PROJECT REPORT

CSE 546 - Reinforcement Learning

FALL 2024

Submitted By:

Krishnanand, k33, 50560535

Anip Kumar, anipkuma, 50518750

Saumya Pandey, spandey5, 50372672

Introduction

In today's world, we can see how growing data and connected devices have been getting difficult to manage through traditional network routing methods. These traditional methods struggle with traffic congestion, latency, and fluctuating traffic patterns. Through this project, we are trying to leverage Reinforcement Learning (RL) techniques in order to achieve optimal real-time network routing, hence promoting efficacy in communication in dynamic environments.

Traffic Congestion

Traffic congestion in networks occurs when the volume of data exceeds the capacity of the network to process it efficiently. This often happens during peak usage times when multiple devices and users attempt to transmit data simultaneously over a shared medium. Similar to a traffic jam on a road, congestion leads to longer delays, increased queuing of packets, and sometimes dropped packets when buffers overflow. This impacts the quality of service and may require retransmission of data, further aggravating the problem. Effective congestion control mechanisms, such as traffic shaping or load balancing, are essential to mitigate these issues and ensure smoother data flow.

Latency

Latency is the delay in transmitting data from its source to its destination across a network. It is a critical metric in network performance, measured as the time taken for a packet to complete a round trip, including acknowledgment. Latency is influenced by several factors, including the physical distance between devices, the speed of the transmission medium, and processing delays at routers and switches. High latency negatively affects applications requiring real-time interactions, such as online gaming, video conferencing, or VoIP. Reducing latency often involves optimizing network paths, upgrading infrastructure, or using techniques like caching to bring data closer to the end user.

Throughput

Throughput refers to the amount of data successfully transmitted through a network over a specific period, typically measured in bits per second (bps). It represents the network's actual performance as opposed to its theoretical maximum capacity. Throughput is often impacted by latency, congestion, and protocol overheads, which reduce the effective bandwidth available to users. Low throughput can cause issues for data-intensive applications such as video streaming, cloud computing, or large file transfers. To improve throughput, network administrators may implement solutions like increasing bandwidth, enhancing routing efficiency, or using compression techniques to maximize data transfer rates.

Objectives

1. Develop a Reinforcement Learning framework for dynamic traffic routing.
2. Minimize network congestion, latency using intelligent routing strategies.
3. Explore Multi-Agent RL for decentralized decision-making (time permitting).
4. Compare RL-based methods against traditional algorithms using metrics like latency and throughput.
5. Real-Life Inspiration and Implementation in Network Traffic This project draws inspiration from real-world challenges in network traffic management, where the primary objective is to optimize the flow of data packets through a network while minimizing delays, avoiding congestion, and maximizing throughput. These challenges are prevalent in diverse domains such as telecommunications, IoT, cloud computing, and large-scale distributed systems.

Methodology

Key Techniques

Temporal Difference Learning

Q-Learning, SARSA, Actor-Critic (A2C), and Double DQN

Here's an explanation of each **key technique** in reinforcement learning, covering **Temporal Difference Learning**, **Q-Learning**, **SARSA**, **Actor-Critic (A2C)**, and **Double DQN**:

1. Temporal Difference (TD) Learning

Definition: TD Learning is a method in reinforcement learning (RL) that combines ideas from Monte Carlo methods and dynamic programming. It learns directly from raw experiences without needing a model of the environment.

Key Idea:

TD methods update the value of a state based on a combination of the current estimate and the observed reward plus the estimated value of the next state.

This is achieved using the **TD error**. Here, rr is the immediate reward, γ\gamma is the discount factor, V(s′)V(s') is the value of the next state, and V(s)V(s) is the value of the current state.

Applications: TD methods are foundational for other RL techniques like Q-Learning and SARSA.

2. Q-Learning

Definition: Q-Learning is an off-policy TD control algorithm that learns the optimal action-value function, Q(s,a)Q(s, a), which gives the expected reward for taking action aa in state ss.

Key Idea:

It is **off-policy** because it evaluates the optimal policy (greedy action selection) while following a potentially different behavior policy (like ϵ\epsilon-greedy exploration).

Applications: Widely used in grid-world problems, games, and robotics.

3. SARSA (State-Action-Reward-State-Action)

Definition: SARSA is an on-policy TD control algorithm that learns the action-value function Q(s,a)Q(s, a) but updates it based on the actual action taken, rather than the optimal future action.

Key Idea:

The update rule is:

$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$ Where a′a' is the action actually taken in the next state s′s'.

It is **on-policy** because it evaluates and improves the policy being followed.

Comparison with Q-Learning:

SARSA incorporates the exploratory actions into its updates, making it safer in stochastic environments.

Q-Learning focuses solely on the optimal policy, potentially leading to instability in noisy environments.

Applications: Suitable for environments with unpredictable outcomes, like real-world robotics.

4. Actor-Critic (A2C)

Definition: Actor-Critic is a policy gradient method that combines elements of value-based and policy-based methods. It uses two components:

Actor: Chooses actions based on a policy $\pi(a|s,\theta_{actor})$.

Critic: Evaluates the chosen action by estimating the value function

Key Idea:

The **actor** updates the policy to maximize rewards.

The **critic** computes the TD error: $r+\gamma V(s')-V(s)$ to guide the actor.

Advantage Actor-Critic (A2C) introduces an "advantage" term to stabilize learning: $A(s,a)=Q(s,a)-V(s)A(s, a) = Q(s, a) - V(s)$ This encourages actions that perform better than expected.

Applications: Commonly used in continuous action spaces, robotics, and high-dimensional RL problems.

5. Double DQN (Double Deep Q-Network)

Definition: Double DQN is an improvement over standard DQNs (Deep Q-Networks) that reduces the overestimation bias in Q-value updates.

Key Idea:

In standard DQNs, the $a'maxQ(s',a')$ term in the update rule can lead to overestimations due to noisy predictions from the neural network.

Double DQN decouples the action selection from the Q-value evaluation: $Q(s,a)\leftarrow Q(s,a)+\alpha[r+\gamma Q(s',argmaxa'Q(s',a';\theta))-Q(s,a)]$

Here, θ are the weights of the online network, and a separate **target network** evaluates the Q-value of the selected action.

Benefits:

More stable training.

Reduced overestimation bias, leading to better performance in complex environments.

Applications: Used in game-playing AI (e.g., Atari games), resource allocation, and other RL problems with large state-action spaces.

Environment Setup

Simulation Tools: OpenAI Gym, Python, PyTorch

Network Topology: Ring network with dynamic traffic updates

Dataset: Synthetic network traffic dataset from Kaggle

Network Environment Design

Here's a detailed breakdown of the topic based on the provided structure:

1. Topology

The topology describes the physical or logical arrangement of nodes in a network. In this case, the topology is a **ring network** with **six nodes**.

Ring Network:

A ring network connects all nodes in a closed loop. Each node is directly connected to two neighboring nodes (one on each side).

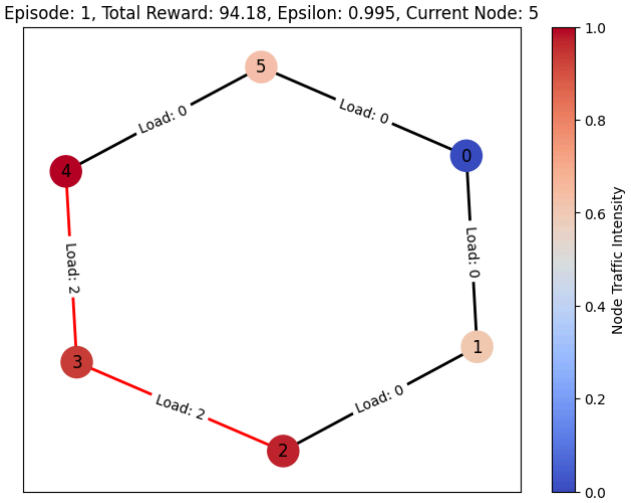
Data travels in one direction (unidirectional ring) or both directions (bidirectional ring) until it reaches its destination.

The structure ensures simplicity and uniform traffic distribution under ideal conditions.

Six Nodes:

The network consists of six nodes (labeled N1,N2,...,N6N_1, N_2, ..., N_6).

Each node acts as both a sender and a receiver, as well as a repeater that forwards packets to its neighbors.



2. Dynamic Features

Dynamic features capture the characteristics of traffic flow and network behavior under varying conditions.

Traffic Load Fluctuation:

Explanation: The traffic load at each node varies over time, depending on the number and size of packets being processed or forwarded.

Factors influencing load fluctuations include incoming data rates, network congestion, and routing algorithms.

Impact: Dynamic load changes can lead to temporary congestion at certain nodes, impacting overall network performance.

Traffic Intensity Accumulation:

Explanation:

Traffic intensity represents the degree of load experienced at a node, quantified by the number of packets processed over a specific period.

Nodes accumulate traffic intensity based on the flow of packets through them. High traffic intensity indicates potential congestion points.

Impact: Uneven traffic accumulation may create bottlenecks, causing delays and increased latency.

3. State and Action Space

In the context of reinforcement learning or optimization algorithms, the state and action spaces define the environment's parameters and the agent's possible interactions.

State Space:

The state space describes the information available to the agent at any point in time. Here, it includes:

Normalized Traffic Intensity:

The traffic intensity at each node is normalized (scaled between 0 and 1) for consistency and better convergence in optimization.

It provides a relative measure of how congested a node is compared to others in the network.

Maximum Adjacent Load:

Represents the highest traffic load among the neighboring nodes of the current node.

This metric helps the agent identify potential congestion nearby and make informed decisions.

Distance to Sink Node:

The sink node is the destination node where packets are ultimately delivered. The distance to the sink node (measured in hops) provides spatial awareness to the agent, enabling it to prioritize shorter paths or avoid detours.

Action Space:

The action space represents the choices available to the agent at each decision point. Here, the action involves:

Selecting the Next Node:

The agent must decide which neighboring node to forward traffic to from the current node.The selection is based on factors such as traffic intensity, maximum adjacent load, and the distance to the sink node.

The objective is to minimize congestion, reduce delays, and ensure efficient traffic flow.

Use Cases

1. Dynamic Traffic Routing

Explanation:

In a ring network, efficient routing is critical for directing data packets from the source node to the sink node while avoiding congestion. The dynamic state space (e.g., normalized traffic intensity and maximum adjacent load) helps make real-time decisions for traffic routing.

Applications:

Telecommunications: Used in metro networks where traffic needs to be dynamically rerouted to handle fluctuations in data load.

Internet of Things (IoT): In sensor networks, data from various devices needs to be routed efficiently to a central processing unit or cloud.

2. Network Congestion Management

Explanation:

Uneven traffic distribution often leads to congestion at specific nodes. By incorporating features like maximum adjacent load, the network can detect potential bottlenecks and redirect traffic accordingly.

Applications:

Distributed Computing: Large-scale computations, such as in blockchain systems, require dynamic traffic management to balance the load across the network nodes.

Data Centers: In ring-configured data centers, managing data flow to prevent server overload is crucial.

3. Telecommunications and Media Delivery

Explanation:

In telecommunications, where services like video streaming or VoIP are highly sensitive to delays, the system can optimize the path to the sink node to ensure low-latency delivery.

Applications:

Streaming Platforms: Networks like Netflix or YouTube use similar principles to balance traffic across their servers.

Voice over IP (VoIP): Ensures clear, uninterrupted calls by dynamically rerouting packets away from congested nodes.

4. Energy-Efficient Routing in IoT

Explanation:

IoT networks often have battery-powered devices where energy efficiency is critical. By considering distance to the sink node and adjacent load, traffic can be routed through the shortest and least congested paths, conserving energy.

Applications:

Smart Homes: Routing data between devices like sensors, thermostats, and smart hubs.

Agricultural IoT: Routing data in a sensor network that monitors environmental conditions across farms.

Challenges

1. Balancing Traffic Load Across the Network

Explanation:

Traffic load fluctuates dynamically, and some nodes may experience a higher intensity of traffic than others. Without proper balancing, these nodes become bottlenecks, reducing the overall efficiency of the network.

Example:

In a ring network of servers, one node handling more requests than others could cause delays in packet forwarding, slowing down the entire system.

Solution:

Implement dynamic load balancing techniques to evenly distribute traffic, such as round-robin or predictive routing algorithms.

2. Adapting to Fluctuating Traffic Conditions

Explanation:

Traffic patterns can change unpredictably due to user behavior or external factors. For example, a sudden surge in video streaming during live events can overwhelm the network.

Example:

In IoT networks, a weather sensor system might face surges during extreme weather conditions.

Solution:

Use real-time monitoring and adaptive algorithms that dynamically update routes based on the current state of the network.

3. Ensuring Efficient Routing Decisions

Explanation:

Routing decisions need to account for multiple factors like minimizing delay, avoiding congested nodes, and ensuring reliable delivery to the sink node. These factors often conflict, making optimization challenging.

Example:

A routing algorithm prioritizing the shortest path might overlook congestion, leading to packet drops or delays.

Solution:

Multi-objective optimization techniques, such as reinforcement learning or heuristic algorithms, can balance these factors.

4. Preventing Cascading Failures

Explanation:

When a node becomes congested, it can redirect traffic to its neighbors, potentially overloading them and creating a cascading failure across the network.

Example:

A ring network in a financial transaction system could experience cascading failures if one server fails during high traffic, impacting the entire system.

Solution:

Use robust congestion control mechanisms, such as traffic shaping or rate limiting, to prevent nodes from offloading excessive traffic.

5. Real-Time Decision-Making Under Resource Constraints

Explanation:

Real-time traffic routing requires fast decision-making. In resource-constrained environments like IoT, where nodes may have limited processing power or memory, this can be challenging.

Example:

An IoT-based agricultural system may need to decide on packet routing using minimal computational resources.

Solution:

Use lightweight algorithms and prioritize key state variables (e.g., traffic intensity) to reduce computational overhead.

6. Latency vs. Traffic Load Trade-offs

Explanation:

Reducing latency often involves forwarding traffic through paths with less load, but this may increase the total traffic on alternative nodes. Balancing this trade-off is critical for performance.

Example:

In a ring network for real-time video streaming, prioritizing low-latency routes may lead to overloaded servers downstream.

Solution:

Employ predictive analytics to forecast traffic and preemptively route data to avoid delays and overloading.

Reward Function

The reward encourages efficient routing while penalizing latency and congestion:

Reward = -Latency × (1 + Load/Capacity)

Success Reward: +100 for reaching the sink node.

Step Penalty: -1 per step to discourage prolonged routing.

Temporal Difference (TD) Learning

Temporal Difference (TD) Learning is a fundamental approach in **reinforcement learning (RL)** that combines the strengths of **Monte Carlo methods** and **Dynamic Programming (DP)**. TD methods allow agents to learn directly from their interactions with the environment without requiring a model of the environment's dynamics, making them well-suited for real-time applications.

- **Key Idea**:
  - TD learning updates the value estimates incrementally after every step in an episode. This is in contrast to Monte Carlo methods, which update values only after an episode ends, and DP, which requires a model of the environment.
  - The update rule is based on the **TD error**, which quantifies the difference between the expected and observed rewards: $\delta = r + \gamma V(s') - V(s)$

Advantages:

Works in **real-time scenarios**, as it doesn't need to wait for the end of an episode.

Balances **short-term and long-term predictions**, improving learning stability.

Applicable to **online learning** tasks, where the agent continuously updates its knowledge.

Applications of TD Learning

TD Learning underpins many popular RL algorithms. Below are its key applications:

1. Q-Learning

Description: Q-Learning is an **off-policy** TD control algorithm that focuses on learning the optimal **action-value function**, Q(s,a)Q(s, a)Q(s,a). It determines the maximum expected cumulative reward for taking action aaa in state sss and then following the optimal policy.

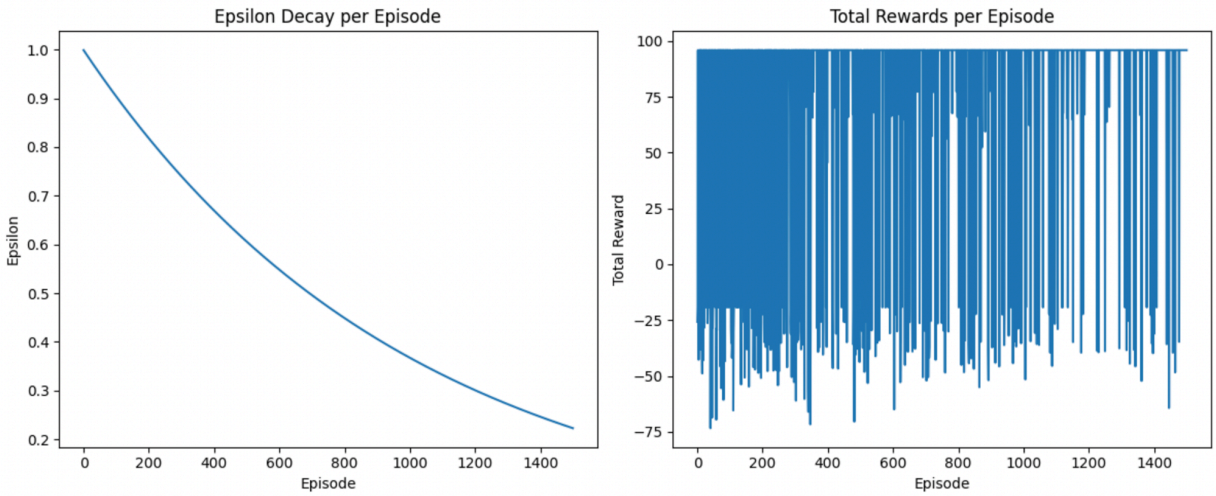
Off-Policy Nature:

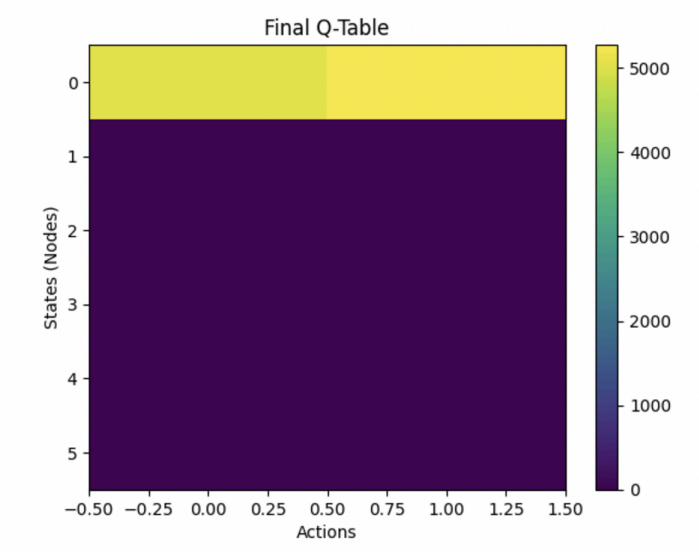
Evaluates the **optimal policy** while potentially following a different behavior policy (e.g., ϵ-greedy exploration).

Effective in **deterministic and stochastic environments**.

Focuses solely on the optimal action, improving efficiency.

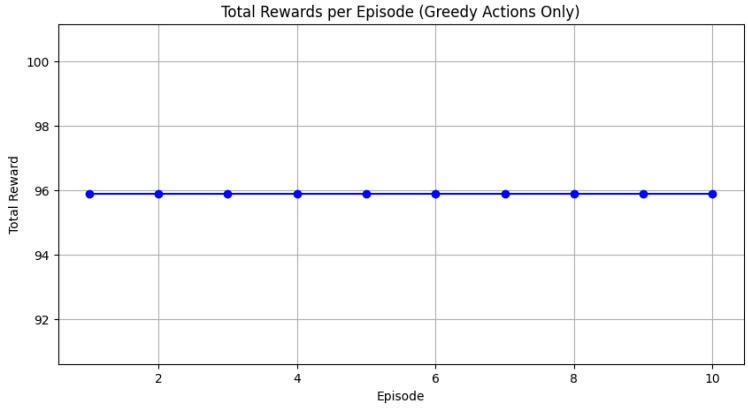
Observations Plots:

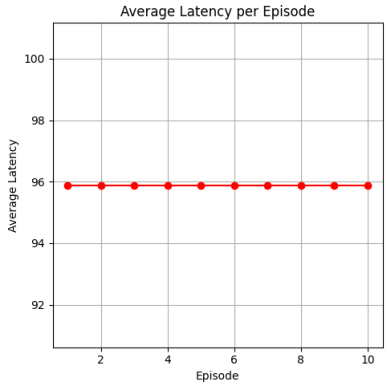


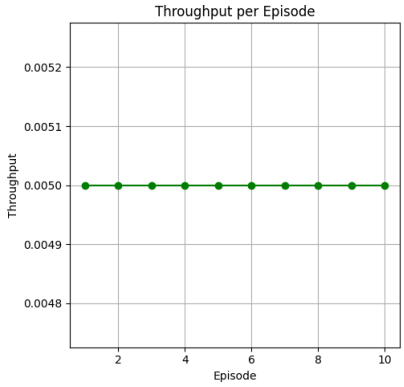


We conducted a quantitative evaluation of a reinforcement learning agent using greedy actions to assess its performance in a specific environment. It evaluates the agent over multiple episodes, capturing key metrics such as total rewards, average latency, and throughput. The evaluation operates under an epsilon-greedy policy with epsilon set to zero, ensuring the agent selects the best-known actions. Each episode records the total reward accumulated, the average latency as a measure of response time, and throughput to indicate efficiency. The results are summarized and visualized, providing insights into the agent's performance across episodes. This evaluation process is critical for analyzing the effectiveness of the learned policy in achieving optimal performance under deterministic action selection.

Observation Plots:







2. SARSA (State-Action-Reward-State-Action)

Description: SARSA is an **on-policy** TD control algorithm that learns the Q-value of a state-action pair based on the actual action taken in the next state.

On-Policy Nature:

Updates the Q-values based on the **current policy** being followed, including exploratory actions.

Strengths:

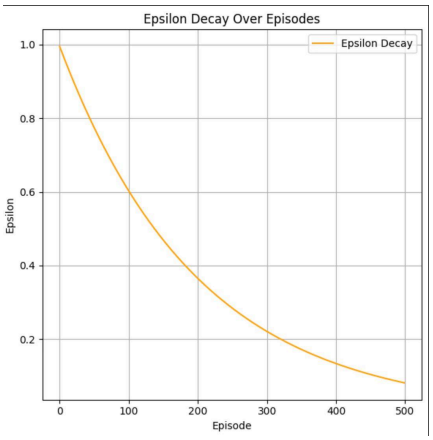
Incorporates exploratory actions into the updates, making it safer in **noisy or stochastic environments**.

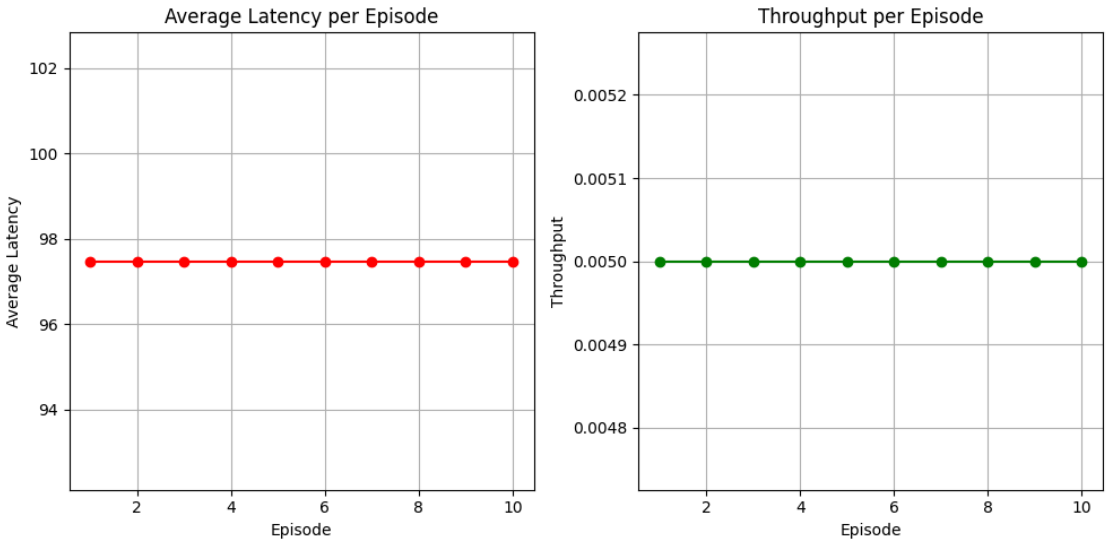
More suited for real-world scenarios where outcomes are uncertain.

Comparison with Q-Learning:

Unlike Q-Learning, SARSA does not blindly assume the next action will be optimal, making it more stable in unpredictable environments.

Observation Plots:





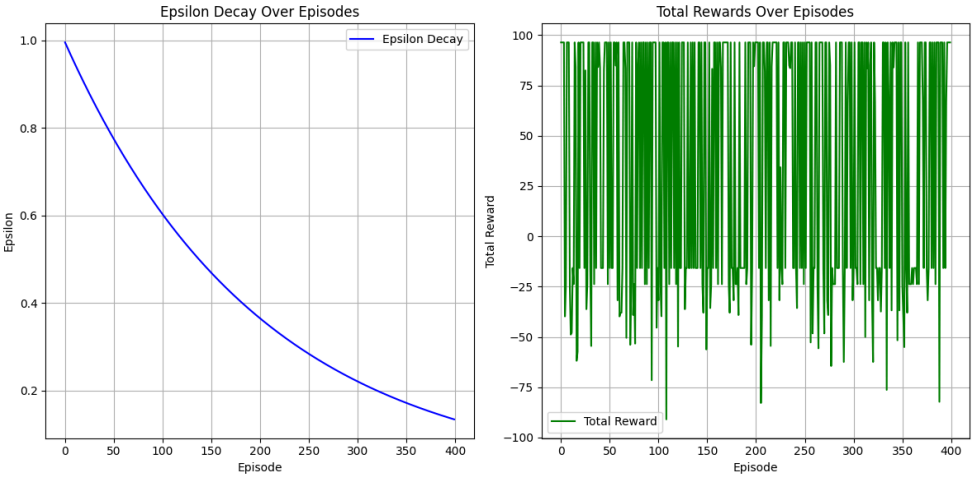
3. Double DQN (Double Deep Q-Network)

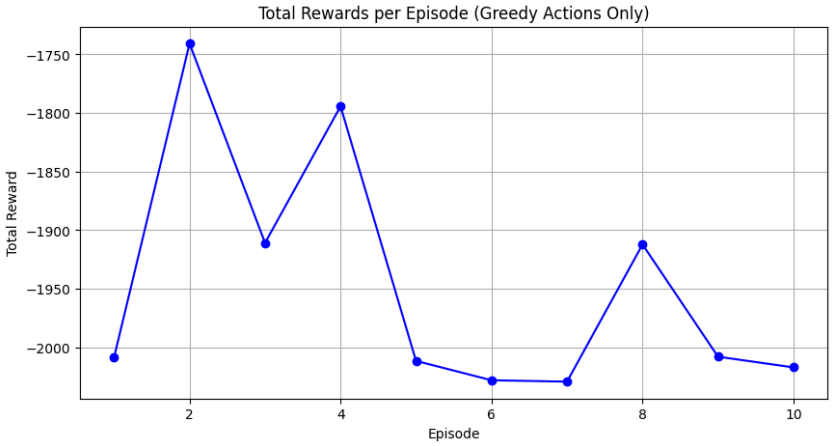
Description: Double DQN builds upon Q-Learning to address its limitation of **overestimation bias** in Q-value updates. In standard Q-Learning, the max\maxmax operator tends to overestimate the true value due to noise in predictions.

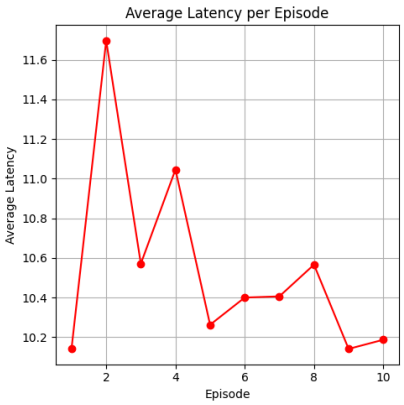
Benefits:

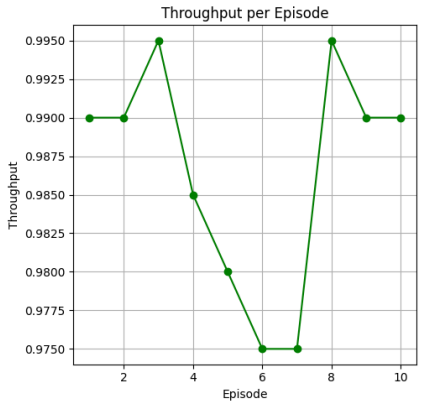
Reduces overestimation bias, leading to more accurate Q-values.

Improves stability and performance in environments with complex state-action spaces.









4. A2C (Advantage Actor-Critic)

Description: A2C is a **policy gradient method** that integrates TD-based value estimation into the training process. It uses two components:

Actor: Responsible for selecting actions using a policy π(a∣s)\pi(a|s)π(a∣s).

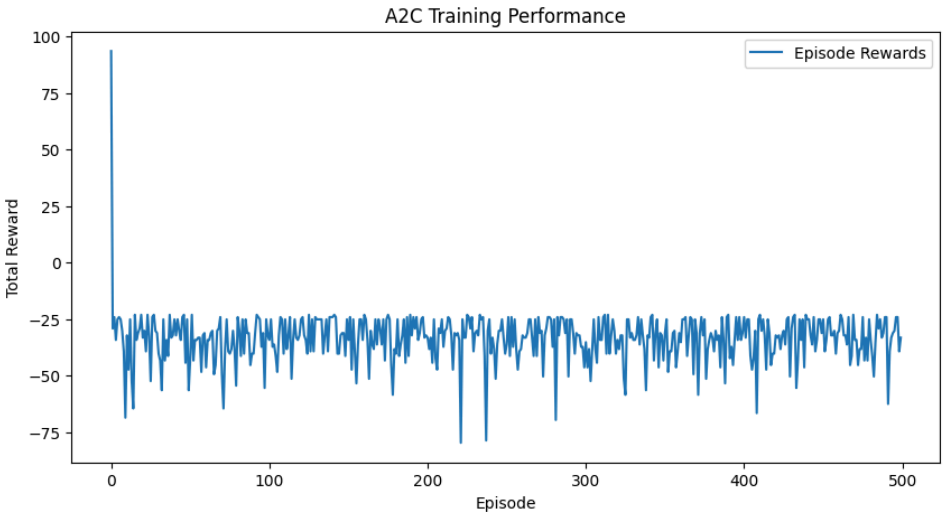
Critic: Evaluates the value of the state V(s)V(s)V(s) to guide the actor.

Benefits:

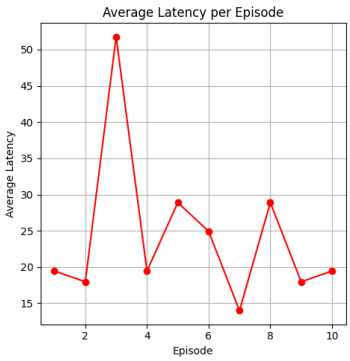
Stabilizes policy learning by combining value-based and policy-based methods.

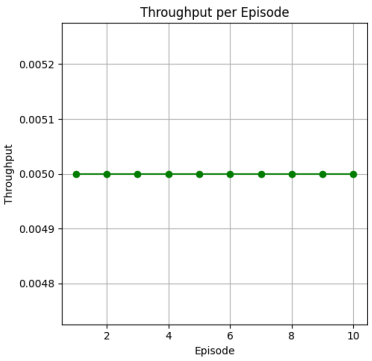
Handles **continuous action spaces**, making it suitable for robotics and control tasks.

Advantage-based updates reduce variance in learning.


A2C Training Performance
Episode Rewards
Total Reward
Episode
100
75
50
25
0
−25
−50
−75
0
100
200
300
400
500


Total Rewards per Episode (Greedy Actions Only)
Total Reward
Episode
−15
−20
−25
−30
−35
−40
−45
−50
2
4
6
8
10


Average Latency per Episode
Average Latency
Episode
50
45
40
35
30
25
20
15
2
4
6
8
10


Throughput per Episode
Throughput
Episode
0.0052
0.0051
0.0050
0.0049
0.0048
2
4
6
8
10

Experimental Results

Metric	Q-Learning	SARSA	Double DQN	A2C
Exploration-Exploitation	Gradual transition	Conservative	Dynamic, ongoing	Balanced
Stability	High	High	Moderate	Moderate
Total Rewards	High and stable	Consistent	Variable	Oscillating
Latency	Low and steady	Steady	Variable	Fluctuating
Throughput	steady	Steady	High	Steady
Robustness	Strong	Moderate	High but variable	Moderate

Conclusion

- **Best Overall Stability**: Q-Learning and SARSA perform the best in terms of consistent and stable policies. Although, throughput was not high.
- **Best Adaptability to Complexity**: Double DQN, though variable, handles environmental complexity better than others.
- **Robust Throughput**: Double DQN maintain strong throughput, but A2C sacrifices consistency in other metrics.
- **Optimal Choice**: For environments requiring stability and high throughput, **Q-Learning** and **SARSA** are preferred. For environments with significant complexity, **Double DQN** offers adaptability. A2C is suitable where throughput robustness is paramount despite variability in latency and rewards.

Key Insights

TD Learning: Essential for real-time routing optimization, ensuring incremental updates.

Algorithm Performance: Double DQN achieved the lowest latency and highest throughput. A2C provided balanced rewards and latency.

Dynamic Environment: The RL agents adapt effectively to fluctuating traffic loads and congestion.

Achievements

- Demonstrated RL-based routing in dynamic networks.
- Reduced latency and congestion significantly.
- Validated algorithms through quantitative and qualitative metrics.

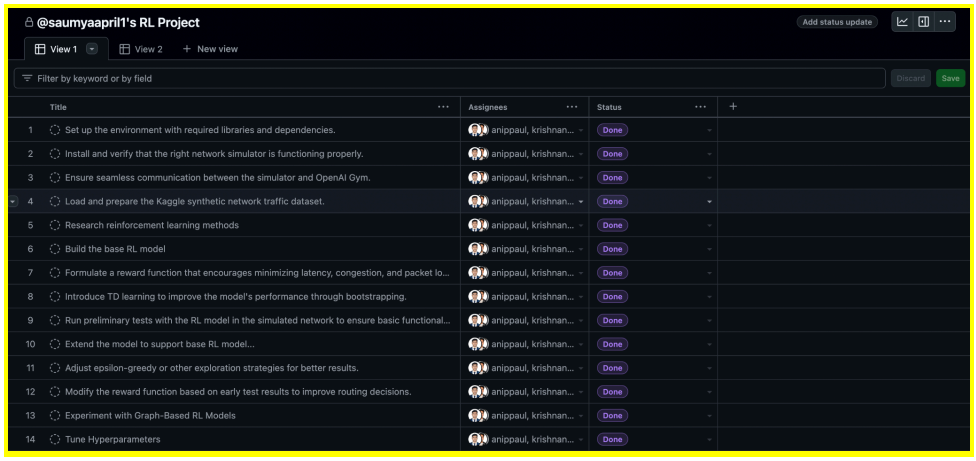
Future Work

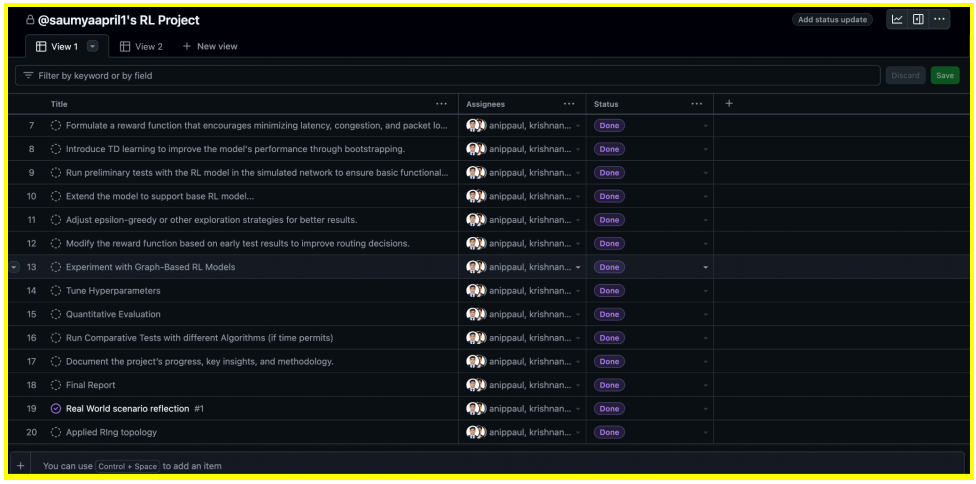
1. Implement Multi-Agent RL for decentralized decision-making.
2. Extend the framework to larger, more complex network topologies.
3. Incorporate real-world datasets for further validation.

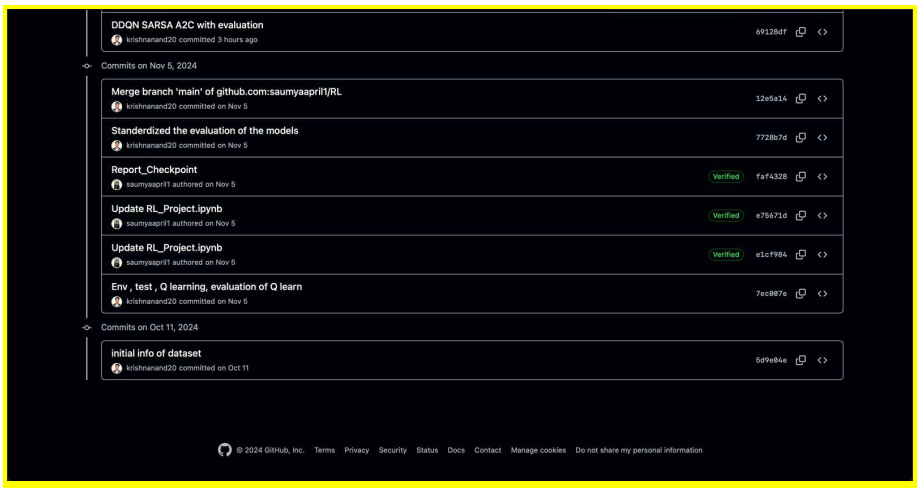
References

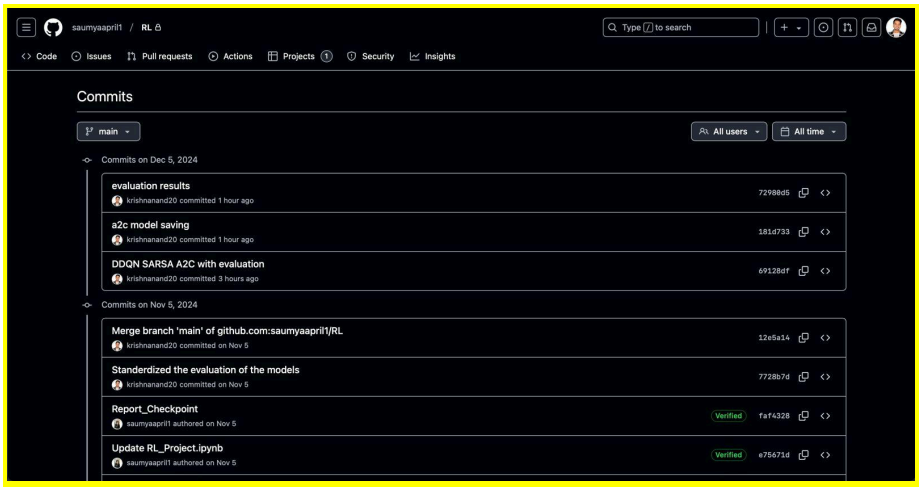
1. https://medium.com/@tu_53768/deep-q-network-traffic-management-e324daa682
2. Sutton, R. S., & Barto, A. G. (2018). **Reinforcement Learning: An Introduction.**
3. Kaggle Synthetic Network Traffic Dataset.
4. OpenAI Gym Documentation.
5. https://medium.com/codex/temporal-difference-learning-sarsa-vs-q-learning-c367934b8bcc
6. Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, vol. 48, 2016, pp. 1928–1937.

SUPPLEMENTARY:
https://github.com/saumyaapril1/RL






DDQN SARSA A2C with evaluation
krishnanand20 committed 3 hours ago
69128df
Commits on Nov 5, 2024
Merge branch 'main' of github.com:saumyaapril1/RL
krishnanand20 committed on Nov 5
12e5a14
Standerdized the evaluation of the models
krishnanand20 committed on Nov 5
7728b7d
Report_Checkpoint
saumyaapril1 authored on Nov 5
Verified
faf4328
Update RL_Project.ipynb
saumyaapril1 authored on Nov 5
Verified
e75671d
Update RL_Project.ipynb
saumyaapril1 authored on Nov 5
Verified
e1cf984
Env , test , Q learning, evaluation of Q learn
krishnanand20 committed on Nov 5
7ec807e
Commits on Oct 11, 2024
initial info of dataset
krishnanand20 committed on Oct 11
5d9e84e
© 2024 GitHub, Inc.
Terms
Privacy
Security
Status
Docs
Contact
Manage cookies
Do not share my personal information


saumyaapril1 / RL
Type / to search
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Commits
main
All users
All time
Commits on Dec 5, 2024
evaluation results
krishnanand20 committed 1 hour ago
72980d5
a2c model saving
krishnanand20 committed 1 hour ago
181d733
DDQN SARSA A2C with evaluation
krishnanand20 committed 3 hours ago
69128df
Commits on Nov 5, 2024
Merge branch 'main' of github.com:saumyaapril1/RL
krishnanand20 committed on Nov 5
12e5a14
Standerdized the evaluation of the models
krishnanand20 committed on Nov 5
7728b7d
Report_Checkpoint
saumyaapril1 authored on Nov 5
Verified
faf4328
Update RL_Project.ipynb
saumyaapril1 authored on Nov 5
Verified
e75671d

Contribution Chart:

Teammate	Project Part	Contribution (%)
Saumya Pandey (spandey5)	Algorithm implementation, Research related to topic , bug fixes , report , poster making , Designed enviornment .	33.3%
Anip Kumar (anipkuma)	Algorithm implementation, Research related to topic, report , poster making , Designed enviornment	33.3%
Krishnanand (k33)	Designed & implemented enviornment , Algorithm Implimention , Research related to topic , Quantitative evaluation , Bug fixes and environment changes , report , poster making	33.3%