Artificial Intelligence Lab Project Synopsis

Training a Self-Driving Car using NEAT (NeuroEvolution of Augmenting Topologies) Algorithm

Jaypee Institute of Information Technology

Submitted by

Ananya Kapoor	**20103104**	**B4**
Dhairya Sachdeva	**20103098**	**B4**
Harshita	**20103331**	

Abstract

This project aims to train a simulated self-driving car to navigate through custom tracks designed in Pygame using the NEAT (NeuroEvolution of Augmenting Topologies) algorithm. The NEAT algorithm is a type of genetic algorithm that evolves neural networks to perform a specific task, in this case, driving the car through the track. The car is equipped with various sensors to detect the environment and make decisions about its movements. The training process involves evaluating the performance of the car on each track and using the NEAT algorithm to evolve the neural network until optimal performance is achieved. The project provides a platform for exploring the capabilities of NEAT and developing self-driving car technologies in a simulated environment. Additionally, the use of Pygame allows for the creation of customized tracks, which can be used to test the car's abilities in different scenarios.

Scope of the Project

The scope of the project is to develop a self-driving car simulation using the NEAT algorithm, which will enable the car to navigate through custom tracks created in Pygame. The project includes the following points:

1. Understanding the NEAT algorithm: a. Study the principles and working of the NEAT algorithm, which is a genetic algorithm that evolves neural networks to perform a task. b. Understand how the NEAT algorithm can be used to develop self-driving cars.
2. Developing the simulation environment: a. Create a simulation environment using Pygame that includes a car and a custom track. b. Incorporate sensors into the car to detect the environment and make decisions about its movements. c. Implement a fitness function to evaluate the car's performance.
3. Training the self-driving car: a. Train the car using the NEAT algorithm to navigate through the track. b. Evaluate the car's performance on each track and use the NEAT algorithm to evolve the neural network until optimal performance is achieved. c. Analyze the performance of the car and the neural network.
4. Customizing the tracks: a. Develop multiple custom tracks to test the car's abilities in different scenarios. b. Explore the impact of different track configurations on the car's performance.

5. Further development: a. Explore the possibility of incorporating other machine learning algorithms to enhance the performance of the car. b. Investigate the potential of applying the simulation to real-world self-driving car technology.

In conclusion, this project aims to develop a self-driving car simulation using the NEAT algorithm to navigate through custom tracks created in Pygame. The project has significant scope for exploring the capabilities of NEAT and developing self-driving car technologies in a simulated environment. The use of Pygame allows for the creation of customized tracks to test the car's abilities in different scenarios, and further development could lead to real-world applications.

Tools and Technologies Used

The following tools and technologies were used in this project:

1. Pygame: Pygame is a Python library designed for game development, and it was used to create the simulation environment for the self-driving car.
2. NEAT-Python: NEAT-Python is an open-source Python implementation of the NEAT algorithm, and it was used to evolve the neural network for the self-driving car.
3. Python: Python is a popular high-level programming language, and it was used to write the code for the simulation environment and the NEAT algorithm.
4. Jupyter Notebook: Jupyter Notebook is an open-source web application used to create and share documents that contain live code, equations, visualizations, and narrative text, and it was used to write and run the code for the project.
5. NumPy: NumPy is a Python library used for numerical computing, and it was used to perform calculations on the data generated by the simulation.
6. Matplotlib: Matplotlib is a Python library used for data visualization, and it was used to plot the results of the simulation.

Design of the Project

The design of the project can be broken down into several components, as follows:

1. Simulation Environment: The simulation environment is created using Pygame, which allows for the creation of custom tracks and the placement of sensors on the self-driving car. The environment also includes a fitness function that evaluates the performance of the car on each track.
2. Self-Driving Car: The self-driving car is equipped with various sensors that detect the environment and make decisions about its movements. The sensors include distance sensors that detect the proximity of obstacles, a speed sensor that detects the current speed of the car, and a steering sensor that determines the direction of the car.
3. NEAT Algorithm: The NEAT algorithm is a genetic algorithm that evolves neural networks to perform a specific task. The NEAT-Python library is used to implement the algorithm, and it is used to evolve the neural network for the self-driving car.
4. Training Process: The training process involves evaluating the performance of the self-driving car on each track and using the NEAT algorithm to evolve the neural network until optimal performance is achieved. The training data is generated by running the simulation multiple times with different parameters and configurations.
5. Performance Analysis: The performance of the self-driving car is analyzed by plotting the results of the simulation using the Matplotlib library. The plots show the progress of the car's performance over time, as well as any improvements resulting from the training process.

Overall, the project design emphasizes the use of Pygame to create custom tracks and the implementation of the NEAT algorithm to evolve the neural network for the self-driving car. The training process is iterative, with the NEAT algorithm evolving the neural network over multiple runs of the simulation. The performance analysis component allows for the visualization of the car's progress and any improvements resulting from the training process.

Implementation

The first code block defines a class **NN** that implements a neural network visualizer. It uses the NEAT (NeuroEvolution of Augmenting Topologies) algorithm to evolve the neural network topology. The class constructor takes a configuration object, a genome object, and a tuple that represents the position where the neural network should be displayed. It creates a list of nodes and a list of connections based on the genome object, then initializes the nodes and connections with the required attributes, such as color and position, and finally draws the nodes and connections on a Pygame surface.

The second code block defines a class **CarAI** that uses the neural networks evolved by the NEAT algorithm to control a car in a game. It takes a list of genomes, a configuration object, and a starting position as inputs. It initializes the neural networks for each genome and creates a list of cars that are controlled by the neural networks. In the **compute** method, it activates the neural network for each car, gets the output of the neural network, and updates the car's state based on the output. It also updates the fitness of each genome based on the performance of the corresponding car. The **compute** method also updates the neural network visualization by refreshing the input values and the output values of the neural network nodes.

The code starts by importing the necessary modules and classes. These include **neat** for the NEAT algorithm, **pygame** for the simulation environment, and custom classes **Node**, **Connection**, **NodeType**, and **Color** for the neural network visualization.

NN class, which represents the neural network. It includes two class attributes that specify the number of input and output neurons in the network.

The constructor for the **NN** class, It initializes several instance variables, including **nodes** (an empty list to store the nodes), **genome** (the genome of the neural network), **pos** (the position of the network in the visualization), **input_names** (a list of names for the input nodes), **output_names** (a list of names for the output nodes), **hidden_nodes** (a list of hidden node IDs), and **node_id_list** (a list of IDs for all nodes in the network).

The code creates the input nodes of the network. It calculates the height **h** of the input layer, creates a **Node** object for each input node, and appends it to the **nodes** list. It also appends the input node ID to the **node_id_list**.

The output nodes of the network calculate the height **h** of the output layer, creates a **Node** object for each output node, and appends it to the **nodes** list.

The **nodes** list will contain instances of the **Node** class, representing the neurons of the neural network. The **genome** attribute of the **NN** instance is set to the **genome** parameter. The **pos** attribute is set to the **pos** parameter plus an offset to center the network. The **input_names** list contains the names of the input nodes. The **output_names** list contains the names of the output nodes. The **hidden_nodes** list contains the IDs of the hidden nodes in the genome. The **node_id_list** list will contain the IDs of all the nodes in the genome.

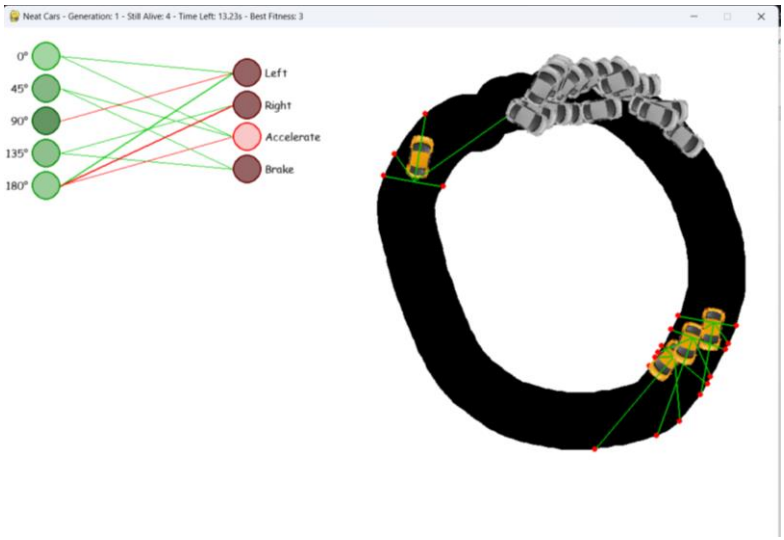
The **h** variable is set to the total height of the input layer, based on the number of input nodes. A **for** loop is used to create an instance of the **Node** class for each input node, and append it to the **nodes** list. The **node_id_list** list is updated to include the ID of the new node.

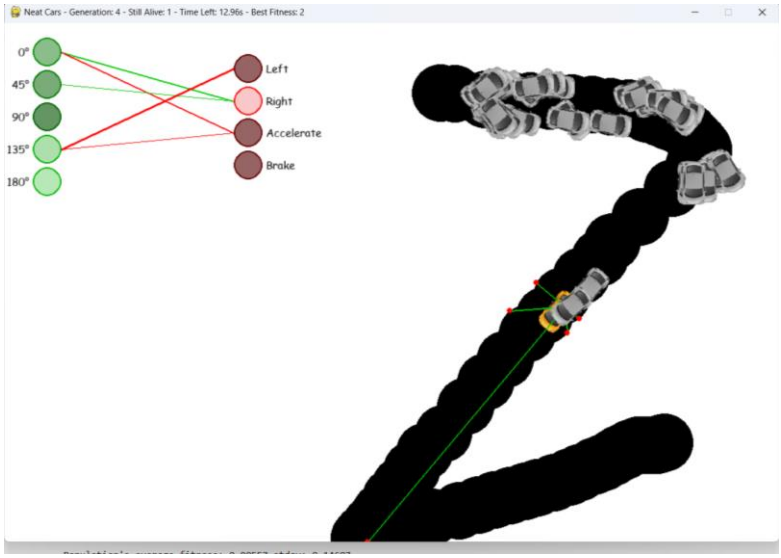
The **h** variable is set to the total height of the output layer, based on the number of output nodes. Another **for** loop is used to create an instance of the **Node** class for each output node, and append it to the **nodes** list. The **hidden_nodes** list is updated to remove the ID of the output node, and the **node_id_list** list is updated to include the ID of the new node.

The **h** variable is set to the total height of the hidden layer, based on the number of hidden nodes. Another **for** loop is used to create an instance of the **Node** class for each hidden node, and append it to the **nodes** list. The **node_id_list** list is updated to include the ID of the new node.

The **connections** list will contain instances of the **Connection** class, representing the connections between the neurons of the neural network. Another **for** loop is used to create an instance of the **Connection** class for each enabled connection in the genome, and append it to the **connections** list.

Results





References

[1] https://neat-python.readthedocs.io/en/latest/

[2] https://www.pygame.org/docs/

[3] http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

[4] https://blog.coast.ai/lets-evolve-a-neural-network-with-a-genetic-algorithm-code-included-8809bece164

[5] https://www.youtube.com/watch?v=MMxFDaIOHsE

[6] https://github.com/CodeReclaimers/neat-python/tree/master/examples/xor