CTRL

Reinforcement Learning

Project Report

Eklavya Mentorship Program

**Authors**: *Aditya Vivekanand, Ariv Fernandes*

**Mentors**: *Viraj Shah, Aryan Karawale*

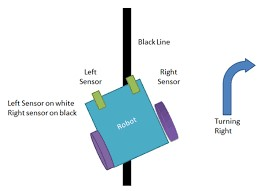
Index

|  |  |  |
| --- | --- | --- |
| **Sr no.** | **Content** | **Page no.** |
| 1. | Overview | 3 |
| 2. | Introduction   * The RL problem * Approach to find a solution * Agent-Environment interaction * Implementing environments and their solutions | 3 |
| 3. | Deeper Conceptual Understanding   * Multi- Armed bandits * Dynamic Programming * Temporal-Difference Learning | 6 |
| 4. | Implementation   * K-armed bandit problem * Simulation of a line following bot using Q- Learning | 13 |
| 5. | Applications | 14 |
| 6. | Conclusion and Future Work |  |

Overview:

This project intends to be an introduction to reinforcement learning based control system. It involves the study and implementation of classical reinforcement learning algorithm, at least one of which will be a simulation to solve control problem.

Here, we aim to have an online simulation of a line follower bot using reinforcement learning.



Introduction:

# The RL Problem:

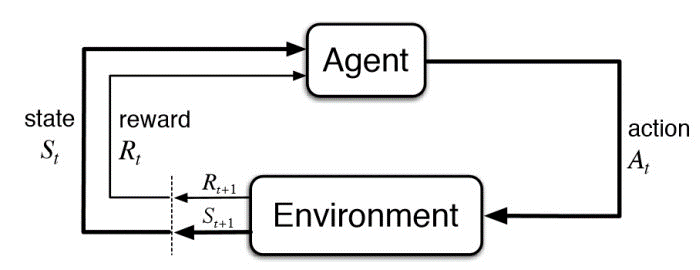
Reinforcement Learning (RL) stands at the intersection of artificial intelligence and decision-making, mimicking the way humans learn through trial and error. This paradigm is particularly potent when addressing problems where explicit programming or supervised learning falls short, making it a key player in fields ranging from robotics to game-playing agents.

# Approach to find a Solution:

At its core, the RL problem revolves around an agent interacting with an environment. The agent, aiming to maximize a cumulative reward, navigates through a sequence of states by taking actions. However, these actions are not explicitly provided; the agent must learn from experience, receiving feedback in the form of rewards or penalties. This dynamic environment introduces a challenge: finding the optimal strategy to maximize the long-term reward.

# Agent-Environment Interaction:

The interaction between the agent and the environment can be treated as a loop, with each iteration corresponding to an exchange of information between them, resulting in a state transition where this can repeat.

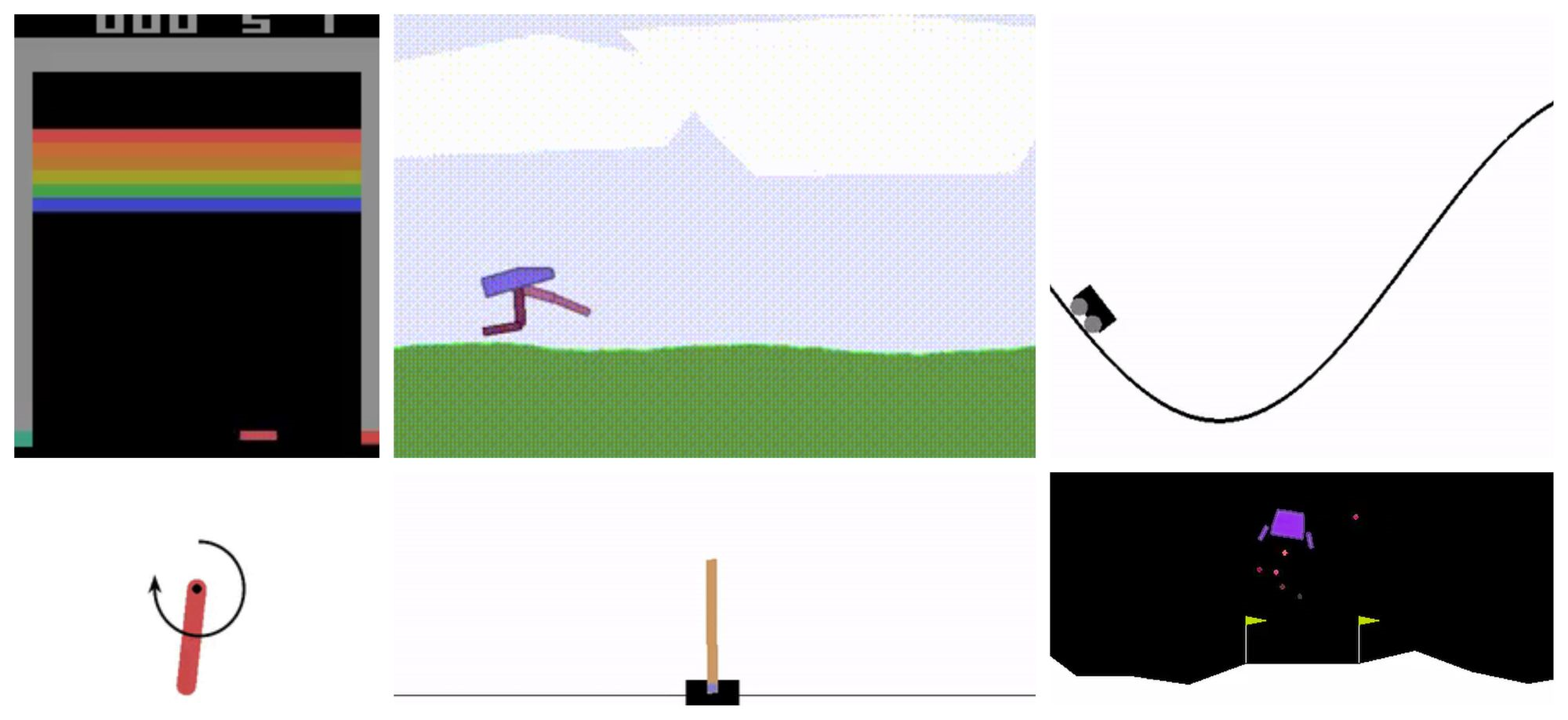


Apart from the agent being trained and the environment it is being trained with, RL learning systems are composed of four major elements:

1. Policy followed by the agent.
2. A reward function determined by the specifics of the problem.
3. A value function learned by the agent.
4. A model of the environment, also learned by the agent.

# Implementing Environments and their Solutions:

Creating a simulated environment is crucial for RL experiments. This can range from game environments for testing algorithms to virtual models mirroring real-world scenarios. However, the challenge lies not only in constructing these environments but also in defining states, actions, and rewards effectively. Implementing solutions involves striking a balance—creating environments that are complex enough to be meaningful but not so intricate that learning becomes intractable. The design choices here significantly impact the learning process and the generalization of the agent's acquired knowledge.



Deeper Conceptual Understanding:

# Multi- Armed Bandits:

In the k-armed bandit problem, you repeatedly choose from k options, each with an unknown reward distribution. Your goal is to maximize the total reward over time. The challenge lies in balancing exploration (trying new options) and exploitation (choosing the known best option). The value of an action is its expected reward, denoted as q\*(a). However, you don't know these values for certain and maintain estimates, denoted as Qt(a). Greedy actions have the highest estimated values, and exploiting them maximizes immediate reward, while exploring non-greedy actions improves estimates for long-term gain. The conflict between exploration and exploitation is a key challenge, and while sophisticated methods exist, this book focuses on simple balancing methods for clarity in the k-armed bandit problem.

For using the estimates to make action selection decisions, which we collectively call action-value methods. A natural way to estimate this is by averaging the rewards actually received:



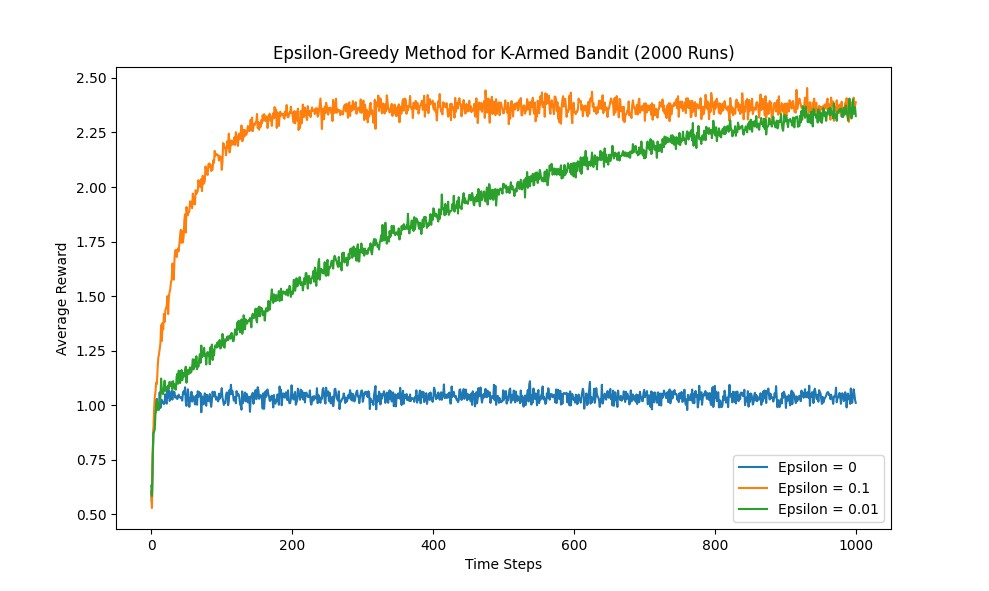
We call this the sample-average method for estimating action values because each estimate is an average of the sample of relevant rewards. Of course this is just one way to estimate action values, and not necessarily the best one.

We write this greedy action selection method as:



Greedy action selection always exploits current knowledge to maximize immediate reward. A simple alternative is to behave greedily most of the time, but every once in a while, say with small probability epsilon(e), instead select randomly from among all the actions with equal probability, independently of the action-value estimates. We call methods using this near-greedy action selection rule epsilon -greedy methods. An advantage of these methods is that, in the limit as the number of steps increases, every action will be sampled an infinite number of times, thus ensuring that all the Qt(a) converge to their respective q\*(a). This of course implies that the probability of selecting the optimal action converges to greater than (1-e), to near certainty.

When a learning method applied to that problem selected action At at time step t, the actual reward, Rt, was selected from a normal distribution with mean q\*(At) and variance 1. For any learning method, we can measure its performance and behavior as it improves with experience over 1000 time steps when applied to one of the bandit problems. This makes up one run. Repeating this for 2000 independent runs, each with a different bandit problem, we obtained measures of the learning algorithm’s average behavior.



The figure compares a greedy method (e = 0) with two epsilon greedy methods (e = 0.01 and e = 0.1).

# Finite Markov Decision Processes:

## 1. Agent-Environment Interface:

The agent interacts with the environment through actions, receiving states as input and producing actions as output.

## 3. Returns and Episodes:

Returns represent the cumulative sum of rewards. In episodic tasks, episodes have a finite duration, while in continuous tasks, interactions continue indefinitely.



## 5. Policy and Value Functions:

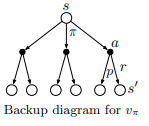
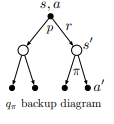
A policy defines the agent's strategy, mapping states to actions. Value functions, like state-value and action-value functions, assess the desirability of states or state-action pairs.

State- value function for a given policy is:



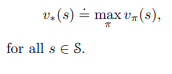
Whereas the action- value function for a given policy is:

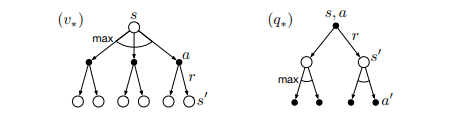


## 6. Optimal Policy and Optimal Value Function:

The optimal policy maximizes expected returns, while the optimal value function represents the maximum expected return achievable in each state under the optimal policy.

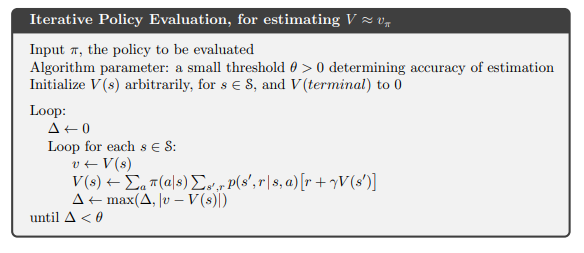


# Dynamic Programming:

Dynamic programming is a problem-solving technique in computer science and mathematics that involves breaking down complex problems into simpler, overlapping subproblems. It optimally solves each subproblem only once and stores the solutions, avoiding redundant computations. For understanding dynamic programming, we need to look over 3 concepts.

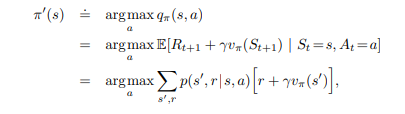
1. Policy Evaluation:

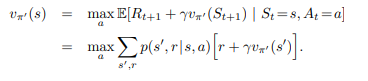
Iteratively calculates the value function under a given policy by updating state values based on the expected returns.



1. Policy Improvement:

After evaluating a policy in dynamic programming, policy improvement follows by making the policy more greedy based on the current value function. Actions are adjusted to maximize expected returns, iterating until convergence to refine and approach the optimal policy for maximizing cumulative rewards.

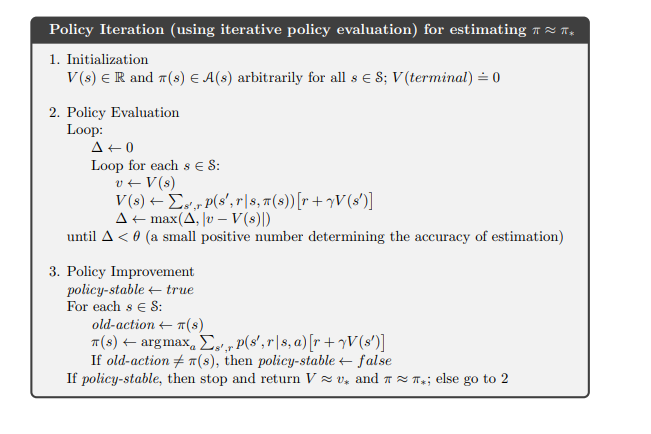




1. Policy Iteration:

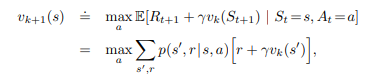
Alternates between policy evaluation and improvement to converge towards the optimal policy, ensuring both are iteratively refined.

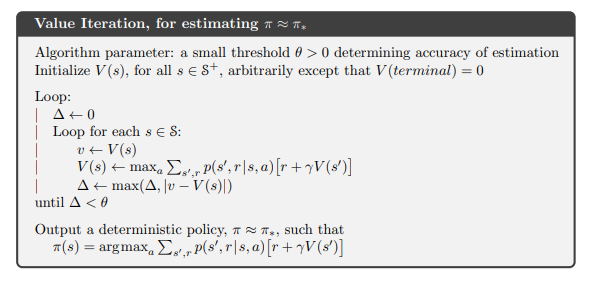




1. Value Iteration:

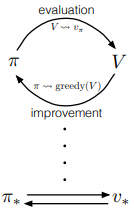
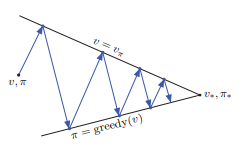
Simultaneously updates state values and improves the policy in a single iteration, efficiently converging to the optimal policy through dynamic programming.





1. Generalised Policy Iteration:

We use the term generalized policy iteration (GPI) to refer to the general idea of letting policy-evaluation and policy-improvement processes interact, independent of the granularity and other details of the two processes.

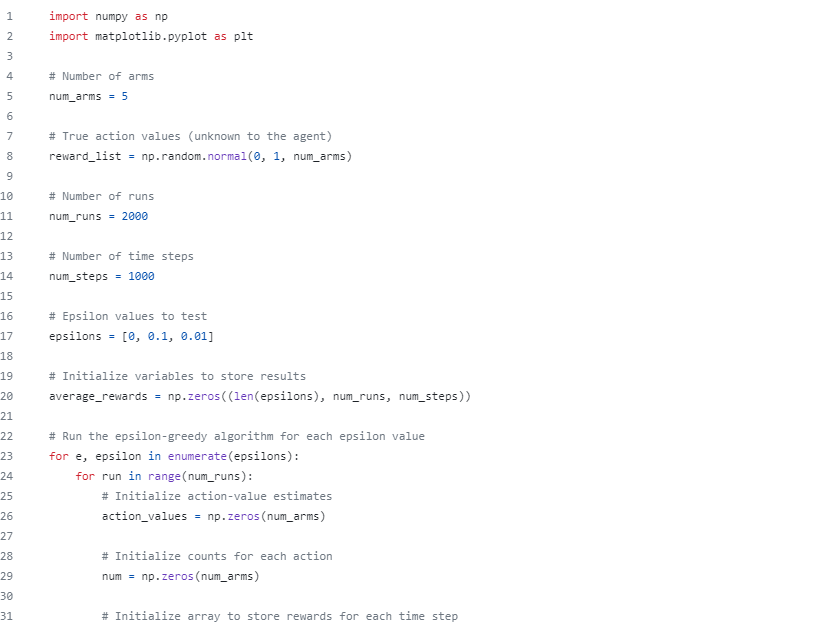
# Temporal-Difference Learning:

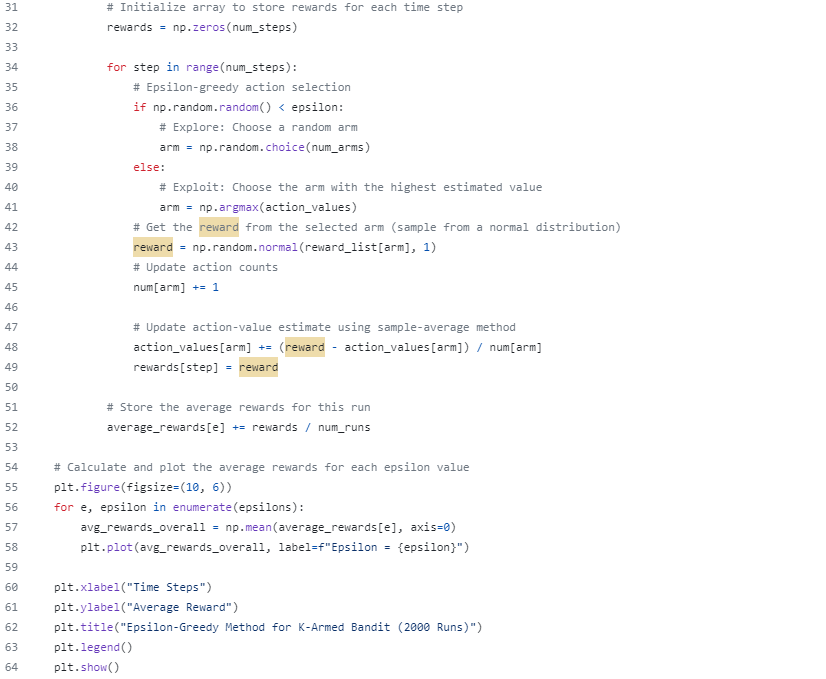
TD learning updates value estimates by bootstrapping from the current estimate and the immediately observed reward, bridging the gap between Monte Carlo and dynamic programming methods. It's often used for online learning in real-time scenarios.

Q-learning is a model-free off-policy algorithm (comes under TD learning) that learns a Q-function, representing the expected cumulative reward for taking an action in a given state. It excels in environments with discrete state and action spaces and is known for its stability and convergence guarantees.

Implementation:

# k- armed bandits:





# Simulation of a line following bot implementing Q- Learning Algorithm:

Applications:

# Robotics:

RL is extensively used in robotics to enable machines to learn and optimize their movements. Robots can learn complex tasks, such as grasping objects or navigating through unknown environments, by interacting with their surroundings and receiving feedback on their actions.

# Game Playing:

In the realm of gaming, RL has achieved remarkable success. Game-playing agents, like AlphaGo, have demonstrated the ability to surpass human performance by learning optimal strategies through interactions with the game environment. This extends beyond board games to video games and simulations.

# Autonomous Vehicles:

RL plays a vital role in training autonomous vehicles. Agents learn to make decisions in dynamic and unpredictable traffic scenarios, adapting to various conditions and optimizing routes. This application contributes to the development of safer and more efficient transportation systems.

Conclusion and Future Work:

1. Exploration and implementation of more problems covering the concepts of dynamic programming, Monte Carlo technique and Temporal-Difference learning.
2. We aim to implement the classic reinforcement learning algorithms, in which we’ll run it on an RPi to solve a control problem.