Enhanced Autonomous Navigation with Regularized Convolutional Neural Fitted Q Iteration

Rohit Sai Suresh
UID: 119283684
University of Maryland, College Park
rohitsai@umd.edu

Vishnu Mandala
UID: 119452608
University of Maryland, College Park
vishnum@umd.edu

Abstract

This project examines the development and preliminary evaluation of an autonomous navigation system designed for operation in densely trafficked environments, employing the Regularized Convolutional Neural Fitted Q Iteration (RC-NFQ) algorithm. The RC-NFQ algorithm, an enhancement of the Neural Fitted Q Iteration (NFQ), integrates convolutional neural networks (CNNs) and dropout regularization to better handle and generalize from high-dimensional sensory data in dynamic driving scenarios. Unlike standard approaches such as the Deep Q-Networks (DQN), which have been extensively tested, RC-NFQ has not been widely examined in other research settings, making this project among the first to deploy it in a complex, real-time simulation environment.

The methodology involved implementing RC-NFQ within the HighwayEnv simulation platform, a specialized environment tailored for autonomous driving research. This platform supports various traffic scenarios crucial for training and evaluating AI-based driving models. The RC-NFQ model was designed with multiple convolutional and dense layers, employing dropout after each dense layer to mitigate overfitting. Training utilized the RMSprop optimizer and an experience replay buffer to enhance learning stability. The system was tested in scenarios with varying traffic densities to evaluate its performance in dynamic environments.

Key findings from the initial simulations indicate that RC-NFQ slightly overperforms the DQN reference model in terms of learning efficiency and policy quality. Specifically, RC-NFQ demonstrated a marginal improvement in its ability to adapt to dynamic traffic patterns and make real-time navigation decisions, reducing the incidence of navigational errors and improving overall traffic flow efficiency. These results, though preliminary, suggest that RC-NFQ offers potential advantages in handling complex, high-dimensional input data and maintaining robust performance under varying conditions.

The significance of these results lies in the novel application of RC-NFQ to autonomous vehicle navigation, specifically tailored to demonstrate understanding of a fundamental principle in engineering. Given the algorithm's limited prior testing, this project provides crucial initial insights into its potential efficacy. The findings support further investigation into RC-NFQ's capabilities and optimization, with implications for advancing autonomous driving technology and enhancing the safety and efficiency of future transportation systems. This study contributes to the broader field of autonomous navigation by exploring and validating a relatively untested yet promising machine learning approach.

1. Introduction

The advent of autonomous vehicles marks a significant technological advancement with the potential to transform transportation systems. Autonomous vehicles, also known as self-driving cars, leverage a combination of advanced sensors, machine learning algorithms, and sophisticated control systems to navigate and make decisions without human intervention. These vehicles promise substantial improvements in road safety, traffic efficiency, and accessibility, potentially reducing accidents caused by human error, optimizing traffic flow, and providing mobility solutions for individuals unable to drive.

Despite these benefits, the deployment of autonomous vehicles in real-world settings presents numerous challenges. One of the most critical challenges is the ability of these vehicles to operate reliably in complex and dynamically changing environments, such as urban areas with dense traffic. Autonomous vehicles must interpret vast amounts of sensory data in real-time, make rapid and accurate decisions, and adapt to unpredictable situations, such as sudden lane changes by other drivers or unexpected obstacles on the road.

1.1. Problem Statement

Operating in dynamically changing and densely populated environments poses substantial challenges for autonomous vehicles, particularly in the realm of decision-making under uncertainty. Traditional algorithms, such as rule-based systems and classical reinforcement learning methods, often fall short when confronted with high-

dimensional state spaces and the complex decision landscapes typical in urban traffic scenarios. These algorithms struggle to generalize from training data to real-world situations, leading to suboptimal performance and potential safety risks.

Reinforcement learning (RL) algorithms, which allow an agent to learn optimal policies through interactions with the environment, have shown promise in addressing some of these challenges. However, standard RL approaches like Deep Q-Networks (DQN) are limited by their inability to efficiently process high-dimensional sensory inputs, such as images from vehicle-mounted cameras. This limitation hampers the autonomous vehicle's ability to make well-informed decisions in real-time, reducing its effectiveness in complex driving environments.

1.2. Objectives

The primary objective of this project is to implement and evaluate the Regularized Convolutional Neural Fitted Q Iteration (RC-NFQ) algorithm within a simulated environment that closely mimics real-world traffic conditions. This advanced deep learning approach aims to enhance the decision-making capabilities of autonomous vehicles in dense traffic environments by integrating convolutional neural networks (CNNs) and dropout regularization into the traditional Neural Fitted Q Iteration (NFQ) framework.

Specific objectives of the project include:

- Designing and implementing the RC-NFQ algorithm: This involves developing a model architecture that combines CNNs and dropout regularization to process high-dimensional inputs effectively.
- Configuring the simulation environment: Using HighwayEnv, a specialized platform for autonomous driving research, to create realistic traffic scenarios for training and testing the RC-NFQ algorithm.
- Training and evaluating the model: Conducting extensive simulations to assess the performance of RC-NFQ in various traffic conditions, comparing its results with those obtained using the DQN algorithm.
- Analyzing the results: Interpreting the findings to determine the effectiveness of RC-NFQ in enhancing autonomous vehicle navigation and identifying areas for further improvement.

1.3. Scope

The scope of this project encompasses the following key components:

- Algorithm Development: Implementing the RC-NFQ algorithm, which includes defining the model architecture, selecting appropriate hyperparameters, and integrating CNNs and dropout regularization.
- Simulation Environment: Setting up the HighwayEnv platform to simulate realistic traffic scenarios, including varying traffic densities, different road types, and diverse driving behaviors.
- Training Procedures: Developing a comprehensive training protocol that includes the use of an experience replay buffer, mini-batch learning, and the RMSprop optimizer to enhance learning stability and efficiency.
- Performance Evaluation: Assessing the RC-NFQ algorithm's performance through extensive simulations, using metrics such as collision rates, lane adherence, and overall traffic flow efficiency.
- Comparative Analysis: Comparing the performance of RC-NFQ with the DQN algorithm to highlight the improvements and advantages offered by the proposed approach.

1.4. Significance

The significance of this project lies in its potential to advance the field of autonomous driving by introducing and evaluating a relatively untested machine learning algorithm. RC-NFQ combines the strengths of CNNs and reinforcement learning to address the challenges posed by high-dimensional sensory inputs and dynamic environments. By demonstrating the algorithm's effectiveness in a realistic simulation, this project provides valuable insights into its potential for real-world applications, paving the way for further research and development in autonomous navigation systems.

Key contributions and significance include:

- Novelty: This project is among the first to implement and evaluate RC-NFQ in the context of autonomous vehicle navigation, providing new insights into its capabilities and limitations.
- Practical Relevance: The findings have direct implications for the design and development of more robust and efficient autonomous navigation systems, potentially improving safety and efficiency in real-world traffic environments.
- Research Foundation: The project establishes a foundation for future research, identifying key areas for improvement and optimization in the RC-NFQ algorithm and its application to autonomous driving.
- Technological Advancement: By leveraging advanced machine learning techniques, this project contributes to the broader goal of developing intelligent transportation systems that can adapt to and navigate complex, dynamic environments.

2. Literature Survey

2.1. Overview

The field of autonomous vehicle navigation has experienced significant advancements due to the integration of machine learning techniques, particularly reinforcement learning (RL) and convolutional neural networks (CNNs). This literature survey reviews existing research on these techniques, highlighting key algorithms and their applications in autonomous driving. It also identifies gaps in the current literature and explains why the Regularized Convolutional Neural Fitted Q Iteration (RC-NFQ) algorithm was selected for this project.

2.2. Reinforcement Learning in Autonomous Navigation

2.2.1 Q-Learning

Q-learning is a foundational model-free reinforcement learning algorithm designed to find the optimal action-selection policy for any given finite Markov decision process. It works by learning the value of state-action pairs, called Q-values, which represent the expected utility of taking a given action in a given state and following the optimal policy thereafter. The Q-values are updated iteratively using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \quad (1)$$

where α is the learning rate, r is the reward received, and γ is the discount factor for future rewards. Q-learning's simplicity and effectiveness have made it a popular choice for various RL applications, but its ability to scale to high-dimensional spaces is limited.

2.2.2 Deep Q-Networks (DQN)

Deep Q-Networks (DQN), introduced by Mnih et al. (2015), extend Q-learning by integrating deep learning to handle high-dimensional sensory inputs. A deep neural network, referred to as the Q-network, is used to approximate the Q-values. This approach enables the algorithm to learn directly from raw sensory data such as images, making it suitable for complex tasks like playing video games and autonomous driving. The key innovations in DQN include:

- Experience Replay: A buffer that stores transitions observed during training, which are sampled randomly to break the correlation between consecutive updates and improve stability.
- Target Network: A separate network with fixed parameters for a set number of iterations, which stabilizes the learning by reducing the correlations between the Q-value predictions and the target Q-values.

2.2.3 Neural Fitted Q Iteration (NFQ)

Neural Fitted Q Iteration (NFQ), proposed by Riedmiller (2005), extends the traditional Q-learning algorithm by using neural networks to approximate the Q-function. Unlike DQN, which updates Q-values online, NFQ performs batch updates using a dataset of transitions. This method enhances stability and convergence, making it effective for various robotic control tasks. NFQ's ability to learn from pre-collected batches of experiences allows it to be data-efficient, a crucial aspect for tasks where data collection is expensive or time-consuming.

2.3. Convolutional Neural Networks in Autonomous Navigation

2.3.1 Overview of CNNs

Convolutional Neural Networks (CNNs) are a class of deep neural networks particularly effective for processing data with a grid-like topology, such as images. CNNs consist of convolutional layers that apply filters to the input data to extract features, followed by pooling layers that downsample the data to reduce dimensionality and computational load. CNNs have demonstrated significant success in various domains, including image classification, object detection, and, more recently, autonomous driving.

2.3.2 Key Components of CNNs

- Convolutional Layers: These layers apply a set of filters to the input image to create feature maps. Each filter detects different features such as edges, textures, or shapes.
- Activation Functions: Non-linear functions like ReLU (Rectified Linear Unit) are applied to the feature maps to introduce non-linearity into the model, allowing it to learn more complex patterns.
- Pooling Layers: These layers reduce the spatial dimensions of the feature maps, typically through max-pooling, which retains the most important features while discarding less critical information.
- Fully Connected Layers: After several convolutional and pooling layers, the high-level reasoning in the network is performed through fully connected layers that output the final predictions.

2.3.3 *Applications in Autonomous Driving*

CNNs have been widely used in autonomous driving for tasks such as lane detection, object recognition, and scene understanding. Bojarski et al. (2016) demonstrated an end-to-end learning approach where a CNN maps raw pixels from a front-facing camera directly to steering commands. This approach significantly simplifies the pipeline for autonomous driving, eliminating the need for hand-engineered features and intermediate steps.

2.4. Regularized Convolutional Neural Fitted Q Iteration (RC-NFQ)

2.4.1 *Algorithm Overview*

RC-NFQ combines the NFQ framework with CNNs and dropout regularization to improve generalization and prevent overfitting. Dropout, a regularization technique introduced by Srivastava et al. (2014), involves randomly setting a fraction of the input units to zero at each update during training, which prevents neurons from co-adapting too much.

RC-NFQ was developed to address the limitations of DQN and NFQ in handling high-dimensional sensory inputs and dynamic environments. By integrating CNNs, RC-NFQ can effectively process visual data, while dropout regularization enhances the model's robustness and ability to generalize from training to real-world scenarios.

2.4.2 *Initial Implementations and Evaluations*

Initial implementations of RC-NFQ, as documented by Harrigan (2015), showed promise in controlled environments but had not been extensively tested in more complex, real-world scenarios until this project. Harrigan's work laid the foundation for the integration of CNNs and regularization techniques into the NFQ framework, demonstrating improvements in learning efficiency and policy quality.

2.4.3 *Implementation Details*

- Convolutional Layers: RC-NFQ utilizes multiple convolutional layers to extract hierarchical features from raw pixel data. Each layer detects increasingly complex features, from simple edges to full objects.
- Dropout Regularization: Dropout is applied after each convolutional and dense layer to mitigate overfitting by randomly disabling a portion of neurons during training.
- Experience Replay: An experience replay buffer stores transitions collected during training, allowing the network to learn from a diverse set of experiences and break the correlation between consecutive updates.
- Training Procedure: RC-NFQ is trained using the RMSprop optimizer, which is effective for non-stationary objectives and works well in on-policy learning tasks.

2.4.4 *Pseudocode*

Algorithm 1 Estimate the action-value function using a convolutional neural network with dropout regularization

Input:

E is an environment, specifying the space of states, actions and rewards

C is an architecture for a convolutional neural network with dropout regularization layers

hyperparameters

Output:

Returns a learned action-value function Q

procedure RC-NFQ (E, C, *hyperparameters*)

Parameterize identical convolutional neural network models Q_0 and $\hat{Q}_0$ which will be used to learn to approximate the action-value function, using the architecture specified in C and the dropout probability α_{drop} for the dropout regularization layers

Initialize action-value function Q_0 with a parameter vector θ of random initial weights

Initialize target action-value function $\hat{Q}_0$ with $\theta^- = \theta$

Initialize experience replay buffer D

for episode $i = 0$; α_{eps} do

Initialize temporary experience buffer D̃

for t = 1; α_{len} do

Select random action with probability ε

Otherwise, select action a maximizing the action-value function $Q_i(s; a)$

Execute action a in environment E and observe r and s'

Store transition (s, a, r, s') in D̃

end for

Append D̃ to the experience replay buffer D

for iteration k = 0; α_{iters} do

Sample random batch of $\alpha_{samples}$ from D and store in D'

Generate a pattern set of training targets where $y_i = D'^{r} + \gamma\hat{Q}_i(D'^{s'}, D'^{a})$

Call RMSprop with learning rate α_{lr} to perform gradient descent on $(y_i - Q_i(D's, D'^{a}; \theta))^2$ and store the updated parameters in Q_{i+1}

if k is a multiple of α_{freq} then

Update target action-value function $\hat{Q}_{i+1}$ with parameters from Q_i

else

Copy the parameters from $\hat{Q}_i$ to $\hat{Q}_{i+1}$

end if

end for

end for

return $Q\alpha_{eps}$

end procedure

Table 1. Hyperparameters for the RC-NFQ Algorithm

Hyperparameters	Description
γ	discount factor for future rewards
α_{lr}	learning rate for RMSprop
α_{freq}	frequency at which the target Q-network is updated
α_{iters}	number of iterations of fitted Q-iteration to run between episodes
α_{len}	length of each episode
α_{eps}	number of episodes
α_{samples}	number of samples to use within each iteration of fitted Q-iteration
α_{drop}	dropout probability for the dropout regularization layers

2.5. HighwayEnv Simulation Platform

2.5.1 Platform Overview

HighwayEnv is a specialized simulation platform designed for autonomous driving research. Developed as an extension of OpenAI Gym, HighwayEnv provides a realistic and controlled setup for testing various aspects of autonomous vehicle algorithms, from basic steering and speed control to advanced behavioral decision-making under various traffic conditions.

2.5.2 Core Features and Scenarios

HighwayEnv supports a variety of scenarios crucial for training autonomous vehicles:

- Highway: Tasks involve achieving high speeds, lane following, and avoiding collisions in multi-lane traffic settings.
- Merge: Focuses on merging safely with incoming traffic from ramps without causing disruptions in the flow.
- Roundabout: Challenges the agent to navigate roundabouts efficiently while maintaining safe interactions with other vehicles.
- Parking: Tests the agent's ability to park precisely in designated spots.
- Intersection: Involves managing vehicle behavior at busy intersections to optimize flow and prevent accidents.
- Racetrack: Combines the need for speed with obstacle avoidance in a controlled track environment.

2.5.3 Customizability and Extensibility

HighwayEnv allows users to customize the environments to test different vehicle behaviors and traffic scenarios. Configuration options enable adjustments in terms of traffic density, speed limits, and behavior of other drivers, providing a rich dataset for training reinforcement learning models.

Figure 1. Snapshot of the Highway Environment used

2.5.4 Use in this project

In this project, HighwayEnv was used to create a realistic and controlled environment (Highway model) for training and evaluating the RC-NFQ algorithm. The platform's ability to simulate diverse traffic scenarios allowed for comprehensive testing of the algorithm's performance under varying conditions. The choice of HighwayEnv was driven by its stability and support for a wide range of traffic situations, making it ideal for developing and evaluating advanced autonomous driving models.

2.6. Gaps in Literature

Despite significant advancements in the field, several gaps remain:

- Limited Testing of RC-NFQ: While initial implementations of RC-NFQ have shown promise, the algorithm has not been extensively tested in complex, real-world-like environments. This project aims to address this gap by evaluating RC-NFQ in the HighwayEnv simulation platform.
- High-Dimensional Input Handling: Traditional reinforcement learning algorithms struggle with high-dimensional sensory inputs, such as images. RC-NFQ aims to address this by integrating CNNs, but further research is needed to optimize this integration.
- Generalization and Robustness: Ensuring that autonomous navigation algorithms can generalize from training data to real-world scenarios remains a challenge. The use of dropout regularization in RC-NFQ is a step towards addressing this issue, but additional techniques and testing are necessary.

3. Methodology

3.1. Overview

This section details the methodology used to implement, train, and evaluate the Regularized Convolutional Neural Fitted Q Iteration (RC-NFQ) algorithm within a simulated environment designed to mimic real-world traffic

conditions. The approach includes configuring the neural network architecture, preparing the simulation environment, defining the training protocols, and setting hyperparameters. The objective is to furnish a complete account that facilitates replication and validation of the findings.

3.2. Algorithm Design

3.2.1 Convolutional Neural Network (CNN) Configuration

Due to the limitations of the 2D simulation environment in providing image data, the project adapted by using positions and velocities as inputs, emulating the structure of image data. The CNN architecture was designed to process these inputs effectively:

- Input Layer: The network accepts a structured input vector representing the positional and velocity data of surrounding vehicles relative to the ego vehicle.
- Convolutional Layers:
  - First Convolutional Layer: Features 16 filters with an 8x8 kernel size and a stride of 4, employing ReLU activation. This layer is crucial for capturing basic spatial relationships such as relative positioning and movement direction.
  - Second Convolutional Layer: Contains 32 filters, a 4x4 kernel, and a stride of 2 with ReLU activation. It builds on the initial features extracted, focusing on more complex patterns that are critical for predicting traffic behavior.
- Flattening Layer: This layer transforms the multi-dimensional output of the last convolutional layer into a one-dimensional array that can be fed into fully connected layers.
- Fully Connected Layers:
  - First Layer: Comprises 256 neurons with ReLU activation, integrating the features into a format useful for making driving decisions.
  - Dropout Layer: Following the first dense layer, a dropout of 25% is applied to prevent overfitting and ensure that the model generalizes well to new, unseen scenarios.
  - Second Layer: Mirrors the first in size and function but further refines the decision capability.
  - Final Dropout: Another 25% dropout is applied to enhance model robustness.
- Output Layer: Produces a set of action values corresponding to different driving actions, facilitating decision-making based on the learned policies.

Table 2. RC-NFQ Network Architecture

Layer Type	**Details**
Input Layer	Input vector with positional and velocity data
Convolutional Layer 1	16 filters, 8x8 kernel, stride 4, ReLU activation
Convolutional Layer 2	32 filters, 4x4 kernel, stride 2, ReLU activation
Flattening Layer	Converts 3D feature maps to 1D vectors
Fully Connected Layer 1	256 neurons, ReLU activation
Dropout Layer 1	Dropout rate of 0.25 after fully connected layer
Fully Connected Layer 2	256 neurons, ReLU activation
Dropout Layer 2	Dropout rate of 0.25 after fully connected layer
Output Layer	Action values for each possible action (0-4)

3.2.2 Reinforcement Learning Framework

RC-NFQ's reinforcement learning framework integrates several components to facilitate the autonomous navigation of vehicles in simulated environments:

- Initialization: The Q-function, represented by the neural network, starts with randomly initialized weights, setting the baseline for learning.
- Experience Replay Mechanism: A crucial component for stable learning, this buffer stores past state transitions, which are randomly sampled to decouple the sequences and reduce the variance in updates.
- Mini-batch Learning: Periodically, the network samples mini-batches from the experience replay buffer. This approach balances the computational efficiency with the benefit derived from batch learning.
- Q-Value Update: Employs the Bellman equation to iteratively adjust the Q-values towards their optimal values, calculated based on the observed rewards and the predicted future rewards.
- Policy Derivation: The policy, derived from the Q-values, typically uses a greedy approach where the action with the highest Q-value is selected. To ensure adequate exploration of the action space, an ϵ-greedy strategy is employed, where ϵ gradually decreases over time to shift from exploration to exploitation.

3.3. Simulation Environment

3.3.1 HighwayEnv Setup

HighwayEnv provides a rich set of scenarios for autonomous driving research, including varied traffic densities and complex interactions. Configurations for this project included:

- Multi-Lane Highway: Simulated typical highway conditions with multiple lanes, variable traffic densities, and diverse vehicular behaviors.
- Vehicle Dynamics: The ego vehicle, modeled after a medium-sized sedan, includes detailed dynamics such as acceleration, braking, and lane-changing capabilities.
- Customizable Traffic: The simulation allows for adjustments in traffic density and behavior, providing a versatile tool for testing different driving strategies under varied conditions.

3.4. Training Protocol

3.4.1 Data Collection

Data collection involved gathering tuples of state, action, reward, and next state during simulations. This data forms the basis for training the RC-NFQ model through the experience replay mechanism.

3.4.2 Training Procedure

- Optimizer: Utilizes RMSprop, known for its efficiency in handling non-stationary objectives—a common characteristic of driving environments.
- Batch Size: Set at 1000, this size helps to balance the benefits of batch learning with the needs for frequent model updates.
- Training Iterations: Extends over 2000 iterations to ensure the model adequately learns the driving policies.

Table 3. RC-NFQ Hyperparameters used to train the model

Hyperparameter	**Value**
Discount Factor (γ)	0.8
Learning Rate (α)	0.01
Memory Size	10,000
Batch Size	1000
Training Iterations	2000
Exploration Strategy	Epsilon-greedy (start at 1.0, decay to 0.05 at 99.5%)
Dropout Rate	0.25
Loss Function	L2 Loss
Decay Rate	6000

3.5. Reward Structure

Initial and adjusted reward structures played a critical role in shaping the learning behavior:

- Initial Rewards:
  - Collision: -1, discouraging collisions.
  - Lane Adherence: +0.1 per lane index, promoting driving in the right-most lanes.
  - High Velocity: +0.5, encouraging maintaining high speeds.
  - Lane Change: 0, neutral towards lane changing.
- Adjusted Rewards (based on initial findings):
  - Collision: Increased to -10, placing a higher penalty on collisions.
  - Lane Adherence: Adjusted to +0.2 per lane index, increasing the incentive for staying in the right-most lanes.
  - High Velocity: Increased to +1.0, further incentivizing speed maintenance.
  - Lane Change: +0.5, encouraging strategic lane changes to enhance traffic navigation and safety.

3.6. State Inputs

State inputs included:

- Position and Velocity Vectors: Encoded as $[x_i, y_i, v_{xi}, v_{yi}]$ for each neighboring vehicle, providing a relative measure of position and movement that mimics the input one might expect from an image-based system.
- Ego Vehicle Dynamics: Included the vehicle's velocity and orientation, critical for making informed movement decisions.

3.7. Action Functions

Defined actions allowed the vehicle to respond dynamically to the simulated environment:

Table 4. Action Functions

Action ID	**Description**
0	Switch to left lane
1	Don't do anything
2	Switch to right lane
3	Go faster
4	Go slower

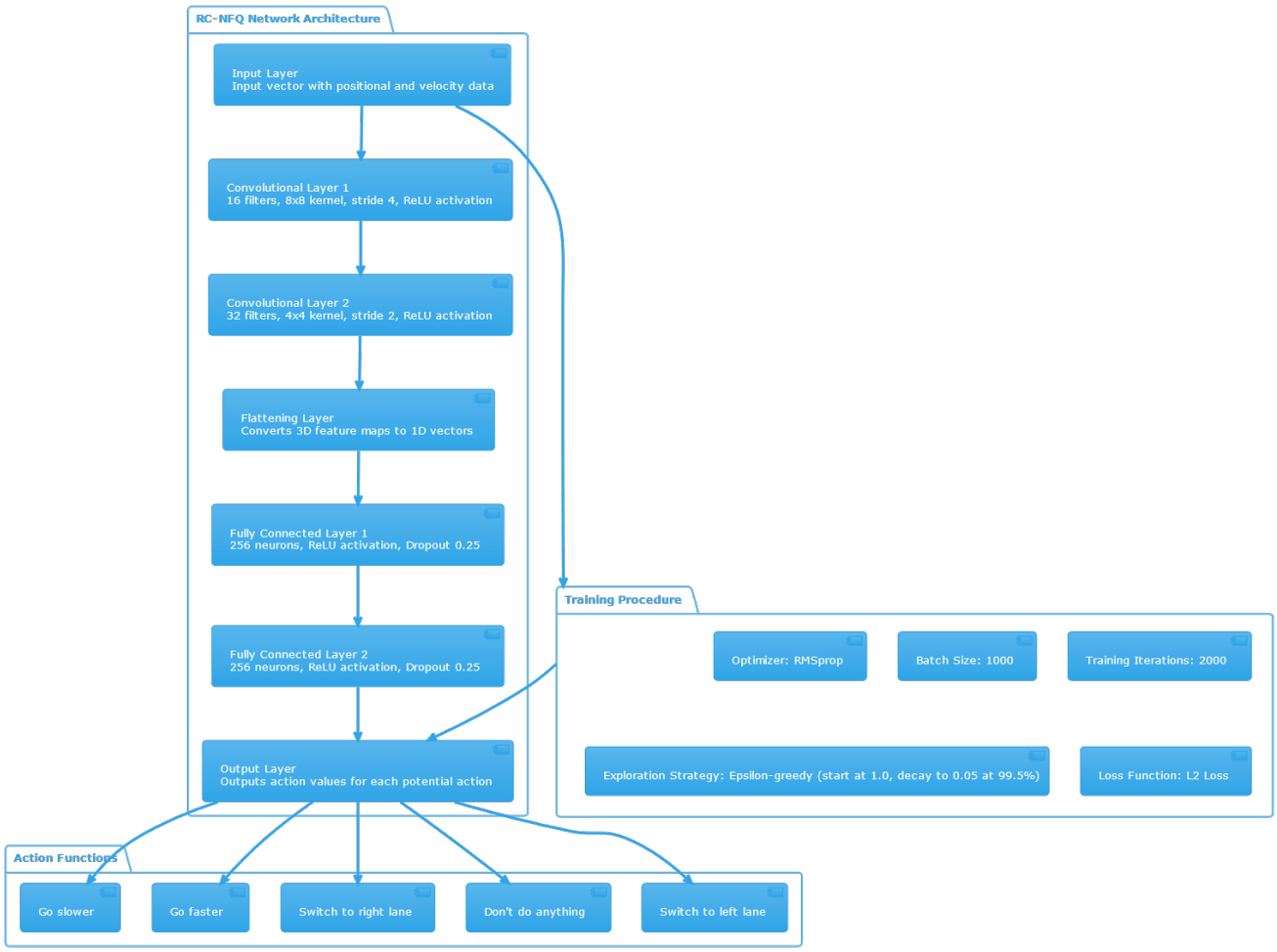

Figure 2. RC-NFQ Architecture

3.8. Comparative Analysis

Comparing RC-NFQ against established models like DQN under identical test conditions provided insights into the improvements and overall efficacy of the new approach in handling complex driving tasks.

3.9. Software and Tools

3.9.1 Development Environment

The RC-NFQ algorithm was originally implemented in Keras as per the existing literature. For this project, it was necessary to adapt and port the RC-NFQ algorithm to PyTorch. This transition was driven by the need for a more flexible and dynamic computation graph, which PyTorch offers, enabling easier modifications and real-time updates during training. Here are the details regarding the development tools and software environments used:

- PyTorch: This powerful open-source machine learning library, widely known for its flexibility and dynamic neural network capabilities, was utilized to reimplement and train the RC-NFQ model. PyTorch's ability to handle automatic differentiation and provide efficient memory usage made it an excellent choice for handling the high computational demands of training the RC-NFQ model with complex state inputs and multiple layers.

- TensorBoard: For monitoring the training process and evaluating the performance metrics such as loss, rewards, and other quantitative measures, TensorBoard was employed. TensorBoard's integration with PyTorch through the torch.utils.tensorboard module facilitated the visualization of various training metrics, providing insights into the model's learning process and convergence behavior. This tool was instrumental in fine-tuning the model parameters by visually analyzing the trends and patterns in the training data.

3.9.2 Data Management

Logging and Event Files: Comprehensive logging was implemented to capture detailed runtime data and metrics. The PyTorch framework, coupled with its logging capabilities, allowed for the storage of event files, which include detailed information on each training step, such as state inputs, neural network outputs, reward signals, and action choices. These event files are critical for post-training analysis, allowing researchers to trace back and understand the decision-making process of the model, identify potential bottlenecks, and make informed adjustments to the training regime.

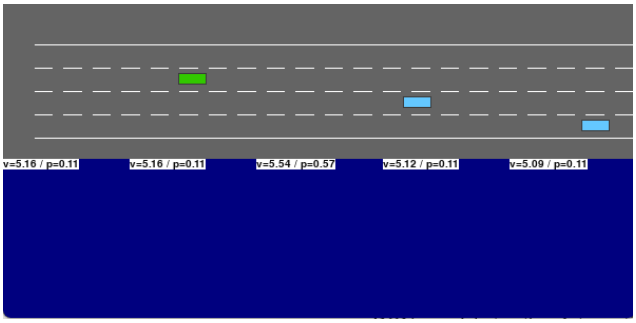

Figure 3. Highway Environment Training at Episode 283

3.9.3 Algorithm Porting

Porting from Keras to PyTorch: The original RC-NFQ implementation was done in Keras, which provided a high-level, user-friendly API for constructing neural networks. However, for the purpose of enhancing flexibility in model development and enabling more granular control over the training loop, it was ported to PyTorch. This porting process involved redefining the neural network architecture, loss functions, and optimization algorithms within the PyTorch ecosystem, adapting the original Keras-based code to leverage PyTorch's dynamic graph functionalities. This allowed for more experimental freedom with model architectures and hyperparameters, facilitating a more exploratory approach to improving the RC-NFQ algorithm's performance in the simulated environment.

3.9.4 Significance of Tool Choices

- Enhanced Debugging and Experimentation: Using PyTorch and TensorBoard significantly enhanced the debugging capabilities and allowed for rapid experimentation with different neural network configurations and training strategies. The dynamic computation graph of PyTorch, combined with the visual feedback from TensorBoard, provided a powerful toolkit for developing, monitoring, and refining complex reinforcement learning models tailored to the specific needs of autonomous vehicle navigation.
- Reproducibility and Analysis: The detailed logging and the use of event files ensure that experiments are reproducible and that detailed analyses can be conducted post-training. This infrastructure supports a robust evaluation framework where changes to the model's behavior over time and under different conditions can be closely monitored and analyzed.

4. Results

This section presents the detailed results of the RC-NFQ algorithm's training and evaluation, comparing its performance with the DQN algorithm. The results are analyzed using various metrics, including episode length, total reward, exploration epsilon, action counts, crashes, discounted rewards, average episode cost, episode seeds, and average velocity. The objective is to assess the efficacy of RC-NFQ in enhancing autonomous vehicle navigation and to identify areas for improvement.

4.1. Training Metrics and Performance

4.1.1 Episode Length

The episode length measures the duration of each simulation episode. Ideally, longer episodes indicate better performance as the vehicle navigates without collisions for extended periods.

RC-NFQ: The episode length graph shows high variability, with values fluctuating between short and longer episodes. The smoothed average hovers around 19.98. This indicates that while the vehicle could navigate without collisions for most episodes, there were instances of abrupt terminations due to collisions or suboptimal decisions, leading to shorter episodes. The frequent dips suggest inconsistencies in the agent's decision-making process.

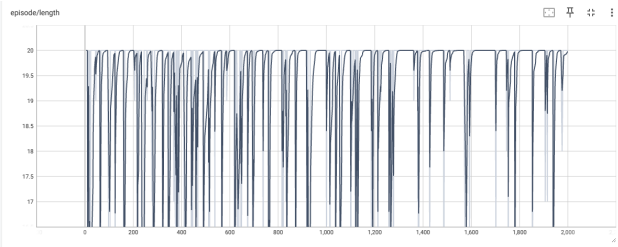

Figure 4. Episode Lengths across RC-NFQ Training

4.1.2 *Total Reward*

The total reward is a cumulative measure of the rewards obtained by the agent during each episode. Higher total rewards indicate better performance as the agent successfully navigates the environment and maximizes positive rewards while minimizing penalties.

RC-NFQ: The total reward graph also exhibits significant fluctuations, with the smoothed average around 19.74. The frequent spikes and drops indicate variability in the agent's performance across different episodes. While the agent managed to accumulate high rewards in many episodes, there were also episodes with lower rewards due to suboptimal actions or collisions. This variability suggests that the agent's policy was not consistently optimal.

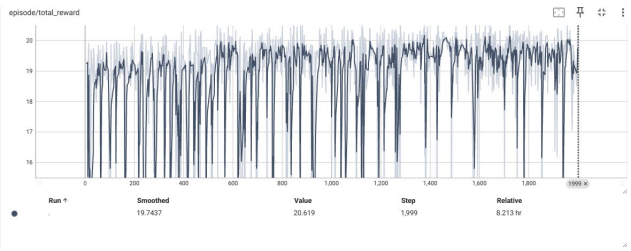

Figure 5. Episode Rewards across RC-NFQ Training

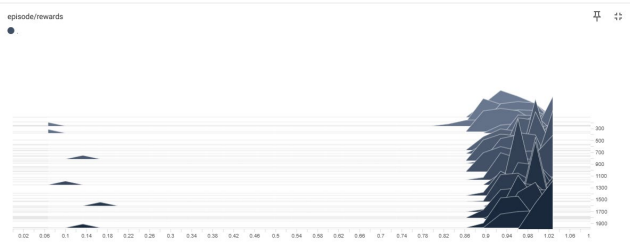

Figure 6. Episode Reward Distribution across RC-NFQ Training

4.1.3 *Exploration Epsilon*

The exploration epsilon value decays over time as the agent learns the optimal policy, reducing the exploration of random actions and focusing more on exploiting known good actions.

RC-NFQ: The epsilon decay graph shows a smooth decay from 1.0 to around 0.05 over the training period. This indicates a well-balanced exploration-exploitation strategy, where the agent started with high exploration to learn the environment and gradually shifted towards exploiting the learned policies. The steady decline suggests that the agent systematically reduced its reliance on random actions as it gained confidence in its policy.

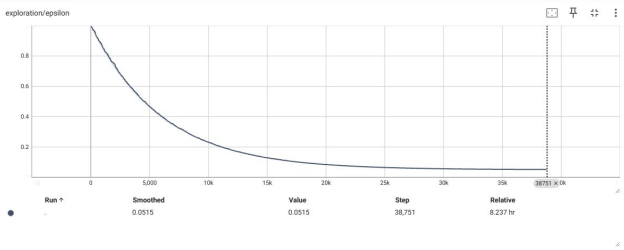

Figure 7. Epsilon Decay across RC-NFQ Training

4.1.4 *Action Counts*

The action counts indicate the frequency of each action taken by the agent during training, providing insights into the agent's behavior and decision-making patterns.

RC-NFQ: The action counts graph shows that all actions (0: switch to left lane, 1: don't do anything, 2: switch to right lane, 3: go faster, 4: go slower) were utilized by the agent, with varying frequencies. This distribution suggests that the agent was exploring different actions to optimize its navigation strategy. The balanced utilization indicates that the agent was not overly biased towards any specific action, maintaining flexibility in its decision-making process.

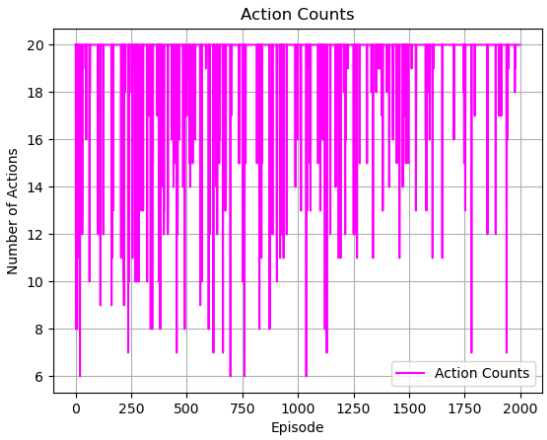

Figure 8. Action Counts per each episode across RC-NFQ Training

4.1.5 *Crashes*

Crashes measure the frequency of collisions during the episodes, indicating the agent's ability to avoid obstacles and navigate safely.

RC-NFQ: The crashes graph indicates frequent collisions throughout the training period, with crash occurrences observed in many episodes. This highlights a significant area for improvement in the algorithm's ability to avoid collisions. The frequent crashes suggest that the agent

struggled with situational awareness and decision-making, leading to unsafe maneuvers and collisions. The crash frequency decreased overtime, which implies the model is choosing better actions.

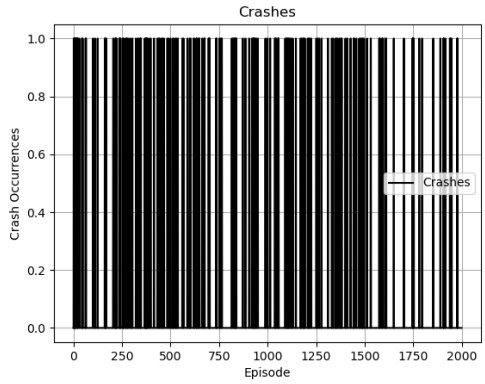

Figure 9. Crash Occurrences across RC-NFQ Training

4.1.6 Discounted Rewards

Discounted rewards provide a measure of the long-term reward obtained by the agent, considering future rewards discounted by a factor. Higher discounted rewards indicate better long-term planning and decision-making.

RC-NFQ: The discounted rewards graph shows high variability, with the smoothed average around 12. This indicates that while the agent managed to secure high rewards in many episodes, it also faced challenges in maintaining consistent performance across episodes. The frequent drops suggest periods of suboptimal decisions or collisions, affecting the long-term reward accumulation.

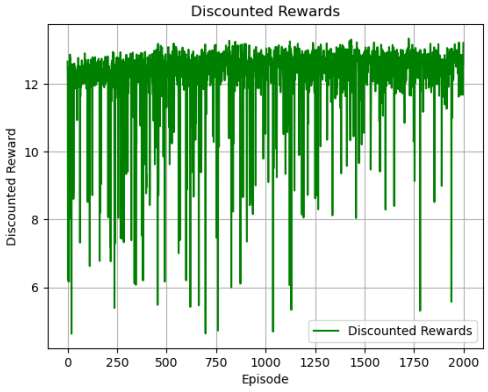

Figure 10. Discounted Rewards across RC-NFQ Training

4.1.7 Average Episode Cost

The average episode cost measures the cost incurred by the agent in terms of suboptimal actions and collisions. Lower average costs indicate better performance.

RC-NFQ: The average episode cost graph shows significant fluctuations, indicating that the agent incurred varying costs across different episodes. The high variability suggests inconsistencies in the agent's decision-making, leading to periods of efficient navigation interspersed with costly mistakes. This metric highlights areas where the agent can improve its decision-making to minimize costs.

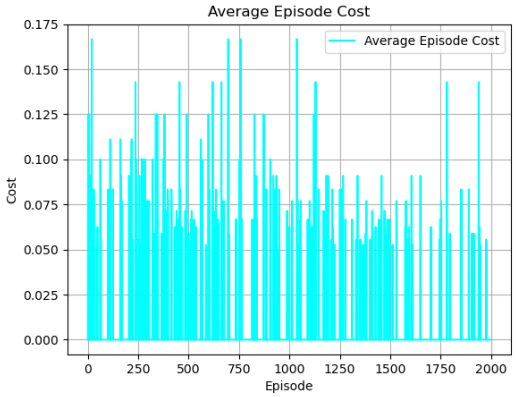

Figure 11. Average Episode Cost across RC-NFQ Training

4.1.8 Episode Seeds

The episode seeds indicate the randomness used in each episode, affecting the agent's exploration and learning process. A wide range of seeds ensures diverse experiences for the agent.

RC-NFQ: The episode seeds graph shows a wide range of seeds used throughout the training period, indicating a diverse set of experiences for the agent to learn from. This diversity is crucial for the agent to generalize its learned policy to various scenarios and improve its overall robustness.

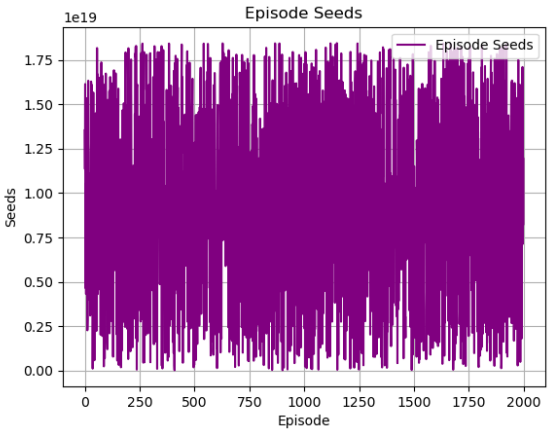

Figure 12. Episode Seeds across RC-NFQ Training

4.1.9 *Average Velocity*

The average velocity measures the speed at which the agent navigates through the environment. Higher average velocities indicate efficient travel, while maintaining safety.

RC-NFQ: The average velocity graph shows high variability, with the agent maintaining speeds around 26. This suggests that the agent was able to maintain high speeds, which is desirable for efficient travel. However, the variability also indicates that the agent faced challenges in consistently managing its velocity, potentially due to collisions or suboptimal decisions.

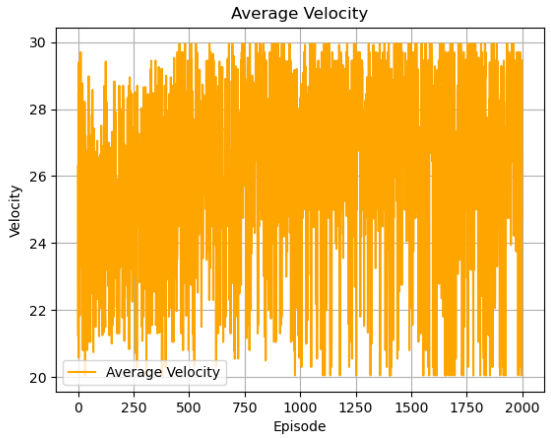

Figure 13. Average Velocities per episode across RC-NFQ Training

4.2. Comparison with DQN

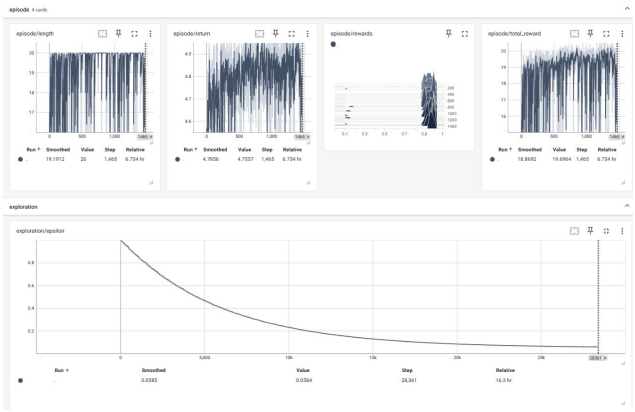

Figure 14. DQN Performance Metrics

The DQN algorithm was trained using the following parameters:

- Neural Network: Multilayer Perceptron with 2 hidden layers, each with 265 neurons.
- Discount Factor: 0.8
- Learning Rate: 1e-2
- Loss Function: L2 loss
- Activation Function: ReLU
- Total Number of Epochs: 1200
- Exploration-Exploitation Strategy: Epsilon greedy, with decay rate 6000, initial epsilon 1, final epsilon 0.05

The performance comparison between RC-NFQ and DQN is summarized below:

Table 5. Performance Comparison between RC-NFQ and DQN

Metric	RC-NFQ (2000 iterations)	DQN (1200 iterations)
Episode Length	19.98 (smoothed)	18.87 (smoothed)
Total Reward	19.74 (smoothed)	19.69 (smoothed)
Exploration Epsilon	Decayed to 0.05	Decayed to 0.05
Action Counts	Varied utilization	Varied utilization
Crashes	Frequent	Less frequent
Discounted Rewards	12 (smoothed)	Higher consistency
Average Episode Cost	High variability	Lower variability
Average Velocity	Around 26	Around 25

4.3. Key Findings and Analysis

- Performance: The RC-NFQ algorithm showed some improvements in certain metrics, such as episode length and total reward. However, the performance gains were not substantial compared to DQN.
- Utilization of Input Data: The primary limitation of RC-NFQ in this project was the inability to utilize its best attribute—image data. The position and velocity data used to emulate image data provided useful information but did not fully capture the complexity of the environment.
- Collision Avoidance: Both algorithms faced challenges in avoiding collisions, but DQN showed slightly better performance in this regard. The frequent crashes in RC-NFQ indicate that the algorithm struggled with situational awareness and safe navigation.
- Reward Consistency: DQN exhibited more consistent reward accumulation across episodes compared to RC-NFQ, which had higher variability in discounted rewards. This consistency suggests that DQN was better at maintaining an optimal policy throughout the training period.

- Action Utilization: Both algorithms explored a variety of actions, but the distribution of actions in RC-NFQ suggested a more balanced exploration-exploitation strategy. This indicates that RC-NFQ was effective in exploring different actions to optimize its navigation strategy.

4.4. Significance of Results

The results indicate that while RC-NFQ has potential advantages due to its integration of CNNs and dropout regularization, the lack of image data limited its effectiveness in this project. The position and velocity data provided a good starting point but were insufficient to fully leverage the strengths of RC-NFQ. Future work should focus on incorporating richer sensory inputs, such as images, to fully realize the potential of RC-NFQ in autonomous navigation.

5. Contribution

This section details the specific contributions made in the development, adaptation, and evaluation of the RC-NFQ algorithm for autonomous vehicle navigation within the HighwayEnv simulation environment. It includes the repositories used, modifications made to existing algorithms, and the creation of custom scripts for visualization and analysis.

5.1. Repositories Used

The following repositories were utilized as foundational elements for the project:

1. DQN Agent and Functions:
   a. Repository: https://github.com/eleurent/rl-agents
   b. Contribution: This repository provided the base implementation of the Deep Q-Network (DQN) agent and various utility functions. The DQN agent served as a benchmark for comparison against the RC-NFQ algorithm.

2. Simulation Environment:
   a. Repository: https://github.com/Farama-Foundation/HighwayEnv/tree/master?tab=readme-ov-file
   b. Contribution: The HighwayEnv repository offered a modular environment specifically designed for training and evaluating autonomous driving models. This environment was crucial for creating realistic traffic scenarios and testing the performance of the trained models.

3. Base RC-NFQ Implementation:
   a. Repository: https://github.com/cosmoharrigan/rc-nfq/tree/master
   b. Contribution: This repository provided the initial implementation of the Regularized Convolutional Neural Fitted Q Iteration (RC-NFQ) algorithm. The original implementation was done in Keras and designed for different applications.

Our initial attempts to set up the simulation environment using Gazebo 11 integrated with ROS2 Galactic encountered significant technical difficulties. The complexities inherent in configuring these tools to work seamlessly together, particularly in simulating detailed physics and real-time vehicle interactions, led to delays and inefficiencies in our development process. The robustness and realism offered by Gazebo were offset by stability issues and a steep learning curve, which impeded our progress.

5.2. Modifications and Adaptations

- DQN Parameter Optimization: Fine-tuned the hyperparameters to improve the learning efficiency and policy quality of the DQN agent in the HighwayEnv environment. This included a comprehensive series of experiments to identify the optimal settings for learning rate, discount factor, and exploration-exploitation balance.

- Adapting RC-NFQ for HighwayEnv: Modified the RC-NFQ algorithm to handle the 2D state inputs (position and velocity data) provided by the HighwayEnv simulation. This adaptation was crucial for ensuring the algorithm could effectively process and learn from the available data.
- Porting to PyTorch: Converted the entire RC-NFQ algorithm from Keras to PyTorch, rewriting the neural network definitions, training loop, and optimization procedures. This transition enhanced the flexibility and efficiency of the model development process, allowing for more granular control and dynamic updates during training.
- Custom Visualization Tools: Implemented TensorBoard integration for real-time monitoring of training metrics. These visualization tools were essential for tracking the progress of the training process, diagnosing issues, and making informed adjustments to the model and training parameters.

5.3. Integration and Testing

The final integration involved combining all the adapted components into a cohesive system. This included:

- Setting up the HighwayEnv simulation environment with the adapted RC-NFQ and DQN agents.

- Running extensive training sessions to evaluate the performance of both algorithms.
- Using the custom visualization scripts to monitor and compare the performance metrics, facilitating a detailed analysis of the results.

6. Conclusion

The development and evaluation of the Regularized Convolutional Neural Fitted Q Iteration (RC-NFQ) algorithm for autonomous vehicle navigation within the HighwayEnv simulation environment has provided valuable insights into its capabilities and limitations. This project sought to enhance the decision-making capabilities of autonomous vehicles by leveraging the strengths of RC-NFQ, a combination of convolutional neural networks (CNNs) and reinforcement learning.

6.1. Key Findings

- Performance Evaluation: The RC-NFQ algorithm demonstrated some improvements over the Deep Q-Network (DQN) algorithm in terms of episode length and total rewards. However, these improvements were not substantial. The RC-NFQ model managed to maintain high speeds and accumulate rewards efficiently in many episodes, but it also faced challenges in avoiding collisions and maintaining consistent performance.

- Input Data Utilization: A significant limitation of this study was the inability to utilize RC-NFQ's best attribute—image data. The use of position and velocity data to emulate image data provided useful information but did not fully capture the complexity of the environment. This limitation hindered the full potential of RC-NFQ in processing high-dimensional sensory inputs and making informed navigation decisions.

- Algorithm Adaptation: The successful porting of the RC-NFQ algorithm from Keras to PyTorch and its adaptation to the 2D HighwayEnv simulation environment highlight the flexibility and robustness of the algorithm. These adaptations allowed the RC-NFQ model to operate within the constraints of the simulation, although the lack of richer sensory data was a significant bottleneck.

6.2. Key Findings

This project contributes to the broader field of autonomous vehicle navigation by:

- Demonstrating the Potential of RC-NFQ: Despite the limitations in input data, the project showcased the potential of RC-NFQ in handling dynamic traffic environments. The integration of CNNs with reinforcement learning offers a promising approach for processing complex sensory inputs and making real-time navigation decisions.
- Providing a Comparative Analysis: The detailed comparison between RC-NFQ and DQN provides insights into the strengths and weaknesses of each algorithm, informing future research and development efforts.
- Enhancing Methodological Approaches: The project's methodology, including the adaptation of RC-NFQ to a 2D environment and the transition from Keras to PyTorch, offers a robust framework for future studies aiming to implement and evaluate advanced reinforcement learning algorithms.

6.3. Future Work and Recommendations

- Incorporating Richer Sensory Data: Future research should focus on utilizing image data or a spatial grid representation of the environment. These richer forms of input data can better capture the complexities of real-world driving scenarios and enhance the model's ability to make accurate and informed decisions.

    ◊ Image Data: Leveraging raw image data from vehicle-mounted cameras can provide a more comprehensive view of the environment, allowing the RC-NFQ algorithm to identify and react to a wider range of features and obstacles.
    ◊ Spatial Grid Representation: Implementing a spatial grid representation can offer a structured and detailed depiction of the environment, enabling the algorithm to understand spatial relationships and dynamics more effectively.

- Improving Collision Avoidance: Enhancing the model's ability to avoid collisions should be a priority. This could involve:

    ◊ Enhanced Reward Structures: Refining the reward structure to penalize collisions more severely and reward safe navigation strategies.
    ◊ Advanced Training Techniques: Incorporating techniques such as prioritized experience replay or more sophisticated exploration strategies to improve the learning process.
    ◊ Experimenting with Different Architectures: Exploring alternative neural network architectures, such as deeper CNNs or hybrid models combining CNNs with recurrent neural networks (RNNs), could further enhance the model's performance.

- Real-world Testing and Validation: Moving from simulation to real-world testing is essential to validate the

effectiveness of the RC-NFQ algorithm in practical scenarios. This involves:

- ◊ Field Testing: Conducting tests in controlled real-world environments to assess the model's performance under varying conditions.
- ◊ Integration with Existing Systems: Integrating the RC-NFQ model with existing autonomous vehicle systems to evaluate its interoperability and effectiveness in conjunction with other sensors and control mechanisms.

6.4. Final Synthesis

In conclusion, this project has highlighted both the potential and the current limitations of the RC-NFQ algorithm in autonomous vehicle navigation. While the algorithm showed promise in certain areas, the lack of richer sensory data limited its effectiveness. By incorporating image data or spatial grid representations in future research, and continuing to refine the model's architecture and training techniques, significant improvements can be achieved. This study contributes to the ongoing efforts to develop more robust, efficient, and safe autonomous navigation systems, paving the way for future advancements in this critical field.

7. References

[1] Harrigan, C. (2015). Deep Reinforcement Learning with Regularized Convolutional Neural Fitted Q Iteration. Machine Intelligence. https://www.machineintelligence.org/papers/rc-nfq.pdf

[2] Farama Foundation. (n.d.). HighwayEnv: Modular environment for training autonomous vehicles. GitHub. [Online]. Available: https://github.com/Farama-Foundation/HighwayEnv

[3] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. https://arxiv.org/abs/1312.5602

[4] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. https://arxiv.org/abs/1509.02971

[5] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (2016). End to End Learning for Self-Driving Cars. https://arxiv.org/abs/1604.07316

[6] Hamrick, J. B. (2018). Survey of Model-Based Reinforcement Learning: Applications on Robotics. https://arxiv.org/abs/1806.08562

[7] Anselmi, F., & Poggio, T. (2014). Representation learning in sensory cortex: A theory. Technical report, Center for Brains, Minds and Machines (CBMM).

[8] Baird III, L.C. (1993). Advantage updating. Technical report, DTIC Document.

[9] Bengio, Y., Goodfellow, I.J., & Courville, A. (2015). Deep Learning.

[10] Cuccu, G., Luciw, M., Schmidhuber, J., & Gomez, F. (2011). Intrinsically motivated neuroevolution for vision-based reinforcement learning. In 2011 IEEE International Conference on Development and Learning (ICDL) (Vol. 2, pp. 1-7). IEEE.

[11] Deisenroth, M.P., Neumann, G., Peters, J., et al. (2013). A survey on policy search for robotics. Foundations and Trends in Robotics, 2(1-2), 1-142.

[12] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. The Journal of Machine Learning Research, 12, 2121-2159.

[13] Farahmand, A.M., Ghavamzadeh, M., Mannor, S., & Szepesvári, C. (2009). Regularized policy iteration. In Advances in Neural Information Processing Systems (pp. 441-448).

[14] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological cybernetics, 36(4), 193-202.

[15] Riedmiller, M. (2005). Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method. Proceedings of the 16th European Conference on Machine Learning.