Introduction to Machine learning and Data mining

Self-Driving Cars

David Chonev and Nenad Jakovchevski

FAMNIT, University of Primorska

June 22, 2024

1 Introduction

This project focuses on developing an automated car navigation system using TypeScript. The goal is to enable various car models to autonomously learn and execute driving tasks with precision and reliability. By harnessing advanced machine learning techniques, the system aims to enhance road safety by minimizing human error and adapting dynamically to highway conditions.

The key approach was to decouple the sensor data acquisition (Sensor) from the decision-making process (Network) and the car control mechanisms (Controls).

Using this abstraction, we demonstrate how to solve the problem of car navigation in various scenarios, such as obstacle avoidance and lane following. We implement this system using several components:

1. *Car* - component manages the vehicle's position, speed, and direction.
2. *Control* - which is a procedure that manage the car's acceleration and turning based on commands.
3. *Sensor* - collects data about the car's environment.
4. *Road* - represents the road's properties.
5. *Network* - acts as the central control unit processing sensor data and controlling the car.

2 Project Overview

The system begins with the ***main*** function, which initializes the key components.
The Following is performed: The ***Car*** module manages the vehicle's position, speed, and direction, while the ***Road*** module defines the road's properties, such as the number of lanes. The ***Sensor*** module continuously collects data about the car's surroundings, specifically the distance to obstacles, using the *update* utility function. This sensor data is then fed into the ***Network*** module, which acts as the central control unit. The ***Network*** processes the sensor data at regular intervals and makes real-time decisions to control the car's movement. Depending on the proximity of obstacles, the ***Network*** adjusts the car's speed and direction by sending commands via a use of *mutation* corresponding with the Genetic Algorithm Principle, to the ***Controls*** module, which in turn applies the described turning to the Car.
This sequence ensures the car navigates safely, avoiding obstacles and adjusting its path as needed. The modular design allows each component to focus on a specific aspect of navigation.

3 Materials & Methods

In the development of our self-driving car project, we utilized a variety of resources, primarily focusing on online tutorials and frameworks that facilitated our understanding and implementation of machine learning algorithms. The following are the key materials and methods used:

3.1 Educational Resources

We extensively referred to educational videos on YouTube, which provided us with foundational knowledge and practical insights into AI development and neural networks. Notable mentions include:

- *Deep Learning Cars* by Samuel Arzt [1]
- *Self-Driving AI Car Simulation in Python* by NeuralNine [3]
- *Deep Learning Car Simulator* by Moataz Elmasry [7]

3.2 Algorithms and Frameworks

We experimented with several machine learning algorithms to optimize the performance of our self-driving car system:

- **Reinforcement Learning**: This approach was used to enable the car to learn from its environment through trial and error, rewarding successful navigation and penalizing failures.

- **Q-learning**: We explored Q-learning as a method to create a policy for the car to decide the best action based on its current state.
- **TensorFlow.js**: We attempted to use the TensorFlow framework in JavaScript to solve local optima issues in the neural network training process.

3.3 Optimization Strategy

Through our experiments, we determined that saving and restarting the generation with the best-performing car was the most effective strategy for optimizing the neural network. This approach allowed us to overcome local optima and continuously improve the car's performance in various driving scenarios.

The integration of these materials and methods enabled us to create a robust and adaptive self-driving car system.

4 Results

In the development and testing of our self-driving car system, we observed several key results that informed our final implementation strategy. Our iterative approach and experiments with different techniques yielded the following insights:

4.1 Generation Save and Reset Method

Initially, we experimented with various machine learning algorithms and frameworks. However, we found that the most effective strategy was to save the best-performing car from each generation and use it as a starting point for the next generation. This save and reset method allowed us to avoid local optima and continuously improve the neural network's performance. By retaining the best car and restarting with it, the AI was able to maintain and build upon successful behaviors.

4.2 Optimal Mutation Rate

Through extensive testing, we determined that a mutation rate of 20% was most effective for our genetic algorithm. This mutation rate provided a balance between exploration and exploitation, ensuring that the AI could adapt to new challenges while retaining learned behaviors. Higher or lower mutation rates resulted in either too much randomness or insufficient adaptability.

4.3 Genetic Algorithm Performance

The genetic algorithm played a crucial role in training the AI. It used a population of cars, each represented by a neural network, and evolved them over successive generations. The algorithm selected the best-performing cars based on their ability to navigate the road and avoid obstacles, and used crossover and mutation to create new generations. This process allowed the AI to learn and adapt continuously.

4.4 Training and Testing with Random Traffic Generation

Our dataset initially trained the AI to recognize and navigate static environments. However, to ensure robustness and adaptability, we introduced random traffic generation in further experiments. This approach tested the trained AI's ability to handle dynamic and unpredictable scenarios, significantly enhancing its performance in real-world situations.

4.5 Impact of Generations on Performance

We observed that the AI's performance improved with an increasing number of generations. The longer the genetic algorithm ran, the more refined the AI's behavior became. Early generations exhibited basic navigation skills, while later generations demonstrated advanced obstacle avoidance and lane-following capabilities. The results showed that more generations led to more optimal solutions and better overall performance.

The culmination of these methods and findings resulted in a highly adaptive and efficient self-driving car system, capable of navigating complex environments with precision.

5 Look-Back and Optimizations

The results of our project indicate significant progress in the development of a self-driving car system using machine learning techniques. The iterative approach of saving and restarting with the best-performing car from each generation proved to be a critical factor in overcoming local optima and enhancing the neural network's performance.

5.1 Interpretation of Results

Our findings show that the save and reset generation method allowed the AI to retain successful behaviors while continually exploring new strategies. The optimal mutation rate of 20% balanced exploration and exploitation, providing sufficient variability to adapt to new scenarios without losing previously learned skills. This approach was particularly effective when the AI was exposed to random traffic generation, demonstrating its ability to handle dynamic and unpredictable environments.

The use of a genetic algorithm enabled the AI to evolve and improve over successive generations. Early generations displayed basic navigation skills, but as the number of generations increased, the AI's capabilities became more sophisticated. This trend supports the hypothesis that genetic algorithms can ef-

fectively train neural networks for complex tasks like autonomous driving.

5.2 Justification of Methodology

The decision to use a genetic algorithm was driven by its suitability for optimization problems where the solution space is large and complex. Unlike traditional gradient-based optimization methods, genetic algorithms do not require gradient information and can navigate rugged fitness landscapes effectively. This characteristic made it ideal for our application, where the environment is highly variable and non-linear.

The implementation of the TensorFlow.js framework allowed us to leverage powerful machine learning tools directly in JavaScript, facilitating seamless integration with our TypeScript-based system. This choice was justified by the need for efficient real-time processing and the convenience of using a widely-supported framework.

5.3 Potential Sources of Error

While our approach yielded promising results, several potential sources of error could have impacted the outcomes:

- **Sensor Accuracy**: The precision of the sensor data directly affects the AI's decision-making. Any inaccuracies or delays in sensor readings could lead to suboptimal control actions.
- **Random Traffic Generation**: Although random traffic generation tested the AI's adaptability, it also introduced variability that could result in inconsistent performance across different runs.
- **Genetic Algorithm Parameters**: The choice of parameters for the genetic algorithm, such as population size and crossover rate, could significantly influence the training process. Fine-tuning these parameters was crucial, but not exhaustive, potentially leaving room for further optimization.

6 Conclusion

In conclusion, our project demonstrates the viability of using machine learning techniques, particularly genetic algorithms, for developing self-driving car systems. The combination of saving the best-performing car, an optimal mutation rate, and random traffic generation contributed to a robust and adaptable AI. Future work could focus on refining the genetic algorithm parameters, improving sensor accuracy, and exploring additional machine learning frameworks to further enhance the system's performance.

References

[1] Samuel Arzt, *Deep Learning Cars*, https://www.youtube.com/watch?v=Aut32pR5PQA&,

[2] Samuel Arzt, *AI learns to park*, https://www.youtube.com/watch?v=VMp6pq6_QjI&,

[3] NeuralNine, *Self-Driving AI Car Simulation in Python*, https://www.youtube.com/watch?v=Cy155O5R1Oo,

[4] Steve Brunton, *Deep Reinforcement Learning: Neural Networks*, https://www.youtube.com/watch?v=IUiKAD6cuTA,

[5] Code Monkey, *AI Learns to Drive a Car! (ML-Agents in Unity)*, https://www.youtube.com/watch?v=2X5m_nDBvS4&,

[6] Anshul Saxena, *Machine Learning Algorithms in Autonomous Cars*, https://www.visteon.com/machine-learning-algorithms-in-autonomous-cars/,

[7] Moataz Elmasry, *Deep Learning Car Simulator*, https://towardsdatascience.com/deep-learning-on-car-simulator-ff5d105744aa,