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number, a reward. The agent's sole objective is to maximize the total reward it receives over the long run. The reward signal thus defines what are the good and bad events for the agent. The reward signal is the primary basis for altering the policy. A **value function** indicates the **total expected reward** an agent can accumulate in the future, starting from that particular state. **Model of the environment** is something that mimics the behaviour of the environment, or more generally, that allows inferences to be made about how the environment will behave.

2] Markov Decision Processes (MDPs)

The Markov Decision Process provides the mathematical bedrock for Reinforcement Learning. An MDP is formally defined by the quintuple $M=\langle S, A, T, R, \gamma\rangle$, where S is the state space, A is the action space, T (s, a, s′) is the transition model, R is the immediate reward function, and γ is the discount factor. The fundamental simplifying assumption is the

Markov Property: the future state and reward depend solely on the current state and the action taken, independent of the entire history of preceding states and actions.

The objective is to find the optimal policy $\pi*$ that maximizes the **Expected Discounted Sum of Rewards** over a potentially infinite horizon. To assess and optimize policies, RL uses the recursive

Bellman equations to define value functions, Vπ (s) and Qπ (s, a). The theoretical robustness of tabular RL stems directly from the convergence proofs associated with these equations, which guarantee that the iterative process converges to the optimal value function, provided the rewards are bounded.

$P(s', r | s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}$

3] Model-Free Methods: Value-Based Algorithms

3.1] Q-learning

Q-learning is one of the most widely adopted reinforcement learning (RL) algorithms. It follows a **model-free** and **off-policy** approach, enabling an agent to learn optimal actions through direct interaction with the environment rather than relying on an explicit model of it. The primary objective of Q-learning is to determine the **maximum Q-value**, which represents the expected future reward of taking a particular action in a given state.

Q-learning is a **value-based** method where the agent incrementally learns the **quality (Q-value)** of state–action pairs by trial and error. Initially, all Q-values are assigned random or zero values, and they are iteratively refined as the agent gains experience. The algorithm maintains a **Q-table** that stores Q-values for each state–action pair, serving as the agent's knowledge base for decision-making.

Key Components of Q-Learning

1. **Q-Values (Action-Values)**

   Q-values quantify the expected cumulative reward associated with performing an action AAA in a state SSS. These values are continuously updated using the **Temporal Difference (TD) learning rule**, allowing the agent to balance new and prior information.

2. **Rewards and Episodes**

   The agent transitions between states by selecting actions and receiving feedback in the form of rewards. Each sequence of interactions continues until a **terminal state** is reached, marking the end of an episode.

3. **Temporal Difference (TD) Update Rule**

   Q-learning employs the following update equation to refine its Q-values:

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_{A'} Q(S', A') - Q(S, A)]$

where:

- S: current state
- A: action taken
- S′: next state
- A′: best possible action in the next state
- R: reward received for action AAA in state SSS
- α: learning rate, controlling the influence of new information
- γ: discount factor, balancing immediate and future rewards

This TD-update enables the agent to learn directly from raw experience without requiring complete knowledge of the environment's dynamics.

4. ϵ-Greedy Policy (Exploration vs. Exploitation)

   The ϵ-greedy policy governs the agent's action-selection strategy by balancing exploration and exploitation:

   - Exploitation $(1-\epsilon)$: the agent selects the action with the highest Q-value, leveraging existing knowledge to maximize reward.

- Exploration (ϵ): the agent randomly selects an action to discover potentially better strategies. This balance ensures the agent avoids suboptimal convergence and continues to improve over time.

Operational Steps of Q-Learning

1. Initialization:
   The agent begins in an initial state SSS, representing the environment's current condition.
2. Action Selection: Based on the current Q-values and the chosen policy (typically ϵ-greedy), the agent selects an action A. It may explore new actions or exploit known high-value actions.
3. Environment Response: Upon executing the action, the environment provides a new state (S) and a reward (R), reflecting the immediate consequence of the action.
4. Q-Table Update: The agent updates the Q-value corresponding to the state–action pair (S, A) using the TD-update rule. This update incorporates both the immediate reward and the estimated future rewards of subsequent states.
5. Policy Refinement and Iteration: With the updated Q-values, the agent refines its policy and continues the learning process across multiple episodes. Over time, as the Q-table converges, the agent's policy approaches the optimal policy, yielding the maximum expected cumulative reward.

3.2] Deep Q-Networks (DQN)

For large state spaces, DQN uses neural networks as function approximators. To stabilize the training, DQN employs two critical architectural solutions: Experience Replay (randomly samples transition to break data correlation) and Target Networks (a separate, delayed network providing a stable objective for the Bellman update). These mechanisms enable DQN to achieve high sample efficiency, making it highly effective for tasks with discrete action spaces.

3.3] C51 (Categorical DQN)

C51 (Categorical DQN) is a well-known Q-learning algorithm for discrete action spaces. In classic Q-learning, the Q value is expressed as a single value for each state-action pair, which tackles the problem of classical Q-learning discretization. Instead of employing a single Q value for each action, C51 extends the DQN by including a categorical distribution across the Q values. QR-DQN (quantile regression DQN) is an enhancement to the DQN algorithm. QR-DQN updates the network's parameters using a quantile regression loss function, which differs from the standard mean squared error loss function used in DQN. It is defined as the sum of the Huber losses for each quantile of the action-value distribution. This loss function enables the agent to learn various possible action values rather than just one optimum action.

3.4] Hindsight experience replay (HER)

Hindsight experience replay (HER) is another type of RL algorithm. It enables the agent to learn from previous failures and overcome them by focusing on the present objective. Additionally, the same experience is replayed with the new aim in mind. It is beneficial for tasks with sparse rewards and is appropriate for various suboptimal goal states. HER uses experience replay, which stores all previous experiences and is compatible with algorithms such as DDPG, DQN, and A3C

3.5] POLICY GRADIENT (PPO):

Policy Gradient (PG) algorithms are a fundamental class of Reinforcement Learning methods that operate by directly optimizing the agent's behaviour strategy, or **policy** (π). Unlike Value-Based methods (like Q-Learning), which learn the value of actions and then derive a policy from those values, PG methods search directly in the policy space via gradient ascent to maximize the expected cumulative reward.

The theoretical foundation for this approach is the Policy Gradient Theorem, which guarantees that the expected sample gradient derived from observed experience is equal to the true gradient of the performance objective.

I. The REINFORCE Algorithm: Monte Carlo Policy Gradient

The prototypical Policy Gradient algorithm is **REINFORCE** (also known as Monte Carlo Policy Gradient). It is characterized as a Monte Carlo method because it relies on collecting a complete episode (a full sequence of interactions from start to termination) before performing any parameter updates.

The Process of REINFORCE

The algorithm operates iteratively, repeating the following steps until the policy parameters converge:

1. **Episode Generation:** The agent, following its current policy πθ, interacts with the environment to generate a full trajectory, which is a sequence of states, actions, and rewards: S_0, A_0, R_1, S_1, A_1, R_2, S_{T-1}, A_{T-1}, R_T.

2. **Return Calculation (Gt):** For every time step t in the episode, the return Gt is calculated. Gt is the total discounted reward received from that point onward until the end of the episode.
3. **Policy Parameter Update:** For each step t, the policy parameters θ are updated using the calculated return Gt and the log probability of the action taken:

$\theta \leftarrow \theta + \alpha \cdot \gamma^t \cdot G_t \cdot \nabla\theta \ln\pi\ (A_t|S_t\ , \theta)$

Where:

- θ: The policy parameters (e.g., the weights of a neural network representing the policy).
- α: The learning rate.
- γ^t: The discounted factor, weighting the reward over time.
- G_t: The measured return from time step t, which serves as the weight for the update.
- $\nabla\theta \ln\pi\ (A_t|S_t, \theta)$: The gradient of the log probability of the action taken. This component mathematically identifies how the action probability should be adjusted with respect to the parameters θ.

If Gt is positive (a high reward was ultimately received), the gradient step increases the probability of taking the action A_t in state S_t. If G_t is negative, it decreases that probability.

II. Challenges and Variance Reduction

The core weakness of the standard REINFORCE algorithm is its **high variance**. Since it is a Monte Carlo method, the gradient estimate for a given action

A_t is entirely dependent on the highly stochastic and noisy total return G_t accumulated across the entire trajectory following that action. This high variance can significantly slow down or destabilize the learning process.

4] Case study:

4.1] Problem Statement

Scenario - Robots in a Warehouse

A growing e-commerce company is building a new warehouse, and the company would like all of the picking operations in the new warehouse to be performed by warehouse robots. In the context of e-commerce warehousing, "picking" is the task of gathering individual items from various locations in the warehouse in order to fulfil customer orders.

After picking items from the shelves, the robots must bring the items to a specific location within the warehouse where the items can be packaged for shipping. In order to ensure maximum efficiency and productivity, the robots will need to learn the shortest path between the item packaging area and all other locations within the warehouse where the robots are allowed to travel. Use Q-learning to accomplish this task!

4.2] The Environment

4.2.1] States

The e-commerce company's warehouse can be represented as a diagram:

Each black square is an item storage location (e.g., a shelf)

Each white square is part of an aisle where the robots can travel

The green square is the item packaging area

Each location in the warehouse is a state

Identified by a row and column index

Terminal states are the black and green squares!

4.2.2] Actions

The actions that are available to the Al agent are to move the robot in one of four directions:

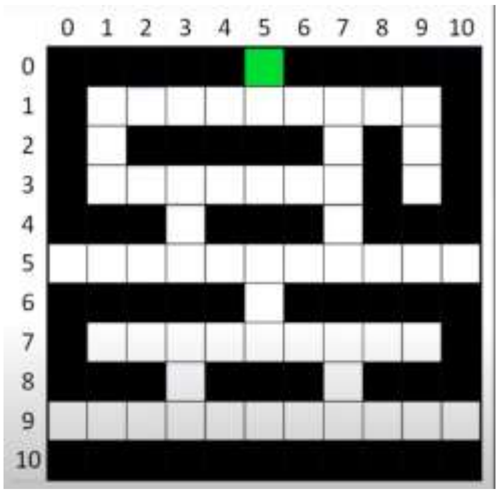
- Up
- Right
- Down
- Left

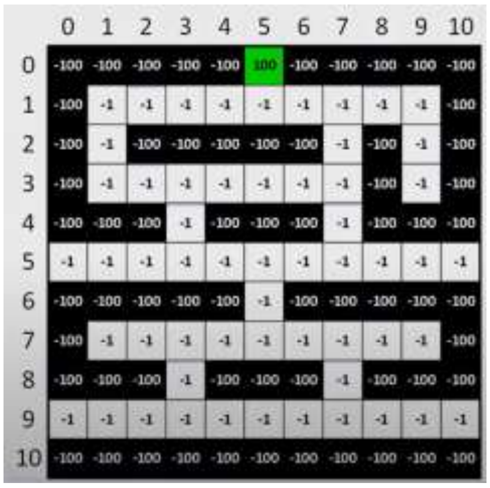
Obviously, the Al agent must learn to avoid driving into the item storage areas!

4.2.3] Rewards

To help the Al agent learn, each state (location) in the warehouse is assigned a reward value. The agent may begin at any white square, but its goal is always the same: to maximize its total rewards

Negative rewards (i.e., punishments) are used for all states except the goal. This encourages the Al to identify the shortest path to the goal by minimizing its punishments.





	0	1	2	3	4	5	6	7	8	9	10
0	-100	-100	-100	-100	-100	100	-100	-100	-100	-100	-100
1	-100	-1	-1	-1	-1	-1	-1	-1	-1	-1	-100
2	-100	-1	-100	-100	-100	-100	-100	-1	-100	-1	-100
3	-100	-1	-1	-1	-1	-1	-1	-1	-100	-1	-100
4	-100	-100	-100	-1	-100	-100	-100	-1	-100	-100	-100
5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
6	-100	-100	-100	-100	-100	-1	-100	-100	-100	-100	-100
7	-100	-1	-1	-1	-1	-1	-1	-1	-1	-1	-100
8	-100	-100	-100	-1	-100	-100	-100	-1	-100	-100	-100
9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
10	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100

```
[-100. -100. -100. -100. -100.  100. -100. -100. -100. -100.-100.]
[-100.   -1.     -1.    -1.   -1.     -1.   -1.    -1.    -1.     -1. -100.]
[-100.   -1.   -100. -100. -100. -100. -100.    -1. -100.     -1. -100.]
[-100.   -1.    -1.      -1.  -1.      -1.    -1.    -1.  -100.     -1. -100.]
[-100. -100. -100.    -1. -100. -100. -100.    -1. -100.  -100. -100.]
[-1.     -1.     -1.      -1.  -1.      -1.    -1.    -1.    -1.      -1.    -1.]
[-100. -100. -100. -100. -100.   -1. -100. -100. -100. -100.  -100.]
[-100.  -1.     -1.      -1.  -1.      -1.    -1.    -1.    -1.      -1.  -100.]
[-100. -100. -100.    -1. -100. -100. -100.   -1.  -100.  -100. -100.]
[-1.    -1.     -1.      -1. -1.       -1.    -1.    -1.    -1.      -1.     -1.]
[-100. -100. -100. -100. -100. -100. -100. -100. -100. -100. -100.]
```

The Al agent's goal is to learn the shortest path between the item packaging area and all of the other locations in the warehouse where the robot is allowed to travel.

As shown in the image above, there are 121 possible states (locations) within the warehouse. These states are arranged in a grid containing 11 rows and 11 columns. Each location can hence be identified by its row and column index.

4.3] Train the Model

Our next task is for our Al agent to learn about its environment by implementing a Q-learning model. The learning process will follow these steps:

1. Choose a random, non-terminal state (white square) for the agent to begin this new episode.

2. Choose an action (move up, right, down, or left) for the current state. Actions will be chosen using an epsilon greedy algorithm. This algorithm will usually choose the most promising action for the Al agent, but it will occasionally choose a less promising option in order to encourage the agent to explore the environment.

3. Perform the chosen action, and transition to the next state (i.e., move to the next location).

4. Receive the reward for moving to the new state, and calculate the temporal difference.

5. Update the Q-value for the previous state and action pair.

6. If the new (current) state is a terminal state, go to #1. Else, go to #2.

This entire process will be repeated across 1000 episodes. This will provide the Al agent sufficient opportunity to learn the shortest paths between the item packaging area and all other locations in the warehouse where the robot is allowed to travel, while simultaneously avoiding crashing into any of the item storage locations!

4.4] Get Shortest Paths

Now that the Al agent has been fully trained, we can see what it has learned by displaying the shortest path between any location in the warehouse where the robot is allowed to travel and the item packaging area. For example, shortest path (3,9) = [[3, 9], [2, 9], [1, 9], [1, 8], [1, 7], [1, 6], [1, 5], [0, 5]]

Shortest path (5,0) = [[5, 0], [5, 1], [5, 2], [5, 3], [4, 3], [3, 3], [3, 2], [3, 1], [2, 1], [1, 1], [1, 2], [1, 3], [1, 4], [1, 5], [0, 5]]

5] Conclusion

This study analysed key Reinforcement Learning algorithms, focusing on Q-Learning and Proximal Policy Optimization (PPO) within a grid-world environment. Results show that Q-Learning converges faster in discrete tasks, while PPO offers greater stability and adaptability for complex or continuous domains. Both methods effectively learn optimal policies through different mathematical approaches—value-based and policy-based learning. Overall, RL remains a powerful framework for autonomous decision-making, with future research aimed at improving scalability, handling sparse rewards, and extending applications to multi-agent and real-world control systems.

6] REFERENCES:

1. Dhakal, B. MDP (Markov Decision Process) — RL (Reinforcement Learning).
2. Medium. Available at: https://medium.com/@binaydhakal35/mdp-markov-decision-process-rl-reinforcement-learning-bc85e5d25031
3. Machines, 2024. Available at: https://www.mdpi.com/2504-4990/4/1/9
4. MIT. Markov Decision Processes: Definition and Value Functions.
5. Introduction to Machine Learning, MIT. Available at: https://introml.mit.edu/notes/mdp.html
6. Wikipedia. Markov Decision Process.
7. Wikipedia. Available at: https://en.wikipedia.org/wiki/Markov_decision_process
8. Lambert, J. W. Value Iteration.
9. Reinforcement Learning Notes. Available at: https://gibberblot.github.io/rl-notes/single-agent/value-iteration.html
10. Statistics Stack Exchange. Available at: https://stats.stackexchange.com/questions/243384/deriving-bellmans-equation-in-reinforcement-learning
11. Stack Overflow. What is the difference between Q-Learning and SARSA?
12. Stack Overflow. Available at: https://stackoverflow.com/questions/6848828/what-is-the-difference-between-q-learning-and-sarsa
13. Su, X., Zhang, Y., & Liu, Q. Comparative Analysis of Q-Learning and SARSA Algorithms.
14. Atlantis Press, 2023. Available at: https://www.atlantis-press.com/article/125998063.pdf
15. Rahman, M. A., et al. Comparative Study of Reinforcement Learning Performance Based on PPO and DQN Algorithms.
16. ResearchGate, 2023. Available at: https://www.researchgate.net/publication/393613412_Comparative_Study_of_Reinforcement_Learning_Performance_Based_on_PPO_and_DQN_Algorithms
17. BytePlus. Available at: https://www.byteplus.com/en/topic/514186
18. Lambert, J. W. The Policy Gradient Theorem and REINFORCE.

19. Reinforcement Learning Notes. Available at: https://johnwlambert.github.io/policy-gradients/
20. Weng, L. Policy Gradient Algorithms.
21. Lilian Weng's Blog, 2018. Available at: https://lilianweng.github.io/posts/2018-04-08-policy-gradient/
22. Greensmith, J., Bartlett, P. L., & Baxter, J. Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning.
23. Journal of Machine Learning Research, 2004. Available at: https://www.jmlr.org/papers/volume5/greensmith04a/greensmith04a.pdf
24. Reddit. How do Actor-Critic Networks reduce the variance?
25. r/reinforcementlearning. Available at: https://www.reddit.com/r/reinforcementlearning/comments/ud1lck/how_do_actorcritic_networks_reduce_the_variance/