**DEVELOPMENT AND SIMULATION OF AUTONOMOUS CAR**

***A project report***

***Submitted in partial fulfillment of the***

***requirements for the award of the Degree of***

**BACHLOR OF TECHNOLOGY**

BY

**ADARSH PRAKASH (BTech/15136/18)**

**MD AQUIB AMIN (BTech/15162/18)**

**AMAN RAJ (BTech/15174/18)**

**Logo

Description automatically generated**

**DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGG**

**BIRLA INSTITUTE OF TECHNOLOGY**

**MESRA, PATNA CAMPUS-800014**

**2022**

**DECLARATION CERTIFICATE**

This is to certify that the work presented in the thesis entitle **“Development and Simulation of Autonomous car (Self Driven car)”** in partial fulfilment of the requirement for the award of Degree of **Bachelor of Technology in Electronics and Communication** of Birla Institute of Technology Mesra, Patna Campus is an authentic work carried out under my supervision and guidance.

To the best of my knowledge, the content of this project does not form a basis for the award of any previous Degree to anyone else.

Date: (Prof. Rajeev Ranjan)

Dept. of Electronics & Communication

Birla Institute of Technology, Mesra

Patna Campus

(INCHARGE) (UG Project Coordinator)

(Dept. of Electronics & (Dept. of Electronics &

Communication Engg.) Communication Engg.)

Birla Institute of Technology Birla Institute of Technology

Mesra, Patna Campus, 800014 Mesra, Patna Campus, 800014

**CERTIFICATE OF APPROVAL**

The forgoing project entitled **“Development and Simulation of Autonomous car”**, is hereby approved as a creditable study of research topic and has been presented in a satisfactory manner to warrant its acceptance as prerequisite to the degree for which it has been submitted.

It is understood that by this approval, the undersigned do not necessarily endorse any conclusion drawn or opinion expressed there in but approve the project for the purpose for which it is submitted.

**(Internal Examiner) (External Examiner)**

**(Chairman)**

**Head of Department**

**ACKNOWLEDGEMENT**

We are overwhelmed in all humbleness and gratefulness to acknowledge our depth to all those who have helped us to put these ideas, well above the level of simplicity and into something concrete. On the very outset of this report, we would like to extend our sincere & heartfelt obligation towards all the personages who have helped us in this endeavour. Without their active guidance, help, cooperation & encouragement, we could not have made headway in the project. We are ineffably indebted to Professor Rajeev Ranjan for his valuable guidance and constant supervision as well as for providing necessary information regarding the project and also for his support in completing the project. We are very grateful to his for his fruitful guidance, inspiration, advice and constant encouragement during the entire course of the project. He has always fuelled our thoughts to think broad and out of the box.

We would also like to thank In-charge (ECE), Dr. Ritesh Kumar Badhai who, in spite of his busy schedule, always guided us in the right direction. We would also like to thank our project coordinator, Dr. Megha Dadel. We are indebted to the members of department of Electronics and Communication, Birla Institute of Technology, Mesra, Patna Campus for their kind co-operation, facilities and encouragement which helped us in completion of this project.

We would like to express our gratitude towards our parents and family members for their patient guidance. Last but not the least we would like to express our sincere gratitude towards our friends and colleagues who have willingly and enthusiastically helped us in completing this project.

**ADARSH PARAKSAH**

**MD. AQUIB AMIN**

**AMAN RAJ**

**ABSTRACT**

This study will cover the simulation and development of a self-driving autonomous car, using reinforcement learning (Q-table, and NeuroEvolution of Augmented Topologies), and compare the results.

This study begins with the simulation of a real world-like physics of a car, i.e., its momentum, friction, acceleration, and front wheel steering to create an environment on which AI is implemented using algorithms like Q-Table and NeuroEvolution of Augmented Topology.

Path finding algorithms (Dijkstra Algorithm) and Path following algorithms are also implemented into the environment.

Apart from these packages various plotting packages such as matplotlib were used to plot the graphs of the training results, and Graphviz to plot the trained network itself

All this is implemented using python and its different libraries, pygame is used to render everything, neat-python is used to implement the AI.

Further these two algorithms were compared on their training time, model size, and interference accuracy. The findings so obtained have been found to be insightful.

**TABLE OF CONTENTS**

**DECLARATION CERTIFICATE 2**

**CERTIFICATE OF APPROVAL 3**

**ACKNOWLEDGEMENT 4**

**ABSTRACT 5**

**TABLE OF CONTENTS**

**LIST OF FIGURES**

1. **Introduction 11**
   1. Motivation 12
   2. Literature Review 13
   3. Project structure 14
2. **Simulating the Environment 15**
   1. Car Physics 16
   2. Collision detection 17
   3. Sensors 18
   4. Rendering 19
3. **Route Selection 21**
   1. Shortest Path Finding 22
   2. Path following 23
4. **Implementation of AI 24**
   1. Reinforcement Learning 25
      1. Q-Learning 26
         1. Q-Table 27
         2. NEAT 28
5. **Python Packages 29**
   1. NumPy 30
   2. Matplotlib 30
   3. NEAT-python 31
6. **Training the model 32**
   1. Important definitions 33
      1. Q-value 33
      2. Reward 33
      3. Episode 34
      4. Markov Decision Process(MDP) 34
      5. State 34
      6. Epsilon 34
      7. Fitness Function 35
      8. Generations 35
      9. Standard deviation 35
   2. Q-Table training 36
   3. NEAT training 36
7. **Result 40**
8. **Conclusion 45**
9. **Future work 46**
10. **References 47**

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Fig no.** | **Fig Name** | **Page no.** |
| Fig. 2.2.1 | Rotation of a rectangle with pivot other than its center. |  |
| Fig. 2.3.1 | Each sensors measuring the distance from the border. |  |
| Fig. 2.4.1 | Render of the map of our college. |  |
| Fig. 3.1.1 | Rendering the path on the map, showing path finding in action. |  |
| Fig. 3.2.1 | Distance of a point from a line segment. |  |
| Fig. 4.1.1 | Model of Reinforcement learning. |  |
| Fig. 4.1.1.1.1 | Q-Matrix |  |
| Fig. 4.1.1.2.1 | Genetic algorithm modifying the connections of the neural network. |  |
| Fig. 6.1.1.1 | Q-value in a Q-matrix. |  |
| Fig. 6.2.1 | Data log of Q-table algorithm. |  |
| Fig. 6.3.1 | Data log of NEAT algorithm. |  |
| Fig. 6.3.2 | Genetic algorithm modifying the connections of the neural network. |  |
| Fig. 7.1 | Graph of average, maximum and minimum fitness of the Q-table algorithm after training. |  |
| Fig. 7.2 | Graph of average, maximum fitness and standard deviation of the trained NEAT model after training on a single path. |  |
| Fig. 7.3 | Graph of Speciation of the NEAT algorithm after training on a single path. |  |
| Fig. 7.4 | Structure of Trained NEAT model after training won a single path. |  |
| Fig. 7.5 | Graph of average, maximum fitness and standard deviation of the trained NEAT model after training on random paths. |  |
| Fig. 7.6 | Graph of Speciation of the NEAT algorithm after training on random paths |  |
| Fig. 7.7 | Structure of Trained NEAT model after training on random paths. |  |

**CHAPTER 1**

1. **INTRODUCTION**

As the world evolves, scientists and researchers are struggling to move human life into a more comfortable place. People around the world are now excited about the launch of self-driving cars.

Studies show some intriguing predictions, such as,

90% reduction in traffic deaths, according to WHO approximately 1.3 million people die each year due to vehicle crashes (94% of which are down to human error).

60% drop in harmful emissions, traffic congestion and reckless driving are the reason for huge amount of harmful emissions which result in global warming, use of self-driven cars can reduce this emission by 60% due to its efficient ways.

40% reduction in travel time, due to its efficient way of vehicle control, self driven cars can reduce travel time by 40%.

So, the objective of the project is to implement a self driven car, using machine learning, and simulating the results.

By simulating the environment, we also avoid any risks of crashes which can and will occur in the training phase.

The main aim of the autonomous decision system is to create some decisions for the self-driving car including path planning, navigation, obstacle avoidance etc. As an example, within the path planning, the autonomous decision system plans a worldwide path in keeping with the target destination and current location firstly, then plans a neighbour path for the autonomous car by merging the local environment information provided by the environment perception system and also the overall path.

Many of researchers predicted that by 2030, autonomous vehicles will be sufficiently reliable, affordable and common to take the place most human driving cars, providing huge savings and benefit. Most experts have predictions are made by people with financial interests in the industry, based on participation with disorderly technologies such as digital cameras, smart phones and personal computers. They tends to ignore remarkable obstacles to self-driving vehicle development, and exaggerate future benefits.

* 1. **MOTIVATION**

Every year the lives of around 1.3 million people are break off as a result of a road accident crash. Around 20 to 50 million people are suffer from non-fatal injuries, with many incurring a disability as a result of their injury.

Road traffic injuries cause extensive economic losses to individuals, their families, and to country as a whole. These losses arise from the cost of treatment as well as lost productivity for those killed or disabled by their car accidents, and for family members they need to take time off from work to care for the injured which causes their financial losses also. Road traffic crashes cost most countries 5% of their gross domestic product

Car accidents in India

India ranks first in the number of road accident deaths among the 194 countries and accounts for almost 12% of the accident related deaths. A total number of 459,002 accidents took place in the country during the calendar year 2020leading to 151,123 deaths and 471,361 injuries.

Road accidents are very serious and are often the result of an interplay of various factors. like (i) human error (ii) Bad driving skills and (iii) vehicular condition.

We are also a very long way away from having a true self-driving car. By a true self-driving car, it means that a car which can be essentially driven in any manner as a human driving a car.

Major automobile companies are trying to achieve true self-driving car. The main motivations behind the idea is that:

* Safer Roads
* Increase in productivity
* More economical
* More environment friendly
  1. **Literature review**
     1. **The Neuroevolution of augmenting topologies (NEAT), Stanley K.:**

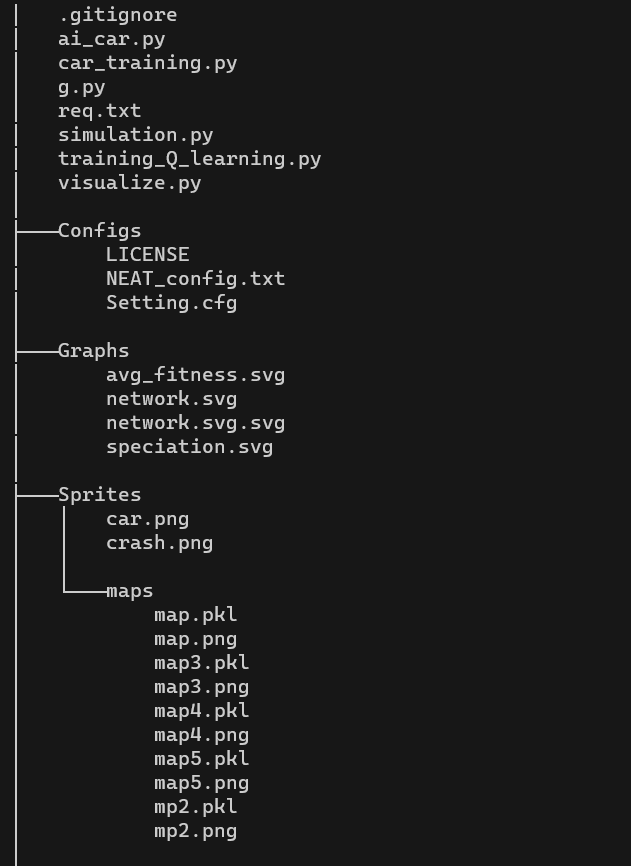
Neruroevolution, i.e., evolving artificial neural networks with genetic algorithms, is a type of reinforcement learning, where the structure of the network is not fixed and it evolves with every generation, according to the reward it receives. NEAT creates a generation with some given population (which we specify) with each individual agent having a different genome. Using the rewards from each action it changes the connections, and/or their weights and even adds new nodes.

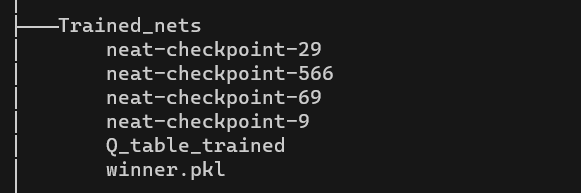
It has been very effective in reinforcing learning tasks, especially those networks with hidden state information. An important question in neural evolution is how to take advantage of the evolution of neural network topology as well as weights. NeuroEvolution of Augmented Topologies (NEAT) that outperforms the best fixed topology methods in a challenging benchmarking reinforcement learning task. We assert that the increased efficiency is due to (1) using a principal approach that combines different topologies, (2) preserving structural innovation using specifications, and (3) for gradual growth from minimal structure. We test this claim through a series of resection studies demonstrating that each component is required for the entire system and for the other. The result is significantly faster learning. NEAT is also an important contributor to GAs because shows how evolution can simultaneously optimize and complement solutions, allowing to develop increasingly complex solutions. complex over time, thus reinforcing the analogy with biological evolution.

* + 1. **Dijkstra algorithm, E.W. Dijkstra:**

Dijkstra's algorithm (named after its discover, E.W. Dijkstra) solves the problem of finding the shortest path from a point in a graph (the source) to a destination. It turns out that one can find the shortest paths from a given source to all points in a graph in the same time, hence this problem is sometimes called the single-source shortest paths problem. This paper will help you understand the underlying concepts of Dijkstra Algorithm with the help of simple and easy to understand examples and illustrations.

* 1. **Project Structure**

****

****

**CHAPTER 2**

1. **Simulating the environment**

Simulating the environment requires simulating the momentum, acceleration, velocity, friction, breaking, and other mechanics, like, front-wheel steering of the car.

To render everything on the screen using the python package, pygame.

**2.1. Car Physics**

To implement the momentum, acceleration, velocity, friction, breaking of the car we have implemented the laws of linear motion in our code. It was necessary to simulate the environment as close to reality as possible because this will able us to train our car in the simulated environment and then just use the trained AI into real world problems.

Here is the code snippet,



We also implemented front-wheel steering using the rotation of a rectangle offset from the center.

**2.2. Collision detection**

The collision detection in this simulation is done by the use of hitboxes. A hitbox is an invisible box which boundaries all parts of the car, so when the hitbox is crosses any part of the road or hits any obstacle, we know that the car had also hit the boundary or an obstacle.

In our case just for the ease of implementation we have used four points at each edge of the car for the hitbox.

Here’s a snippet of the code of checking the hitboxes for any crash,



So, we are just checking if the given coordinates of the hitbox are on the specified path, and are also on the road or not. For that we just check the given coordinate (pixel) of the map image for the color of the road.

To rotate the hitbox with the car, we implemented the formula of rotation of a rectangle with pivot at a point other than its center. For this we first shift the rectangle to the origin rotate it to the given angle then shift it back to its centre.

Fig. 2.2.1

**2.3. Sensors**

We have used distance sensors which measures the distance between the car and the obstacle and gives this distance as the output. This output is then fed to the neural network.

We have simulated 8 sensors, 45 degrees to each other on the car, which measures the distance between the car and an obstacle.



Fig. 2.3.1

For calculating the output of the sensors, we just check collision in a straight line at the angle of the sensor, till we detect collision.

Here is the snippet of the code used to calculate the output of the sensors,



**2.4. Rendering**

To render everything on the screen we have used the python package, Pygame, which has been already defined in this report. This is a CPU intensive process, so in the training phase we have only rendered the environment only a few times.

Here is the final render of the environment,

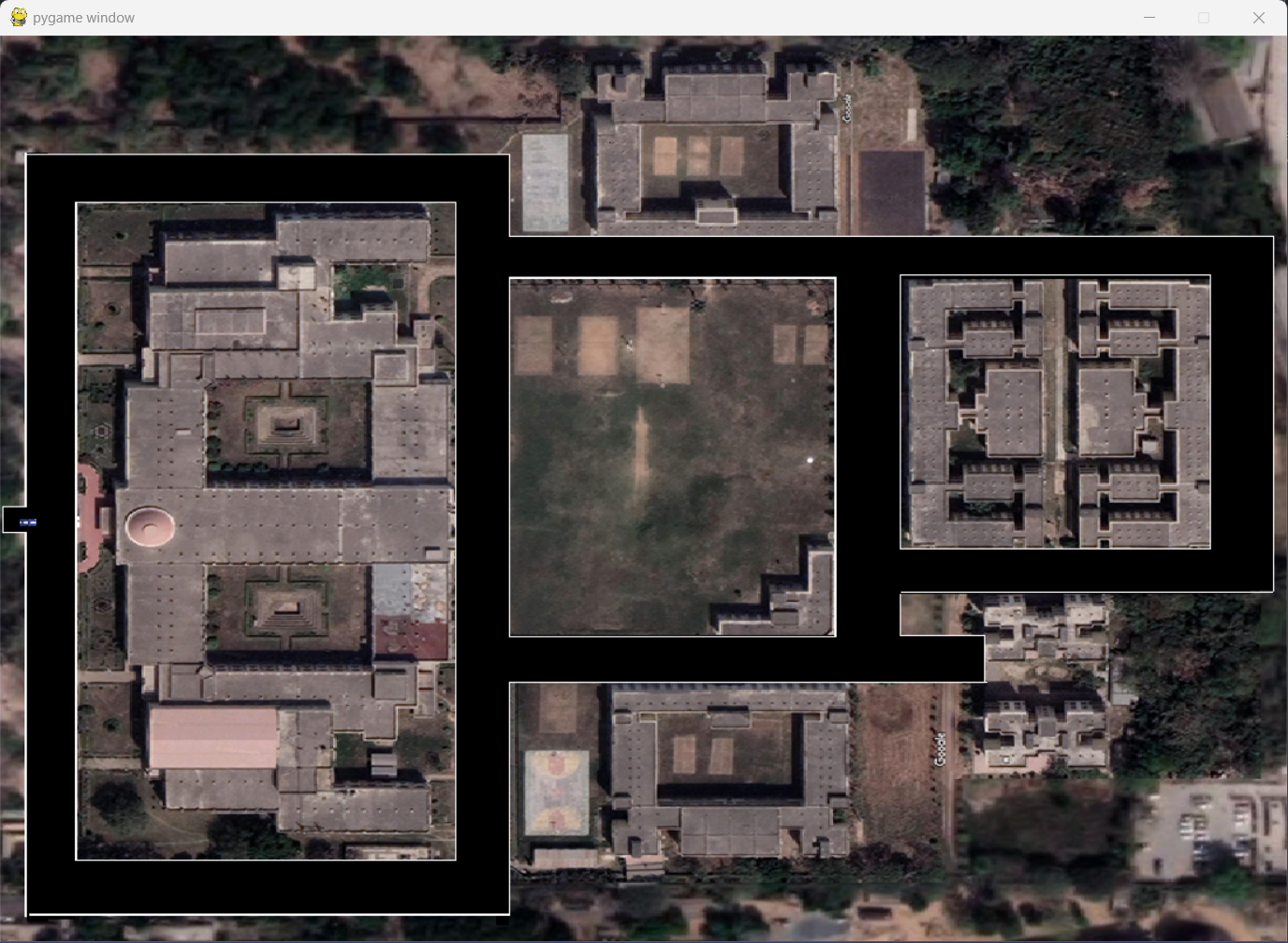


Fig. 2.4.1

We have used the map of our college campus as the environment for training which we have obtained from Google maps. We have also painted the roads black for ease of implementation, and the borders white just for clarity of viewing.

**CHAPTER 3**

**3. Route Selection**

Route selection is just selecting the start and destination for the car and then selecting the shortest path from the start to the destination, and then making the car follow the given path to reach the destination. To implement this, we had to make a graph resembling the paths of the given map on which route selection can be implemented.

For example, the graph we created for the map of our college campus is,

{(16, 412) : [((43, 412), 0)],

        (43, 412) : [((16, 412), 0), ((43, 120), 0), ((43, 727), 0)],

        (43, 120) : [((412, 120), 0), ((43, 412), 0)],

        (412, 120) : [((412, 188), 0), ((43, 120) , 0)],

        (412, 188) : [((612, 188), 0), ((412, 120), 0), ((412, 420), 0)],

        (412, 420) : [((412, 530), 0), ((412, 188), 0)],

        (412, 530) : [((620, 530), 0), ((412, 420), 0), ((412, 727), 0)],

        (412, 727) : [((412, 530), 0), ((43, 727), 0)],

        (43, 727) : [((412, 727), 0), ((43, 412), 0)],

        (612, 188) : [((740, 188), 0), ((412, 188), 0)],

        (740, 188) : [((612, 188), 0), ((740, 322), 0), ((1060, 188), 0)],

        (740, 322) : [((740, 454), 0), ((740, 188), 0)],

        (740, 454) : [((740, 530), 0), ((740, 322), 0)],

        (740, 530) : [((822, 530), 0), ((740, 454), 0), ((620, 530), 0)],

        (620, 530) : [((740, 530), 0), ((412, 530), 0)],

        (822, 530) : [((740, 530), 0)],

        (1060, 188) : [((1060, 322), 0), ((740, 188), 0)],

        (1060, 322) : [((1060, 454), 0), ((1060, 188), 0)],

        (1060, 454) : [((1060, 322), 0), ((740, 454), 0)]}

* The node of the graph are the pixels coordinates on the map.

To implement route selection, we have implemented two algorithms, which are shortest path finding (Dijkstra algorithm), and path following.

* 1. **Shortest Path finding (Dijkstra Algorithm)**

The Dijkstra algorithm is an algorithm for finding short paths between nodes on a graph, which may represent, for example, road networks. Dijkstra is based on the fact that the sub path of a shortest path is itself a shortest path on its own.

The algorithm comes in many forms. The original Dijkstra algorithm found a shortcut between two given nodes, but the common variant corrects one location as the "source" area and finds shortcuts from source to all other nodes in the graph, producing a shorter route path.

In the source code provided in the graph, the algorithm finds the shortest path between that node and the rest. It can also be used to find the shortest routes from one destination to another destination by setting up an algorithm once the shortest route to the destination has been determined. For example, if graph nodes representing cities and curb costs represent driving distances between pairs of cities connected by a straight road (for convenience, ignore red lights, stop signs, toll roads and other obstacles), the Dijkstra algorithm is used. to find a shorter route between one city and all other cities. The most widely used shortcut algorithms are network router agreements, in particular IS-IS (Intermediate and Intermediate System) and Shortcut Shortcuts (OSPF). It is also used as a subroutine in other algorithms similar to Johnson's.

To implement Dijkstra in this map we ran Dijkstra on the graph we created.

Here we have selected a start and a destination (Main gate to Boys Hostel-3) and calculated the shortest path using Dijkstra.



Fig. 3.1.1

* 1. **Path following**

The path following algorithm which we have used first takes the shortest path which we found using Dijkstra, which is just a list of nodes (coordinates) which forms the shortest path and makes all other paths out of bound. For this we took a padding of 30 pixels from the line connecting to coordinates of the shortest path, which will be the only path the car is allowed to run.

To calculate the distance from a line segment for padding we used the algorithm, Minimum distance between a point and a line by Paul Bourke.

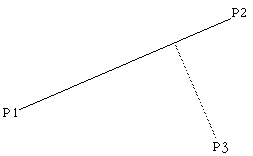


Fig. 3.2.1

The point P3 (x3, y3) is closest to the line at the tangent to the line which passes through P3, that is, the dot product of the tangent and line is 0, thus

(P3 - P) dot (P2 - P1) = 0

Substituting the equation of the line gives,

[P3 - P1 - u(P2 - P1)] dot (P2 - P1) = 0

Solving this gives the value of u

Substituting this into the equation of the line gives the point of intersection (x,y) of the tangent as

x = x1 + u (x2 - x1)

y = y1 + u (y2 - y1)

The distance therefore between the point P3 and the line is the distance between (x,y) above and P3.

**CHAPTER 4**

1. **Implementation of AI**

To implement AI in this environment we have used reinforcement learning.

* 1. **Reinforcement Learning**

Reinforcement learning is a subset of Machine Learning. It is about intelligent agents taking suitable action to maximize reward in a particular situation. It is used by different software and machines to find the best possible behavior or path that it should follow in a particular situation. Reinforcement learning differs from supervised learning in that, in supervised learning, the training data contains the answer key, so the model itself is trained with the correct answer, while in reinforcement learning, there is no answer, but the reinforcer decides what to do to perform the given task. In the absence of a training dataset, there is bound to be experience.

In reinforcement learning, developers design a method to reward desired behaviors and punish negative behaviors. This method assigns positive values ​​to desired actions so that encourages the actor and negative values ​​to undesirable behaviors. This programs the agent to look for the maximum aggregated reward in the long run to reach the optimal solution. These long-term goals help prevent the agent from focusing on less important goals. Over time, the agent learns to avoid the negative and look for the positive. This learning method has been applied in artificial intelligence (AI) as a way to drive unsupervised machine learning through rewards and punishments.

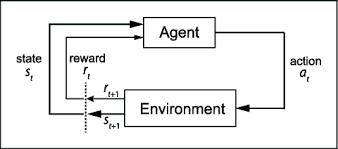


Fig. 4.1.1

The type of reinforcement learning we have used in this project is Q-Learning.

* + 1. **Q-Learning**

Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It requires no model of the environment (hence "no modules"), and it can handle random transitions and rewards problems without requiring throttling.

For any finite Markov decision process (FMDP), Q-learning finds the optimal policy in the sense of maximizing the expected value of the total reward over all successive steps, starting from state present. Q-learning can determine the optimal stock picking policy for any given FMDP, with an infinite polling period and a partially random policy. "Q" refers to the function the algorithm calculates - the expected reward for an action performed in a given state.

It works with Q bits, which tells the agent the optimal step. Before learning begins, Q is assigned any random value. Then at every episode the agent selects any action, and gets a reward, enters new state (that may depend on both the previous state and the selected action), and Q is updated. The core of the algorithm is a **Bellman equation** as a simple value iteration update, using the weighted average of the old value and the new information:

Shape

Description automatically generated with medium confidence

Where, rt is the received reward when the agent progresses from the state st to the state st+1, and α is the learning rate.

An episode of the algorithm ends when state st+1 is a final or terminal state. However, Q-learning can also learn in non-episodic tasks (because of the property of convergent infinite series). In the case, the discount factor is less than 1, the action values are finite, even if the problem contains infinite loops.

For all cases of final states sf, Q(sf, a) is never updated, rather is set to the reward value r observed for state sf. In most of the cases,Q(sf, a) can be taken to equal zero.

* + - 1. **Q-Table**

Q learning is a basic form of reinforcement learning that uses Q values ​​(also known as action values) to iteratively improve the behavior of a learning agent.

Q table is just a 2-D matrix, known as Q Matrix, which contains combinations of all the possible inputs and outputs. It returns the row with the highest value as output, and the reward is used to balance the matrix using the **Bellman equation** which backtracks the reward into the matrix and by this backtracking the agent learns to explore the environment in the most optimal way possible by each generation.

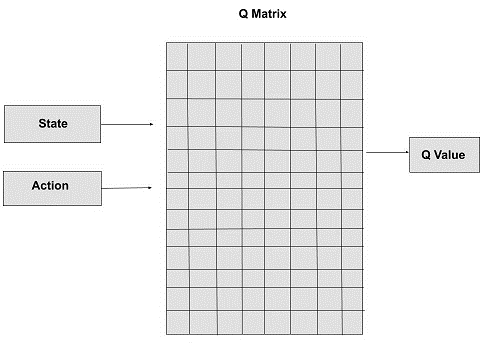


Fig. 4.1.1.1.1

Q value or action value: The Q value is defined for states and actions. Q(S, A) is an estimate of quality of action A in state S. This estimate is Q(S, A ) will be iteratively computed using the TD Update rule we will see in future topics.

During its lifecycle, the agent starts from the start state, making several transitions from the current state to the next depending on its choice of action and into the environment in which the agent Interactive. At each step of the transition, the agent of one state performs an action, observes the reward from the environment, and then transitions to the other state. If, at some point, the agent finds itself in one of the end states, this means that the transition is no longer possible. It is supposed to be the completion of an episode.

* + - 1. **NEAT**

Neruroevolution, i.e., evolving artificial neural networks with genetic algorithms, is a type of reinforcement learning, where the structure of the network is not fixed and it evolves with every generation, according to the reward it receives. NEAT creates a generation with some given population (which we specify) with each individual agent having a different genome. Using the rewards from each action it changes the connections, and/or their weights and even adds new nodes.

It has been very effective in reinforcing learning tasks, especially those networks with hidden state information. An important question in neural evolution is how to take advantage of the evolution of neural network topology as well as weights. NeuroEvolution of Augmented Topologies (NEAT) that outperforms the best fixed topology methods in a challenging benchmarking reinforcement learning task. We assert that the increased efficiency is due to (1) using a principal approach that combines different topologies, (2) preserving structural innovation using specifications, and (3) for gradual growth from minimal structure. We test this claim through a series of resection studies demonstrating that each component is required for the entire system and for the other. The result is significantly faster learning. NEAT is also an important contributor to GAs because shows how evolution can simultaneously optimize and complement solutions, allowing to develop increasingly complex solutions. complex over time, thus reinforcing the analogy with biological evolution.

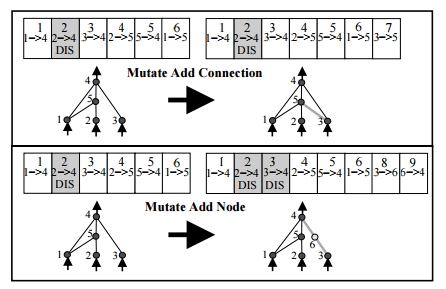


Fig. 4.1.1.2.1

This adapting of structure property of NEAT is what makes it one of the best algorithms to use in reinforcement learning.

**CHAPTER 5**

1. **Python Packages**
   1. **NUMPY**

NumPy is the basic package which can replace list and arrays for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as hidden arrays and matrices), and a series of procedures for fast array operations, including math, logic, shape manipulation, sorting, selection, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, stochastic simulations, and more.

It adds support for huge, multi-dimensional arrays and matrices, alongside an enormous collection of high-level Mathematical function to work on these multi-dimensional arrays. The predecessor of NumPy, Numeric, was initially made by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant made NumPy by integrating highlights of the contending Num array into Numeric, with a lot of extensive modification. NumPy is opensource software and has numerous donors.

* 1. **Matplotlib**

Matplotlib is a plotting library for the Python programming language. Matplotlib has a lot of numerical mathematics extension like NumPy.

It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, WX Python, Qt, or GTK+.

There is also a procedural "Pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged SciPy makes use of Matplotlib.

Matplotlib was originally written by John D. Hunter, has an active development community, and is distributed under a BSD-style license. Michael Droettboom was appointed as the lead developer of matplotlib shortly before John Hunter's death in August 2012 and stars Thomas Caswell.

* 1. **NEAT-python**

Neat-python is a python module which implements the NeuroEvolution of Augmenting Topologies in python.

In this module, a population of individual genomes is given, each genome contains two sets of genes that describes the structure of its neural network, which are,

* Node genes, which defines a single neuron
* Connection genes, which defines the connection between each neurons.

To scale the solution to a problem, the user must provide a fitness function that computes a unique real number that indicates the quality of the individual genome: better problem solving means higher scores than. The algorithm progresses through a user-specified number of generations, with each generation produced by reproduction (sexually or asexually) and mutation of the previous generation's healthiest individuals.

Duplication and mutation operations can add nodes and/or connections to the genome, so the genome (and the neural networks they create) can become increasingly complex as the algorithm advances. develop. When a predefined number of generations is reached, or when at least one individual (for the fitness criterion function max; other generations is configurable) exceeds the user-specified fit threshold determined, then the algorithm is terminated.

We are provided with a configuration file were we can input the number of input nodes, output nodes, hidden nodes, population size, activation function, mutation rate, bias, weights initialization, etc.

**CHAPTER 6**

1. **Training the model**

We then trained both our models, the main things to track during training the models are its Q-value, reward, episodes, epsilon, fitness, generations, genome, standard deviation.

* 1. **Important definitions**
     1. **Q-Value**

The ‘q’ in q-value stands for quality. The Q-Value represents how effective an action is in each state, in regards of gaining some future rewards. Higher the Q-Value better the action. During training this Q-value is balanced using the **Bellman equation** backtracking the reward.

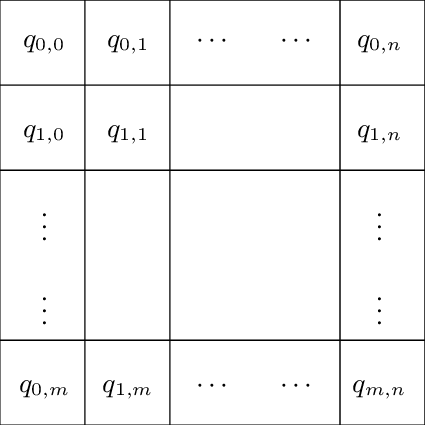


Fig. 6.1.1.1

* + 1. **Reward**

Reward is defined by the programmer. It is an incentive mechanism that guides the agent to the correct action using rewards and punishments. The agents goal is to maximize the total reward. All actions bring about rewards, which can be roughly divided into three categories: positive rewards emphasizing the desired action, negative rewards emphasizing the action that the agent must deviate from, and zero, that is, the agent does nothing special or unique.

* + 1. **Episode**

All states lie between the initial and final states; e.g. a game of chess. The employee's goal is to maximize the total reward they receive in an episode. In situations where there is no terminal state, an infinite set is considered. It is important to remember that the different volumes are completely independent of each other.

* + 1. **Markov Decision Process (MDP)**

The Markov property means that each state depends only on its previous state, the selected action is performed from that state, and the reward is received immediately after performing that action. Mathematically, this means: s' = s'(s, a, r), where s' is the future state, s is its previous state, and a and r are the action, and reward. There is no need to know what happened before s - the Markov property assumes that s contains all relevant information. The Markov decision process is a decision process based on these assumptions.

* + 1. **State**

Each scenario encountered by the agent in the environment is formally called a state. An agent moves from one state to another by performing actions. Also worth mentioning are the terminal states, which mark the end of an episode. No state can occur after reaching the terminal state and starting a new episode. Usually, a terminal state is represented as a special state where all actions move to the same terminal state with a reward of 0.

* + 1. **Epsilon**

The parameter Epsilon is related to the epsilon greedy action selection process in the Qlearning algorithm. In the action selection step, we choose the specific action based on the Q values ​​we already have. The epsilon parameter introduces randomness into the algorithm, forcing us to try different actions. This avoids being stuck in a local optimization.

If epsilon is zero, we never discover but always exploit existing knowledge. In contrast, setting epsilon to 1 forces the algorithm to always perform random actions and never use past knowledge. Usually epsilon is chosen as a small number close to zero.

* + 1. **Fitness Function**

A fit function is simply defined as a function that takes a candidate solution to a problem as input and outputs the "fit" or "good" degree of the solution to the problem under consideration.

The fit calculation is done many times in GA and so should be fast enough. A slow fitness value calculation can negatively affect GA and cause it to become exceptionally slow.

In most cases, the fit function and the objective function are the same because the goal is to maximize or minimize the given objective function. However, for more complex problems with multiple goals and constraints, an algorithm designer may choose to have another suitable function.

A fitness function must have the following characteristics –

* The fitness function must be fast enough to compute.
* It must quantitatively measure the adequacy of a given solution or how suitable instances can be generated from the given solution.
  + 1. **Generation**

Genetic algorithms simulate natural selection, which means that species that are able to adapt to changes in the environment can survive, reproduce and pass on to the next generation. Simply put, they simulate "survival of the fittest" between successive generations of individuals to solve a problem. Each generation consists of a population of individuals and each individual represents a point in the search space and a possible solution. Each individual is represented by a string of characters/integers/floats/bits. This sequence is similar to Chromosomes.

* + 1. **Standard Deviation**

The standard deviation is a number that describes the dispersion of values.

* A low standard deviation means that most numbers are close to the mean (mean).
* High standard deviation means the values ​​are spread over a wider range.
  1. **Q-Table training**

We have trained the Q-Table model for 5000 episodes on a single track and 5000 episodes on random tracks. The training phase took us around of 7 to 8 days to train the model. We trained the model on AMD Ryzen 5 4600Hz, with AMD Radeon Graphics 3.00 GHz, and 8.00 GB RAM.

During training we stored some data, like population-age, population-size, population-fitness, adjusted-fitness, stagnation, population’s average fitness, best fitness, average adjusted fitness, mean genetic distance, standard deviation, and generation time.

We have initialized a random NumPy array for the Q-table with 5 columns and 9 rows, using the following syntax,

*#q\_table initiation*

*q\_table = np.random.uniform(low = -2, high = 2, size = ([11, 11, 11, 11, 11] + [9])) #11x11x11x11x5 0-10 is the output of the sensors*

During the training phase we print some data related to the training process like, episode’s average fitness, minimum fitness, maximum fitness.

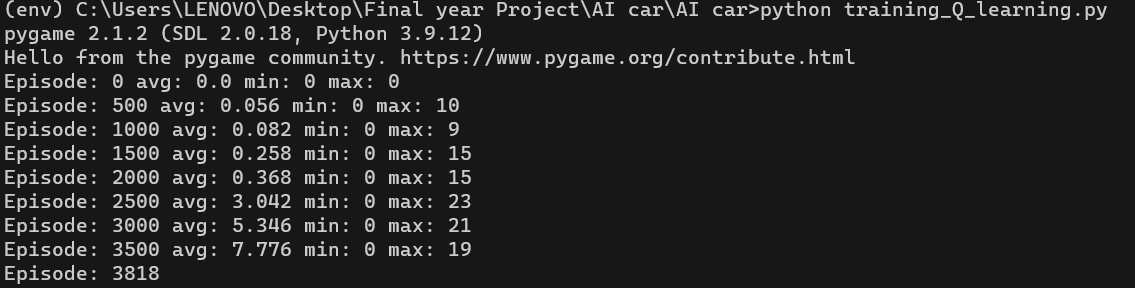


Fig. 6.2.1

We have also plotted the graph of average fitness through the episodes during the training phase.

* 1. **NEAT training**

We have trained the NEAT model for 1000 generations on a single track and 500 generations on random track. The training phase took us around of 7 to 8 days to train the model. We trained the model on AMD Ryzen 5 4600Hz, with AMD Radeon Graphics 3.00 GHz, and 8.00 GB RAM.

During training we stored some data, like population-age, population-size, population-fitness, adjusted-fitness, stagnation, population’s average fitness, best fitness, average adjusted fitness, mean genetic distance, standard deviation, and generation time.

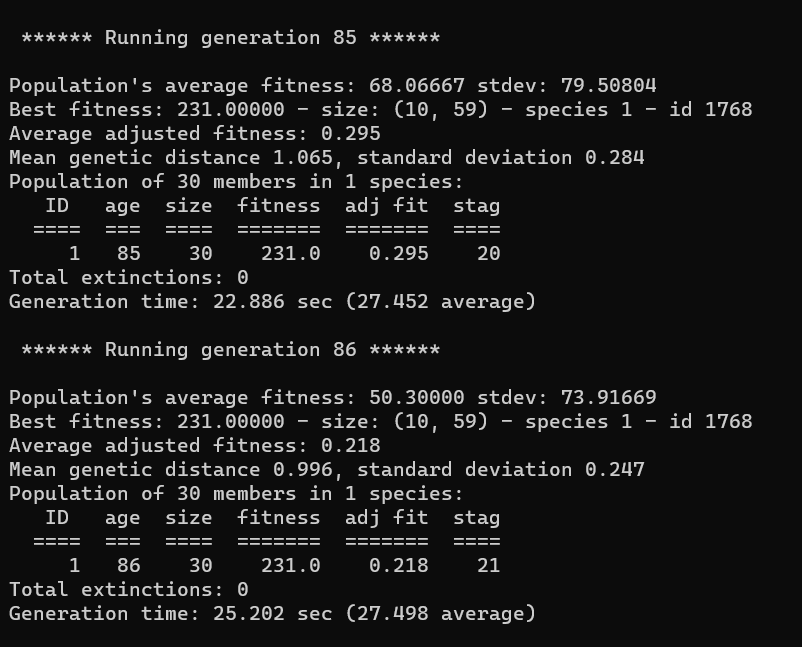


Fig. 6.3.1

We have also plotted some graphs to show to result and accuracy of the model, and the structure of the resulting neural network, like the average fitness of each generation, network, speciation.

During the training process the structure of the neural network changes according to the feedback it receives from the rewards.

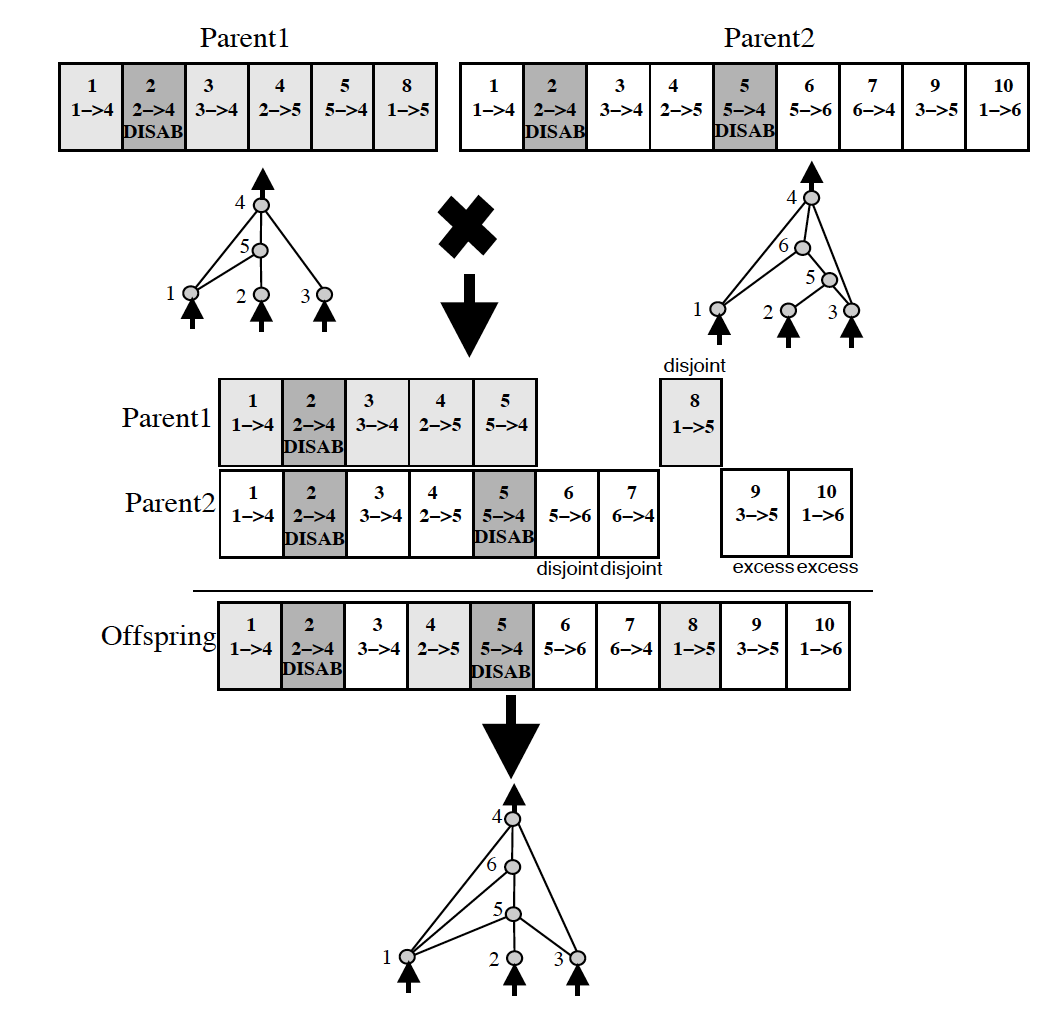


Fig. 6.3.2

We also added checkpoints after every 10 generations to save the progress of the training process in case of some mishap. The final trained network is also saved as ha pickle file to access if needed.

**Syntax:**

*#run simulation for a maximum of 1000 generations*

*population.run(run\_simulation, 1000***)**

**CHAPTER 7**

1. **Result**

After training the Q-Table algorithm for 5000 episodes, we get the graph of average fitness as below,

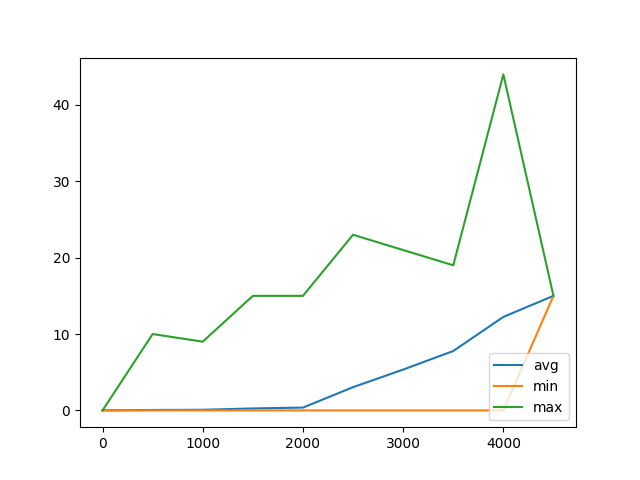


Fig. 7.1

We can here conclude that after training the agent for 5000 episodes, the average fitness is rising but there is still some room for improvement.

**Trained table:**

[[[[[[-3.74098189e+00 -3.33730611e+00 -1.25448581e+01 ...

-1.14939964e+01 -2.42207515e+00 -1.26673305e+01]

[-1.93817856e+00 -2.63215270e+00 -4.62704766e+00 ...

-4.64977547e+00 -4.65412856e+00 -3.96446117e+00]

[-1.80322658e+00 -1.39610506e+00 -1.68074944e+00 ...

-3.82626044e+00 -3.87873427e+00 -2.54690581e+00]

...

After training the NEAT algorithm agent for 500 generations on a single path, we plot the graph of average fitness of the generations,

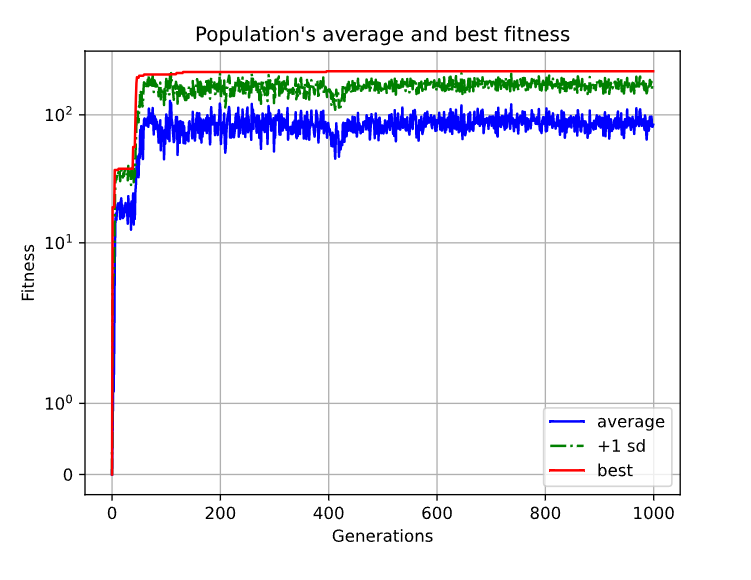


Fig. 7.2

Here we can see the much more random behavior of the NEAT algorithm during the training process, which is a characteristic of a genetic algorithm. We also conclude that after 200 generations the agent has almost learned its environment and there’s not much room for improvement here.

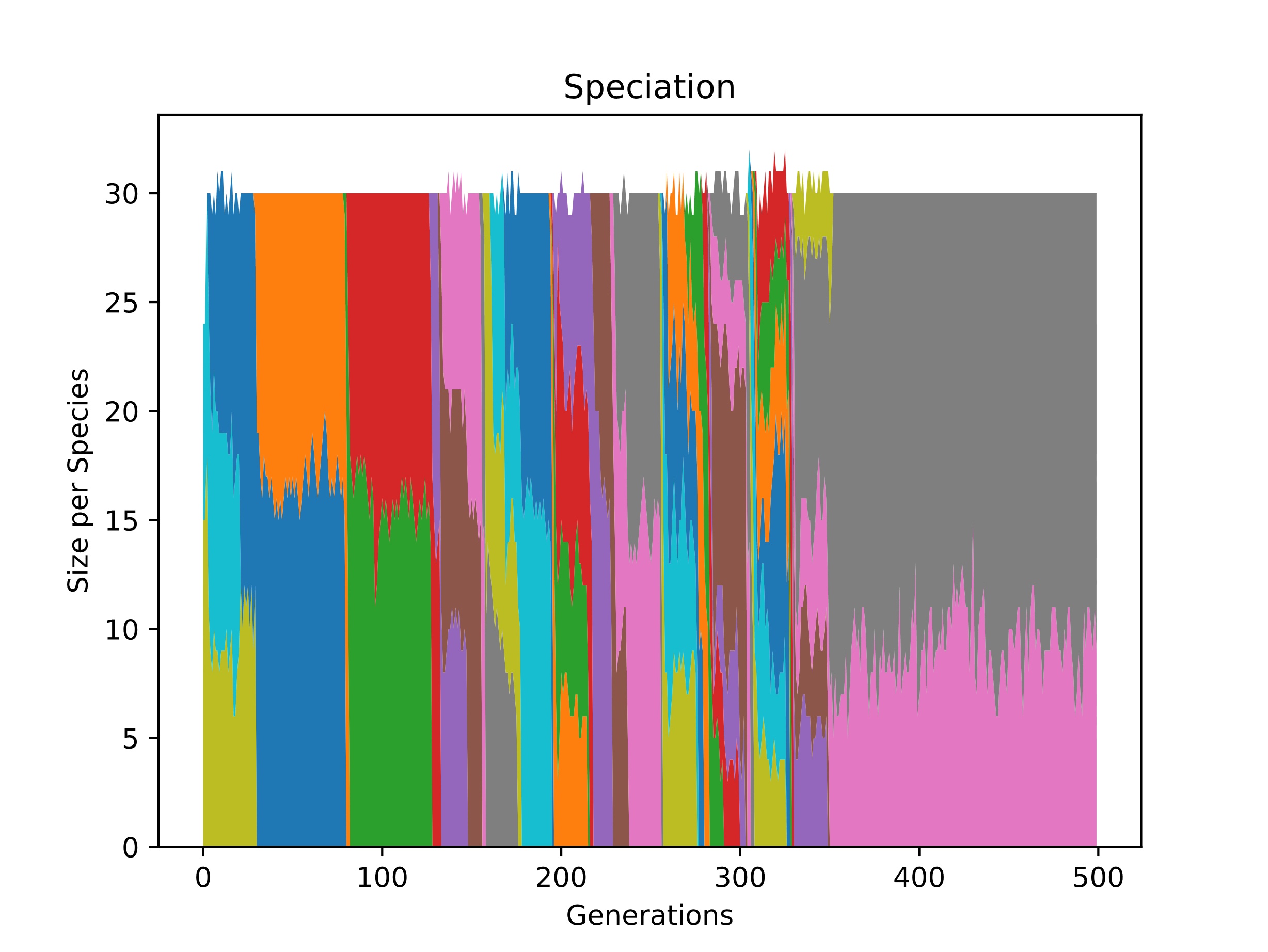
We have also plotted the graph for speciation in this case,

Fig. 7.3

Here, we can see numerous species starting and going extinct, but at the end the two best species remain, and every other species goes extinct. This is how this algorithm takes advantage of the genetic evolutionary theory.

The structure of the neural network is also plotted,

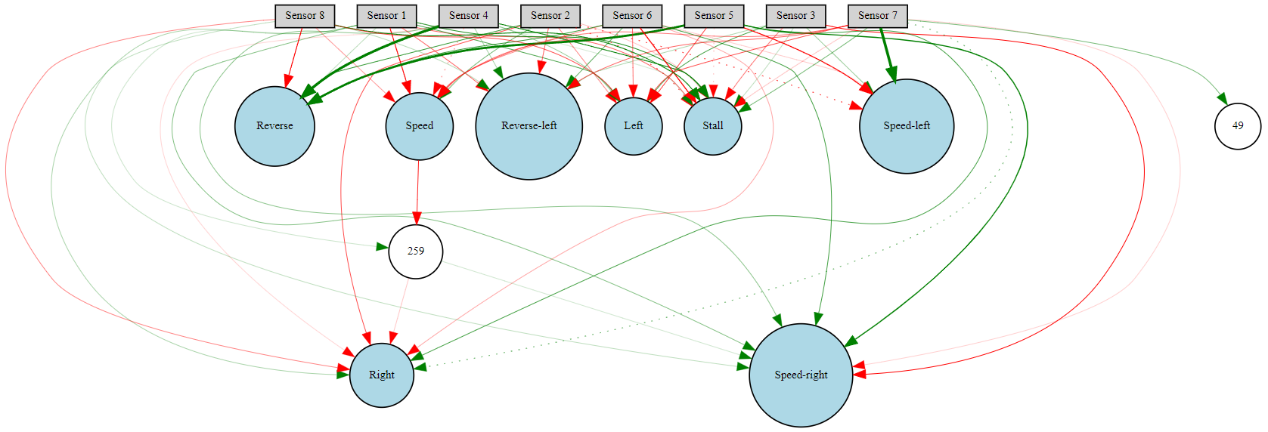


Fig. 7.4

There’s not much to conclude from the structure of a neural network, but it helps us to understand the inner working, or the thinking behind the neural network. It shows the connections between the input and output nodes.

Then we trained the NEAT algorithm for 500 generations of random paths, we plot the graph of average fitness of the generations,

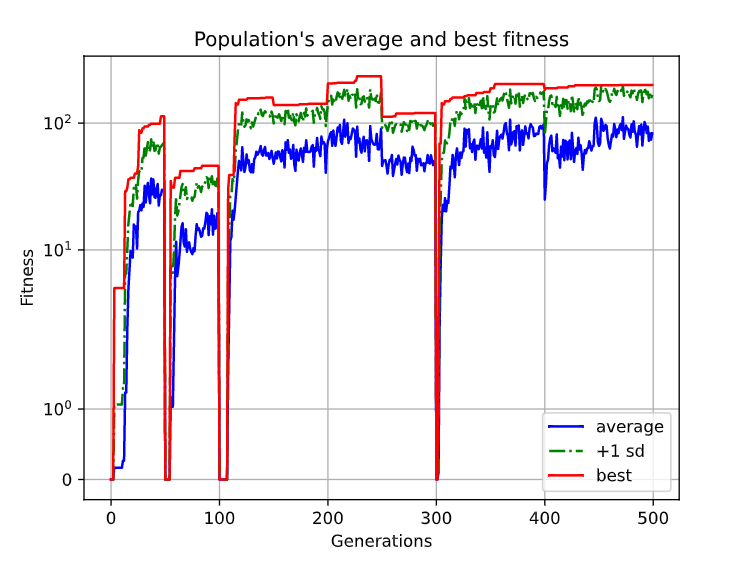


Fig. 7.5

Here we can see the much more random behavior of the NEAT algorithm during the training process, which is a characteristic of a genetic algorithm. The dips at every 50 generations are the point in the training process where we have changed the path to a different random path on the map. Smaller dips happen when the new random path is similar to the path the agent of already trained on, and the big dips happen when the new random path selected is totally different from the paths the agent is already been trained.

We have also plotted the graph for speciation in this case,

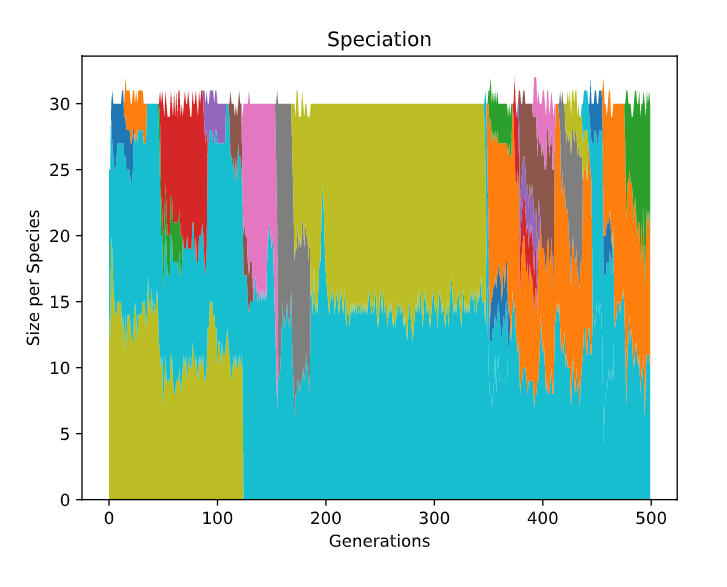


Fig. 7.6

The structure of the neural network is also plotted,

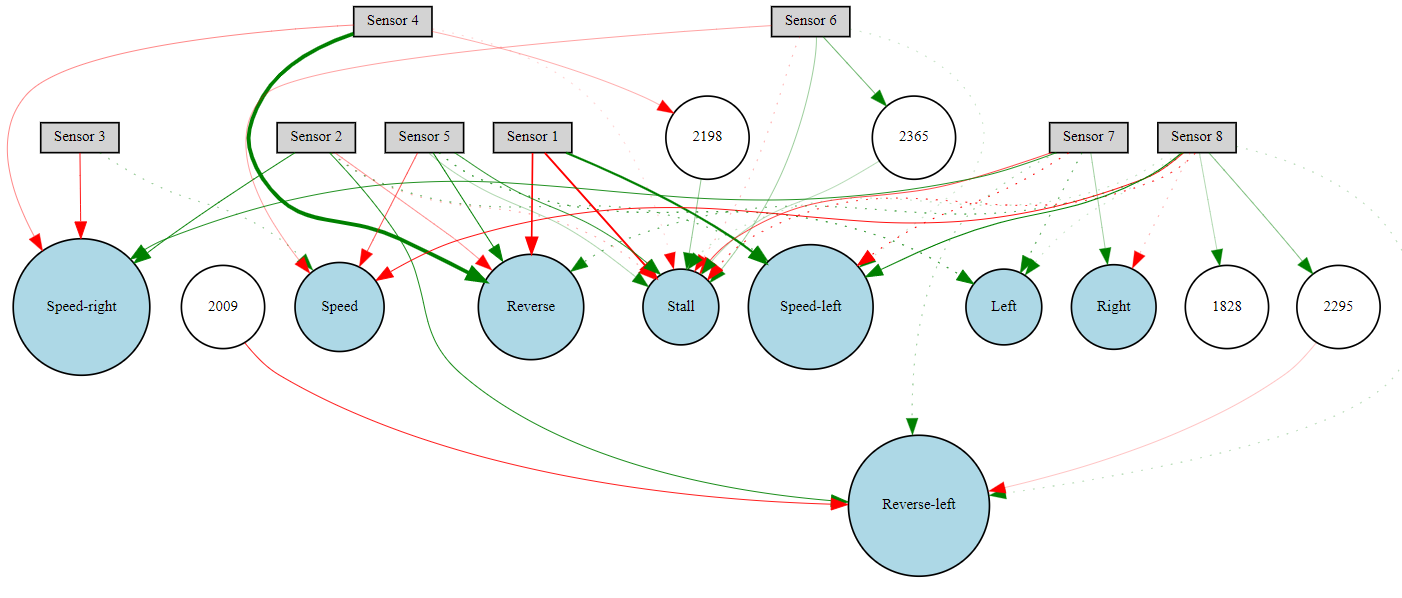


Fig. 7.7

This is a much more complex structure, due to the random path selection of the algorithm.

1. **CONCLUSION**

Here we conclude that if training the model on an easy path, the NEAT algorithm shows a clear-cut advantage in every aspect, but when we start to train the model on a complex path, both the algorithms have almost the same training time, and accuracy which is much different than what we assumed when we started this project.

NEAT being a much more complex algorithm based on life-like genetic behaviors of natural selection has some advantages on the very basic Q-table approach, but the difference is meniscal which is really fascinating,

1. **FUTURE WORK**

* Simulating the Environment in 3 Dimensional rather than 2 Dimensional, using the blender game engine.
* Implementation of deep neural network AI, using tensorflow and keras python libraries.
* Adding obstacles and other vehicles in the environment.
* Implementing a traffic system in the environment.
* Implementing road traffic rules.
* Taking map and road information from google maps.

1. **REFERENCES**

[1] K., S. (n.d.). *The neuroevolution of augmenting topologies (NEAT) users page*. NeuroEvolution of Augmenting Topologies. Retrieved May 5, 2022, from <http://www.cs.ucf.edu/~kstanley/neat.html#intro>

[2] *Welcome to neat-python's documentation!¶*. NEAT. (n.d.). Retrieved May 5, 2022, from <https://neat-python.readthedocs.io/en/latest/>

[3] 2022, *The neuroevolution of augmenting topologies (NEAT) users page*. NeuroEvolution of Augmenting Topologies. Available at: [http://nn.cs.utexas.edu/downloads/papers/stanle y.cec02.pdf](http://nn.cs.utexas.edu/downloads/papers/stanle%20y.cec02.pdf) [Accessed 5 May 2022].

[4] Mayank, M. (2018, August 18). *Reinforcement learning with Q tables*. Medium. Retrieved May 5, 2022, from https://itnext.io/reinforcement-learning-with-q-tables-5f11168862c8

[5] *Deep Q-learning*. GeeksforGeeks. (2019, June 18). Retrieved May 5, 2022, from https://www.geeksforgeeks.org/deep-q-learning/

[6] A. Bhalla, M. S. Nikhila and P. Singh, "Simulation of Self-driving Car using Deep Learning," 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS), 2020, pp. 519-525, doi: 10.1109/ICISS49785.2020.9315968.

[7] Carew, J. M. (2021, March 29). *What is reinforcement learning? A comprehensive overview*. SearchEnterpriseAI. Retrieved May 5, 2022, from https://www.techtarget.com/searchenterpriseai/definition/reinforcement-learning#:~:text=Reinforcement%20learning%20is%20a%20machine,learn%20through%20trial%20and%20error

[8] *Adjacency list*. Programiz. (n.d.). Retrieved May 6, 2022, from https://www.programiz.com/dsa/graph-adjacency-list

[9] Wikimedia Foundation. (2022, May 3). *Dijkstra's algorithm*. Wikipedia. Retrieved May 5, 2022, from https://en.wikipedia.org/wiki/Dijkstra%27s\_algorithm

[10] Bibuli, M., Caccia, M., & Lapierre, L. (2016, April 21). *Path-following algorithms and experiments for an autonomous surface vehicle*. IFAC Proceedings Volumes. Retrieved May 5, 2022, from https://www.sciencedirect.com/science/article/pii/S1474667015320747

[11] Violante, A. (2019, July 1). *Simple reinforcement learning: Q-learning*. Medium. Retrieved May 6, 2022, from https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

[12] <http://rmgi.blog/pygame-2d-car-tutorial.html>

[13] <https://stackoverflow.com/a/2233538>, distance between a point and a finite line segment represented by two points

[14] <https://stackoverflow.com/questions/15098900/how-to-set-the-pivot-point-center-of-rotation-for-pygame-transform-rotate>

[15] <https://www.youtube.com/watch?v=paU8JumpTXc> for recalculating hitboxes.