Computer Science and Artificial Intelligence

COC257

B822620

Autonomous Car Simulation

by

Kieran M. Apps

Supervisor: Dr. Shaheen Fatima

Department of Computer Science

Loughborough University

June 2022

Autonomous Car Simulation

Kieran Apps (B822620)

Loughborough University

Table of Contents

Section 1: Introduction and Description

1.1 Overview

Choosing this project was relatively easy for me as cars are a big passion of mine and the area of self-driving and autonomous cars has exploded in the rate of development and popularity in recent years, with the likes of Tesla and their own Autopilot system in their vehicles. As this technology develops further, it will become more prevalent across various vehicle applications especially as we see society's acceptance of self-driving cars and other autonomous vehicles.

Creating this project will give me great knowledge of this new emergent field of A.I. and allow me the learn and practically try out some of the key techniques that are used to create autonomous driving systems. The project itself will be a simulation of an autonomous car using Unity to host the simulated environment, track and car itself. Then, Python will be used to program the Artificial Intelligence used to control the car's actions thanks to its useful libraries and lots of support around the language for A.I..

Using this combination of Python and Unity allows me to utilize my knowledge of Python and allows me to explore a new area of Python, that being the A.I and Machine Learning side of the language, as well as explore a new language C# with Unity. Thus, developing my skills of developing A.I and learning new language quickly and efficiently such that I can implement the simulation needed.

I believe this project will also allow me to further explore Machine Learning and A.I. techniques that truly interest me, allowing me to develop a deeper understanding of such techniques so that I will be able to apply them to many other fields. On top of exploring the new aspects of A.I. that I have not previously been exposed to, I will also be able to adopt some knowledge from previous University modules into this one task proving that I can apply what was previously taught to a new situation.

For the track the car will have to navigate, I have selected a real racetrack found in Lincolnshire (northern England) called Cadwell Park. This will ensure there is a wide variety of corner types and angles as well as some lengthy straights for the car to adapt to. Not only does it ensure a wide range of corners for the car to learn, but it will allow for a direct comparison to real cars on the same track such that we can see how similarly this A.I. performed to real experienced drivers on the same track. The car will likely not behave in precisely the same way; however, this comparison could give some extremely useful insight into the next steps of development for the car and the current successes that the car made within this project.

1.2 Aims and Objectives

My main task for this project is to create a successful autonomous vehicle that can navigate a predefined track successfully avoiding any collisions with the track itself (i.e. without hitting any walls). Building on from this base task, I want to try and create a car that is able to navigate the course at speed, meaning that the vehicle progresses to learn to accelerate and decelerate (or brake) for corners and straights, rather than the car just maintaining one constant slow speed to travel around the track.

There have been previous papers and projects exploring the simulation of an autonomous vehicle, however, most of these projects aim for the base case that I want to achieve, this being a car that can navigate around a predefined track safely without colliding with any of the environment. Other projects used a slightly different case which is navigating around in a traffic like environment meaning following road laws, signs and potentially even avoiding collisions with other motorists.

My project differs in the aspect that I want to achieve a more advanced car that as mentioned, is able to understand a path and use this knowledge to accelerate and brake accordingly to provide a faster lap, whilst also remaining safe and avoiding the environment.

I also plan to have the car 'see' in a similar way to which real autonomous vehicles 'see' using Lidar by implementing a similar method into the simulation to gather data about the environment that the car will use to navigate.

1.3 Short Summary of the Project

To summarise, I will be developing an autonomous car simulation using the latest methods in Artificial Intelligence such as Deep Neural Networks (Deep Learning) and Genetic Algorithms to help evolve the car into a safe, effective and fast autonomous vehicle which then will be comparable to a nearly identical real-world scenario to find the strengths, weaknesses and improvements needed for this A.I. system.

In a simplified bullet point list, the progress and objectives will be as follows, from the most basic and beginning of the project, to implementing the A.I. into the car:

- Create 3D model of Cadwell Park
- Create Unity project, load in the track and create a car model
- Create the car controller
- Create all of the event handling within Unity (for collisions, resetting the car and maintain knowledge of the current world state)
- Start the Python side of the project
- Add in a WebSocket to both Unity and Python
- Set up the Neural Network generation and forward pass
- Implement the Genetic Algorithm (this choice will be explained further in this report)
- Train the cars using a multitude of different network topologies
- Analyse the performance of these different topologies

Section 2: Literature Review

As part of this project, a considerable amount of research into relevant and potential methods on how to create, control and train the A.I. of the car was undertaken. Outlined in this section is a report on all my findings and how they can be applied to this area of Artificial Intelligence.

2.1 Machine Learning Methodologies; Supervised, Unsupervised, Reinforcement and Evolutionary:

In machine learning, there are numerous ways for a computer to 'learn', these main methods being Supervised and Unsupervised learning. Each of these methods has its benefits and drawbacks as well as being more applicable to certain applications that to others.

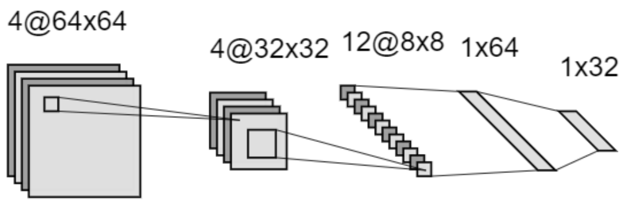
Supervised Learning:

Supervised learning is the most common type of learning in the Machine Learning space currently and as a result has many algorithms associated with it. This method relies on having knowledge of a data set provided [1] and then uses this knowledge, after making its own predictions, to adjust the parameters within the network using a concept known as back propagation [2] so that the next predictions it makes will be more in line with what is actually labelled/categorised within the data set.

We can use this method to help classify areas within an image that the car/vehicle is using to see to help defined certain objects within the image. For example, the roadway can be given as training data [3] and used to teach the machine, thus the car, where it is allowed to navigate to, as well as where within this navigable space is free from obstructions. Not only is this method useful for classification of roadways, but also pedestrian detection can be performed using this method [4].

Convolution Neural Networks (CNNs) are most typically used to process and perform the object detection and classification of these types of images. This type of network is extremely similar to the usual Neural Network architecture in that there are many layers of neurons connected by weighted edges [5] however, the convolution layers usually behave differently in how they achieve their own output values and is also mainly used to process images. These layers take the image and pass a filter over each of the pixel values in the form of a matrix to produce the output of that convolution and convolution layer. A typical filter size is 3x3, hence, this will also shrink the image size on output.

Figure 1: Convolutional Neural Network Example



Therefore, supervised learning with the aid of a CNN is a potential combination of techniques to allow a car to know where to drive and recognise objects of importance within its field of view, such as sign posts, pedestrians and other cars allowing it to then make decisions on where to navigate to.

Overall, this method of taking video from a vehicle and analysing it to perform a whole manner of operations like segmentation (finding the drivable area, as described) and object detection (such as for pedestrians) [6] is widely used in the pursuit of autonomous vehicles.

Unsupervised Learning:

Usually, unsupervised learning is used in a case where there is an extremely large volume of data provided to a system and is unlabelled or uncategorised.

Within this method there are again lots of different techniques and algorithms that can be applied to better fit the situation that it will be used in with some of the most widely known being Hierarchical Learning and Data Clustering [7].

Data Clustering [8] is perhaps one of the most popular methods of this learning type and is used to find patterns throughout a large set of unlabelled data and then use this to group (cluster) together similar points of data. One method for this type of clustering is k-means clustering. K-means clustering [9] uses a point called a centroid which upon initialisation are randomly placed within the data space then used to calculate the Euclidian distance to all other points with these points then assigned to the cluster centroid closest to them. After this, the centroid is recalculated to be the centre of the cluster created and the process repeats. The process repeats k times, hence k-means, since this method can be prone to converging in an undesirable location, or local optima, so running multiple times can help relieve this issue to gain a result that will be successful enough in a practical scenario.

These types of methods however cannot easily be applied to an autonomous driving scenario as this is mostly a data mining technique which under the circumstances of an autonomous vehicle would not aid in creating a safe and efficient self-driving machine.

Reinforcement Learning:

Reinforcement learning at its core is very similar to trial and error [34], where an agent tries many different options available to it and is given a value called a reward (calculated using a reward function) depending on how beneficial this action is to the system.

The value function considers the expected/predicted, accumulative, discounted and future reward of any action from the current state that the agent is in. Using this, one of these actions is taken. Usually, the best, or most rewarding action will be taken, which is known as a greedy option, alternatively, some reinforcement algorithms use a probability to determine whether some other action that may not be optimal is taken. This technique of choosing an action is called ε-greedy [35], where ε is the probability of some other, non-optimal action being taken. Choosing either of these options is given a name: Exploitation (picking the best, greedy option) and Exploration (not picking the greedy and as the name suggests, exploring the space).

As rewards for given actions in certain states are calculated the agent is knowledgeable enough to choose accurately the best option for its current state.

A simple example of reinforcement learning in practice is a simple robot trying to find the exit of a maze. Any action the robot takes that moves it closer to the exit gives a reward and an action further away gives none, or a smaller reward. As time progresses and the robot explores, the matrix of reward-actions is filled the best values of reward for each action in a current state. Q-learning [36] and functions within this method are used to calculate the values within this matrix and is based on some knowledge of other rewards in states that can be reached from any other state.

Evolutionary:

Evolutionary learning is a relatively new technique within the Artificial Intelligence space and at its core, takes inspiration from the concept of Darwinian evolution [10] and applies this to the Neural Networks or systems that need to be taught to perform a certain task. Because of this inspiration from nature and the natural path of evolution that life takes, a way to judge and discriminate against different individuals in a population must be conceived so that only the best can move forward to produce the next generation, thus improving the A.I.'s ability to perform the given task.

The function required to determine this performance is called a fitness function and is used to perform the fitness evaluation of each individual which then allows for the reproduction of these best individuals to take place.

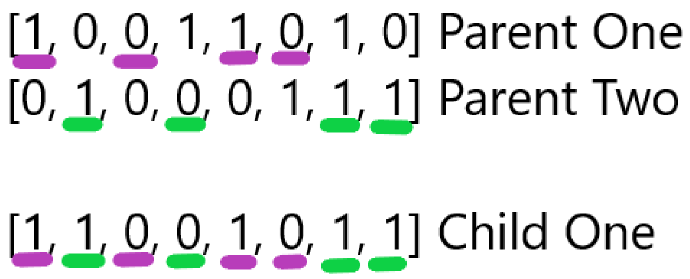
Reproduction, as the name suggests, is creating a new set of individuals (reproducing them) to populate the next generation, however this itself does come with some challenges. First, is the method to perform this reproduction and the second challenge is how to prevent the individuals from becoming too similar too quickly and thus not being able to explore a large enough search space to find acceptable results. To combat these issues, cross-over and mutation [11] can be implemented into an evolutionary algorithm to reproduce the individuals chosen during fitness evaluation.

Cross-over:

Crossover is the act of taking a gene/chromosome from one of the parents in the set that passed the fitness evaluation the best and using this to ‘split’ the two parents into two separate genome sequences. After this point is chosen the two parents swap portions to mix together their chromosomes creating this child individual in the new population.

Another way to perform cross-over is with uniform cross-over which is where single chromosomes of an individual are randomly taken from one of the parents and placed into the child individual at the same point this gene was taken from.

Figure: 2 (Showing uniform crossover)


[1, 0, 0, 1, 1, 0, 1, 0] Parent One
[0, 1, 0, 0, 0, 1, 1, 1] Parent Two
[1, 1, 0, 0, 1, 0, 1, 1] Child One

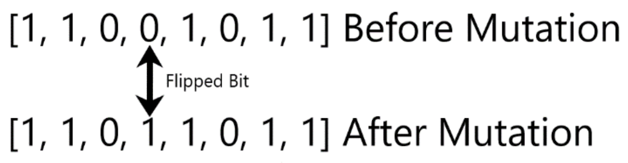
Here (Figure 1, above) you can clearly see that with uniform cross-over for the first gene in the sequence of Child One is taken from Parent One and the second in Child One is from Parent Two.

Mutation:

Mutation is used to randomly change the value in any specified slot in a genome of an induvial during the cross-over phase of reproduction to any other valid value. The rate at which mutations occur can be variable and given by a value such as 0.1 to mean a 10% chance that a mutation will occur for this individual.

For a binary individual, such as Figure 2 and Figure 3, the mutation is as simple as flipping a bit from 1 to 0, or 0 to 1. In more complex examples such as a Neural Network the value can be a replacement of a weight or bias to a new random value within the specified range that these weights and biases can exist in, for example, between 0 & 1 or -1 & 1 are some commonly used values.

Figure: 3 (Showing mutation)


[1, 1, 0, 0, 1, 0, 1, 1] Before Mutation
Flipped Bit
[1, 1, 0, 1, 1, 0, 1, 1] After Mutation

Both of the above techniques are extremely useful and relevant when using a Genetic Algorithm (GA) [12] as the main algorithm for this reproduction method.

Genetic Algorithms, as well as some other methods such as NeuroEvolution of Augmenting Topologies (NEAT) [13], will be discussed later in this paper's literature review (Section 2.3).

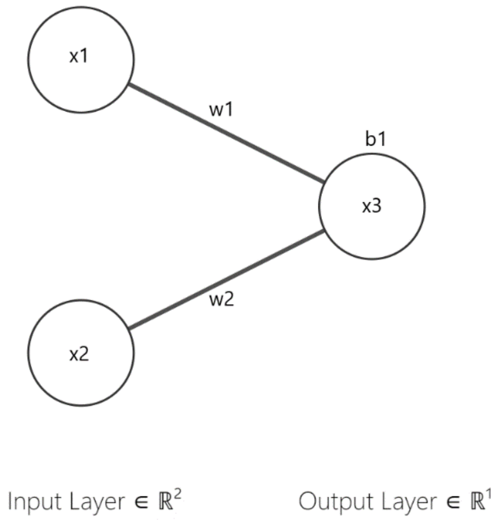
2.2 Neural Networks and Deep Learning:

Neural Networks (NN) [14] are the core component of many A.I. applications and take inspiration from the neurons within the human (or other mammalian) brains. This means that there are neurons that take input from other neurons and depending on the signal provided, produce some sort of output that is either strong enough to trigger an action or is too weak to do anything.

How is this concept applied to computation?

To digitise this concept of a neuron, the first thing to be done is find a neat way to represent this neuron in a computer system. The representation of this neuron consists of the value of the node (given as an input to the network, or an input from a previous layer of nodes) and a set of weights and biases, used to signify the strength of the node (or neuron) similarly to how connections in the human brain can be strong and weak. The larger the weight, the more likely a node is to trigger an action within the network since the output of a node is a function of the inputs, weights and biases.

Figure: 4 (Simple Neural Network (Visualisation created using: http://alexlenail.me/NN-SVG/index.html))



In Figure 3 above, an extremely basic NN topology is provided to help visualise how these weights and biases link the inputs to outputs. Just two inputs, x1 and x2 are connected to an output node, x3 and both of these connections have a random weight (when initially assigned) w1 and w2 with the output node x3 also having a bias attached to it, b1. Together, using the input values, weights and bias the value of the output can be calculated. The formula for this is single node as follows:

$$x_{kn} = \left(\sum_{k=0}^{i} x_i * w_i \right) + b_{kn}$$

This is repeated for all nodes in a given layer, Figure 3 only needs this computed once for the single output, but if there were more outputs, or this was a hidden layer with multiple nodes the calculation may need to be computed *N* many times for the size of the new layer.

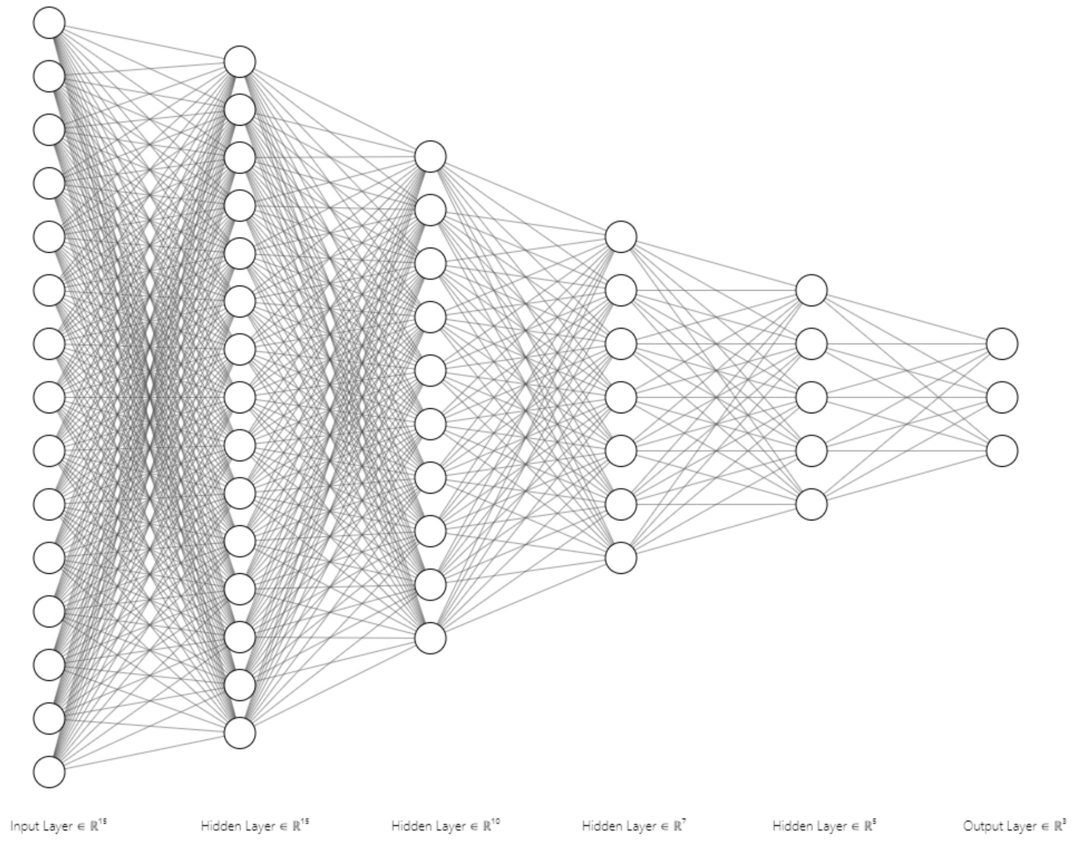
Training a Neural Network can involve many different techniques, with the main method being back propagation supervised learning.

Back propagation is performed after a forward pass of a given network has been completed and then the predicted results are compared to the actual results in the training data. At the core of this method is the concept of ‘loss’ which is essentially the error rate of the network during its forward pass [15] and this is then used to ‘back propagate’ across the network updating the weights using the root mean squared error as it travels back through (other functions/formulas can also be used to back propagate and calculate new values). This means that the next time the forward pass is performed all of the weights will produce a slightly more accurate prediction since this loss alters them slightly in the correct direction of the learning gradient (an increase or decrease).

Another method of training a Neural Network is using the Evolutionary Learning methods mentioned in section 2.1, with one common evolutionary algorithm being Genetic Algorithms, which will be explained in section 2.3 of this report.

Most NN’s are much more complicated than Figure 3 shows and tend to look more like Figure 4 (below), with many more layers creating a Deep Neural Network.

Figure: 5 (Complex Neural Network)

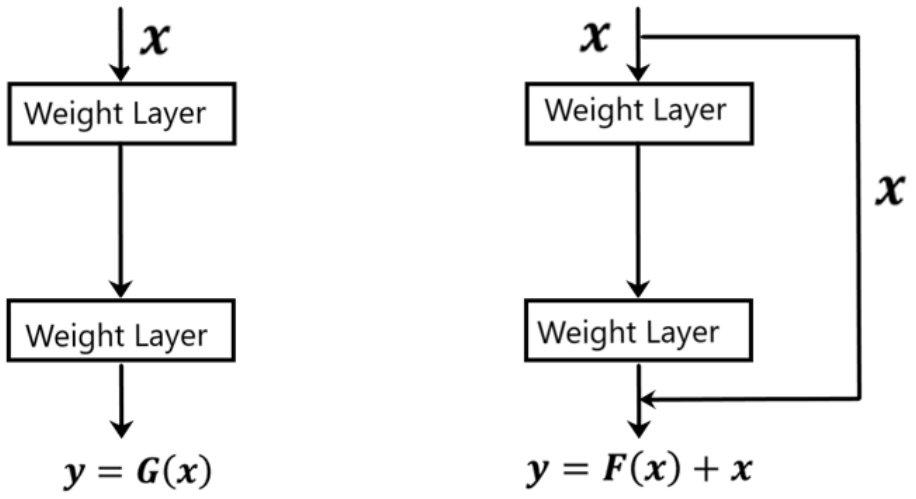

Input Layer $\in \mathbb{R}^{15}$
Hidden Layer $\in \mathbb{R}^{15}$
Hidden Layer $\in \mathbb{R}^{10}$
Hidden Layer $\in \mathbb{R}^{7}$
Hidden Layer $\in \mathbb{R}^{5}$
Output Layer $\in \mathbb{R}^{3}$

Deep Neural Networks (DNN) and Deep Learning are still in essence extremely similar to usual Neural Networks and learning; however, they operate on a much larger scale. DNNs are architecturally the same as a normal Neural Network however, they have many more hidden layers between the input and output. For example, a normal Neural Network can have 0 or 1 hidden layers and if there are 2 or more hidden layers, this then constitutes as a Deep Neural Network. Thanks to the extended network of neurons, a Deep Neural Network is much more able to adapt to more complex tasks and learn much more intricate or specific scenarios by tuning all of its parameters. This explains the popularity of Deep Learning as it can be applied to almost any problem and given the correct training data, a DNN could produce extremely accurate results, sometimes much better than humans in some tasks.

With this larger topology of Neural Networks, the usual training method of back propagation does not always work as the loss can diminish as it travels further up the network, thus is not able to alter the weights correctly. These networks can also (if the correct training method is used) be overfit if there is not enough data to train the network, meaning that the size of a network can also be slightly dependent on the data provided to it.

To solve this loss training problem, a solution called a residue block can be implemented to ensure the strength of the signal/loss is not lost. In simple terms, this allows the signal to be passed back along a network separately from the signal/loss that is also being used to process the changes needed to the weights.

Figure 6: (Residue Block)



Employing this method allows for these larger networks to be trained effectively and make proper use of their size and capabilities with more layers and nodes. An example of a network that operates this way is the ResNet [16] architecture.

2.3 Genetic Algorithms (and NEAT):

2.3.1 Genetic Algorithms

Genetic Algorithms (GA) [17] are a popular form of evolutionary learning that as mentioned in section 2.1 take direct inspiration from Darwinian evolution and apply this technique to how the network is modified and evolves to create a better performing Neural Network. This algorithmic technique takes all of the key concepts from evolutionary learning and applies them, such as the cross-over and mutation functions.

GAs use chromosomes to represent the attributes within a network and apply genetic operators to help guide the population of individuals into the correct direction to reach a global optimum [18]. Parameters are initialised to be completely random and for a Neural Network this would mean initialising a random set of weights and biases to fit the size of the network that is desired, with each of these values being the genes or chromosomes to be used within the GA.

Once the population have been tested and run, each will have gained a fitness value calculated by their performance within the given task, some examples of parameters to factor in while creating the fitness function could be the accuracy of the output or time taken to perform the task at hand. Subsequently, the parents of the next generation are selected using these fitness values, with a higher value being the better performing individuals, therefore giving them a higher chance of being picked for the cross-over. Alternatively, the top 2 or 3 performing individuals can be picked, removing the chance element of the selection process, ensuring the best genes are carried into the next generation.

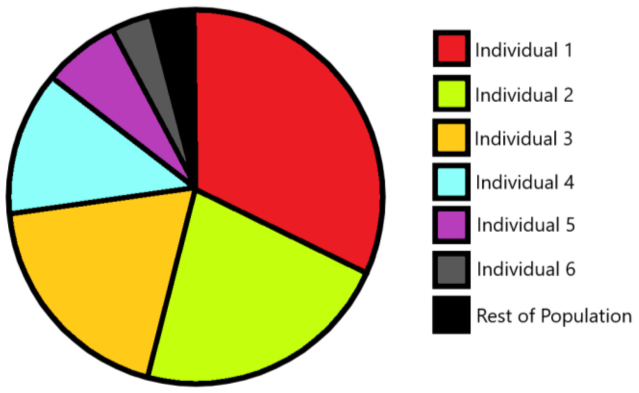
An explanation on selection methods:

As mentioned, there are multiple ways to select the parents for the next generation of individuals, each providing a different twist on how they are selected and thus producing different and diverse populations.

Roulette wheel selection [19]:

Roulette wheel uses a chance selection meaning that it is not guaranteed to always pick the best parents for the next generation, however it does keep a more truly random aspect to how the population evolves over time. The better an individual performed the more likely it will be to be picked (it takes a larger portion of the 'pie'), since these percentages are taken from the fitness of the individual against all others.

Figure 7: Roulette Wheel Selection



Elitism [19]:

Elitism, as the name suggests, picks the best individuals and chromosomes and uses these in the new population to ensure that the best genes are always carried through into the next generation. Since the best genes are always carried over, this means that the GA will evolve extremely rapidly allowing for fast convergence on a solution.

Rank Selection [19]:

Ranking takes the fitness of the individuals and sorts them in order of best to worst (best having the highest fitness score, worst the lowest) and then uses these rankings to pick out the individual to use. For example, the worst may be rank 1 and best rank *N* with *N* being the number of individuals in the population. Then the probability that an individual is chosen is proportional to its rank within the population so that no one individual will have an 80% or 90% chance of being picked. A formula for this proportional probability is:

$$p = \frac{r}{\sum_i^N r_i} x100$$

Where 'r' is the rank of the individual (a number value between 1 to *N*).

In a pseudo-code format this could look something like:

Figure 8: High level pseudo-code of a Genetic Algorithm

```
initialize individuals with random weights
while min performance NOT met:
    for each individual in the population:
        perform the task
        evaluate performance
        save fitness value to the individual
    end for
    select best individuals from population
    for newIndividual in newGeneration:
        select crossover point
        swap genes and save as newIndividual
        mutate one chromosome with x random chance
    end for
end while
```

Figure 6 above uses the single point cross-over technique, but this simple pseudo-code algorithm can be easy adapted to use any cross-over technique, such as uniform cross-over described in section 2.1, or two-point cross-over.

The GA technique can be applied to a multitude of applications, with a good application of this technique being optimisation problems [20]. GAs are also preferred for these types of optimization problems over the more well known for optimization problems, such as heuristics since the GA can scale with the problem size in a much nicer way, the scaling is more linear than the alternative heuristic method (this is for larger NP problems) [20].

GAs can also be used in the field of autonomous driving [21] [22] [30], which will be the most relevant for this project.

Section 2.3.2: Exploring NeuroEvolution of Augmenting Topologies:

NeuroEvolution of Augmenting Topologies (NEAT) [23] is an extension of Topology and Weight Evolving Artificial Neural Networks TWEANNs (which is an evolution of sorts from a GA) and operate in a very similar way, however, have one large key difference during the reproduction and mutation phase.

History markers [24] are one of these additions that drastically increase the capabilities but also the complexities of this algorithm. In essence, history markers keep track of the ancestors of different individuals preventing a matching during cross-over that is incompatible, meaning that two individuals may have evolved from a different ancestor and therefore contain different genes (a different topology) resulting in the loss of one of these extra genes during the cross-over phase. These markings ensure that two individuals are mated (crossed over) with each other if they have the same ancestry, preventing the loss of any of these newly evolved genes which could lead to a better overall solution. Since, cross-over with an individual of a different species can lose an adaptation unique to the individuals within this species, leading to the loss of a mutation that could be drastically beneficial to the network overall.

Leading on from the historical markers is the addition of speciation to the algorithm. Speciation is a method of keeping similar individuals associated with each other my grouping them within the same species, hence speciation. This technique is useful because when a new gene is added to the network, this can actually have a negative effect of the performance and the fitness of the individual to perform the given task, but with speciation different genes can be placed into a separate category. Individuals then compete within their own species category rather than with the global population, allowing it to develop and improve before being placed against the entirety of the population.

In order to produce these different species, the NEAT algorithm has to able to add/create them at some point. This is performed in the same way to normal weight value mutations and also performed at the same time as these more basic mutations of the network. Mutations with NEAT can also add whole new genes/chromosomes (weight nodes) or new connections between existing genes. This is how the new species of networks evolve and why the need for the historical markers and speciation is so integral to NEAT to keep the network improving and exploring as many different avenues as possible.

Figure 9: NEAT adding a new gene (node)

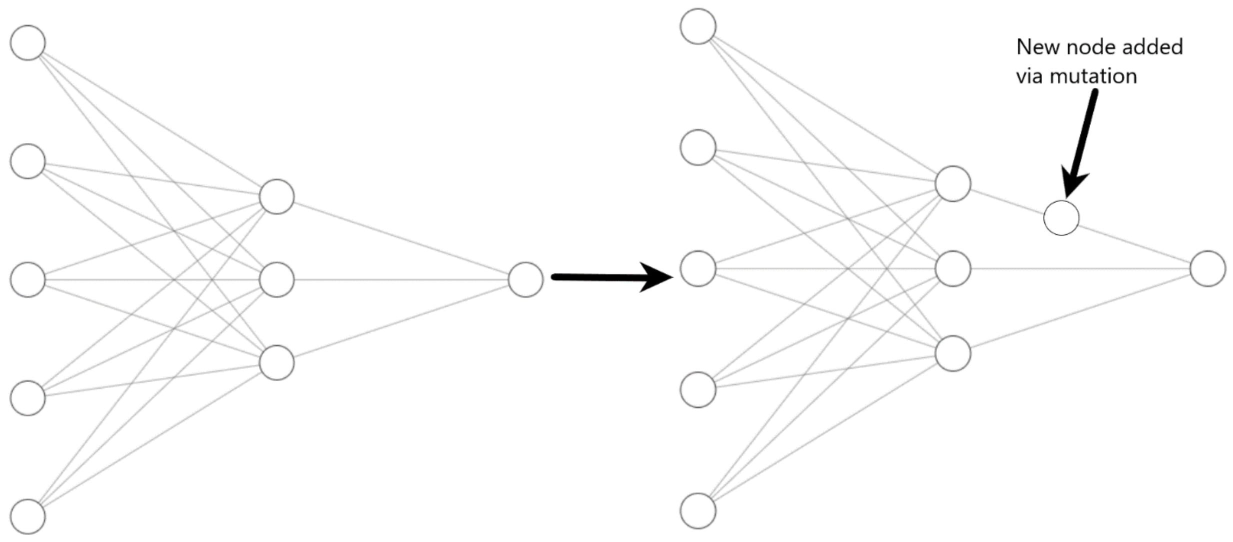
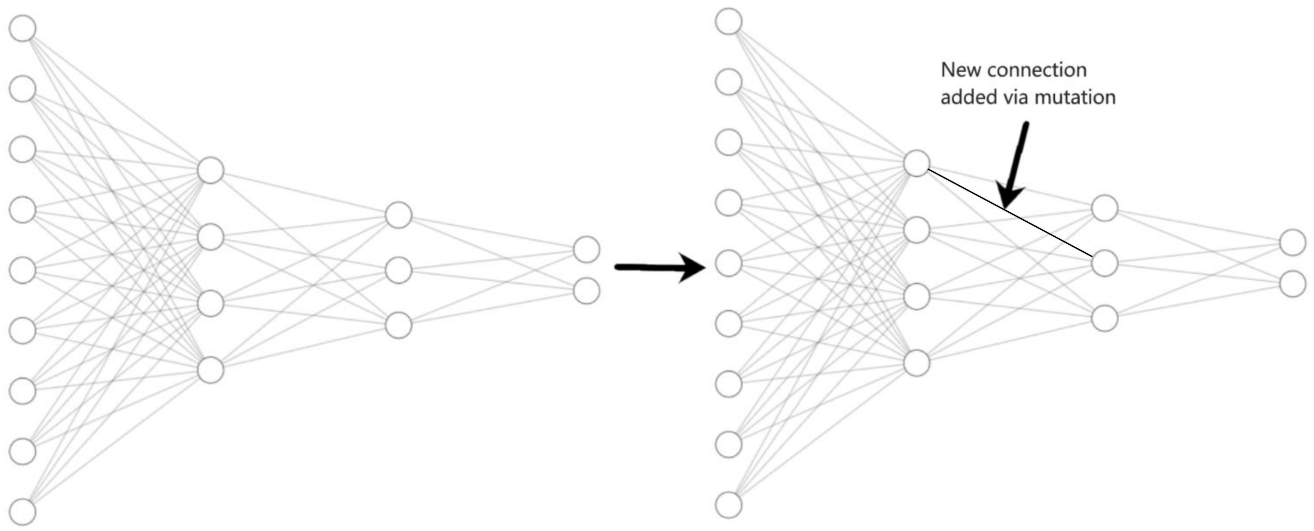


Figure 10: NEAT adding new connection between genes (nodes)



Initialization of the networks in a NEAT algorithm also start as extremely basic [25] and the more complex forms of the network only come as a result of running the algorithm itself and not a pre-set network topology as would be designed in a normal Neural Network or the more basic Genetic Algorithm.

NEAT can also be extended to deep learning [26], but rather than treating each node as a chromosome, each individual layer is treated as its own chromosome. To translate a DeepNEAT into a normal Deep Neural Network each chromosome in the NEAT algorithm is translated into its respective layer using a set of parameters to do so. A set of hyperparameters are also stored about each chromosome that can then be applied to the entire network, with some of these hyperparameters being the learning rate of the network/algorithm.

2.4 Methods for Automated Vehicles:

Automated vehicles have rapidly grown in popularity and acceptance among businesses and the public over the past decade thanks to some companies like Tesla commercialising this new technology and putting it as an option in their own cars.

There are 5 levels to vehicle automation, each with a different amount of computer input compared to human driver input into the vehicle control [27].

Figure 11: Table of automation levels

Level of Driving Automation	Description of the artificial control
1: Driving assists	Automation provides slight aids, such as lateral assistance (lane assist), or longitudinal assistance to drivers
2: Partial automation	Automation performs **both** lateral and longitudinal assistance to drivers
3: Conditional automation	Automation performs Dynamic Driving Tasks (DDT)
4: High automation	Automation performs DDT and DDT fallback
5: Full automation	Automation performed full DDT with DDT fallback as well as Operational Design Domain (ODD)

From [27] SAE 2016 states that DDT is these subtasks: 1) Lateral vehicle motion, 2) longitudinal vehicle motion. 3) monitor the driving environment using object detection and event detection, recognition, classification and response preparation, 4) object and event response execution, 5) manoeuvre planning.

Autonomous vehicles need to be able to ‘see’ the environment they are in, so that they can make informed decisions about where to go, where is free of other traffic, where other cars are on the road and if there is an obstruction or pedestrian in the way. To do this, some different vision techniques can be applied to a car either combined, or individually, with the two main systems for a cars vision being LIDAR and Cameras [28]. Both of these methods have their strengths, for example, the cameras on an autonomous car can be used to detect objects within the environments, which for use on public roads is extremely important so that a car can see road signs or traffic lights and act accordingly, as well as react to other cars or pedestrians moving around. LIDAR sensors are useful for calculating 3D point distances within the environment, therefore are able to give the car the special awareness it will need to navigate safely around any obstacles, as well as plan paths correctly, by knowing how far a certain object (even if it does not know what it is just by lidar data) is and how fast it is going. Using this data, localization [29] can also be performed. Some other information will also be used, such as GPS data for route planning to guide the car as it navigates around these obstacles and to its destination.

These methods are used in conjunction for self-driving cars to be used on public roads due to the complexity of the challenge of driving on a road with other road users and a potentially unlimited number of uncertainties that come with this activity. The cameras object detection capabilities can be used not only to detect certain objects like pedestrians, as mentioned, but also to detect the roadway to drive on. This can then be used in path planning for the vehicle, that can then choose one of these paths to follow that could be either; least obstructed, fastest lane for a corner or just keeping the vehicle in a safe spot on the road, while also following the route to the destination.

Methods for creating autonomous vehicles:

Deep learning is often used in autonomous vehicles [30] due to their complexity and the extra complexity of a deep network and deep learning caters for this well. The task of controlling a vehicle can be broken down into two main tasks which are lateral vehicle control (i.e. steering, turning the vehicle) and longitudinal vehicle control (i.e. accelerating and braking the vehicle). Deep learning is paired with the images captured by the car, which are then passed into a Neural Network, most likely in this case for image and object detecting a Convolutional Neural Network (CNN).This method of learning for a self-driving car usually utilises the supervised learning method, given that you can have labelled images of pedestrians, the road and other cars fed to the CNN which can then learn to recognise all of these different objects. Using this method even calculations of steering angles can be made from the detection of road and pathways. This means that, even though the A.I. and Neural Network is mostly trained on just recognizing images, it can be expanded to allow for accurate vehicle controls to be determined through what it finds within an image, whether that be braking for a traffic light, or turning to avoid a collision, to accelerating onto a motorway.

CNNs are one of the most popular deep learning methods for autonomous vehicles with already known CNNs, such as GoogLeNet [31], being used to recognize all objects within an image for the car.

However, supervised deep learning requires immense amounts of data to accurately train the networks involved to give reliable and acceptable outputs [30], which can be difficult to come by for the autonomous driving area.

Evolutionary learning (NeuroEvolution) is also one method that can be applied to an autonomous vehicle [32]. To create an autonomous vehicle using this method, real life applications cannot be used during any sort of training phase, as this technique starts off not ‘knowing’ anything. Therefore, this would not be safe to use on real roads, or tracks. Due to this, a simulation is used to first train the A.I..

GAs for autonomous driving can be used in both an ‘empty’ [32] road simulation and a ‘busy’ road [33] simulation, proving that this technique can be applied in a range of complexities within the self-driving area and with some further development, these simulations could be expanded to control real vehicles on public roads safely. This method can however lack some of the precision needed for a truly safe real road autonomous car, meaning that it is the best method for creating a baseline to fine tune and increase the robustness from. An autonomous car created from a GA learns the basics of collision avoidance and car manoeuvrability much faster than any other method.

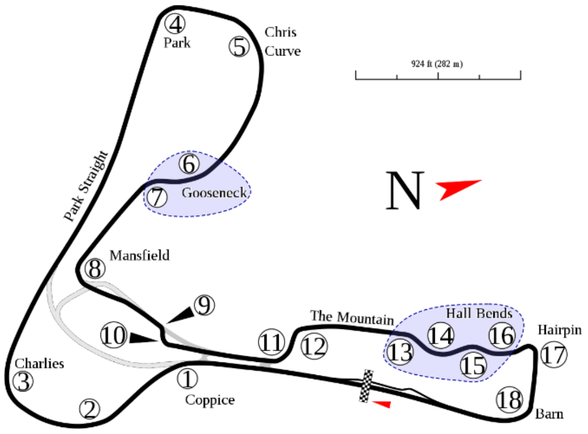
NeuroEvolution and GAs also benefit from not needed hundreds of thousands of data to learn from, as they learn and evolve from their own experiences by choosing the two best performing individuals, meaning for an autonomous driving task, a GA can be extremely helpful if there is limited data to develop and learn from. The GA can learn the basics of the task through its own evolution and then this agent can be further tweaked by any extra data and techniques to ensure a robust and safe operating vehicle is created.

Section 3: Methodology used

In my own autonomous car simulation, I will be using the following two key technical A.I. based components: Deep Neural Networks (DNN) and a Genetic Algorithm to evolve and train the cars.

Due to the complexities of the task that I am training the car to overcome (despite it being a relatively simple autonomous car, the task is still complex) a Deep Neural Network most likely will be required. A DNN will be able to tune more parameters within the network itself, meaning that the car can adapt to more stimuli as well as make more finely tuned adjustments, for example, whilst turning a corner, the DNN would be able to better adapt to a change in angle, or speed if it were moving too fast. The nature of the track it will be learning means that this adaptability is crucial as Cadwell Park contains a large variety of corners, angles of said corners and straights.

Figure 12: Cadwell Park Outline



Using this in combination with a Genetic Algorithm will also mean that the solution for training the DNN will be much simpler and will not require the implementation of a residue block to store any ‘loss’ value. No ‘loss’ value will be generated as there will be no back-propagation through the network. However, the complexities arise when trying to find an adequate fitness function to evaluate the performance of each car as it attempts to navigate and drive around the track.

The reason I will be implementing these two techniques for the autonomous car is because as stated, autonomous vehicles are a complex form of Artificial Intelligence, so will benefit greatly from the extra flexibility that the DNN will offer, over a basic Neural Network with no hidden or only one hidden layer.

As part of the project, I will be exploring how different complexities and sizes of networks affect the car, whilst also trying to find the best possible topology for the network to be to control the car in the best way possible, as will be judged by the fitness function.

I also want to explore the area of Evolutionary Learning and implementing a Genetic Algorithm will be the perfect way to discover this, new to me, area of Machine Learning and A.I. and a relatively new development in Artificial Intelligence as a whole. On top of this, while there were some autonomous vehicle papers that used Evolutionary Learning and Genetic Algorithms to develop their own car simulation and some other examples I managed to find online using Genetic Algorithms or NEAT to create an autonomous vehicle, in comparison to some other methods Evolutionary Learning has been used less. This could be because of the novelty of Evolutionary Learning and/or that supervised learning, using cameras to detect object within a vehicles field of view has already been established as a robust method for creating reliable autonomous vehicles.

Despite this, I still want to explore the potential of Evolutionary Learning with autonomous vehicles in order to better understand Evolutionary Learning as a whole and how to adapt this technique to be used in a lesser explored field and contribute further to the knowledge already found for using this technique for this application.

One more reason for this evolutionary approach, is because there is no data available that I can use to train a network that would match the environment the car will be driving in. This means that it will not be possible to explore a supervised learning method, unless I drove the car around the track in Unity hundreds, if not thousands of times myself, but this is not practical to do to collect data. Therefore, the lack of training data, means that Evolutionary fits well here as it learns from its own experience, rather than others' own experiences.

The more basic Genetic Algorithm will be sufficient for this task when compared to the NEAT evolutionary algorithm, since the network, despite being a DNN, will not need to be immensely complex. Within this GA, I will be using uniform cross-over to create the next generation of cars from the parents, which will start out as just the two best (using elitism) of the current generation to be these parents.

The fitness function this algorithm will use to calculate each cars performance will first start out as:

$$f = \frac{distance}{time}$$

This is more commonly known as the average speed. It will be the first fitness function I try (but subject to change depending on its effectiveness on producing successful cars) because I want to try and create a car that not only can navigate around the course successfully, but also do so in a fast manner, trying to as closely as possible, replicate a real drive around the track.

To implement these techniques, I will be using a combination of software and programming languages.

I will be using Unity3D to host the simulation of the track and vehicle and Blender to create the 3D model of Cadwell Park.

Unity is a free game development software that allows for easy creation of game like applications, which will be similar to what I will be making with my simulation. It also, due to its popularity, has lots of support online through its own documentation and community support, in terms of posts on forums and online video tutorials. Since I do not have extensive familiarity with the language used (C#) this will be very helpful to me. I also want to expand my knowledge base of programming languages so using C# is a bonus for me to learn a new programming language.

For the A.I. and GA I will be implementing, I have chosen to create this outside of the Unity code space and implement it with Python. The reasoning for this is that I am already very familiar with Python thanks to multiple modules that have used this language throughout university and with many modules using Python for Artificial Intelligence applications. I can therefore transfer knowledge I have gained from other modules into this final project to show my development and learning over the past years.

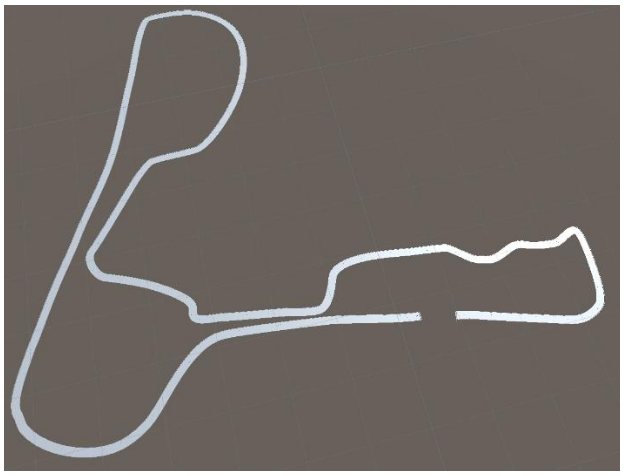
Section 4: Implementation

4.1 Unity

On a more detailed level, within the Unity project I will import the 3D model of Cadwell Park from Blender that I will create myself as well as all the distance gates that will be used to measure how far the car has travelled. Once imported, I can label them all as ‘Track’ for the track and ‘Marker’ for the distance markers/gates. Using these tags within Unity, I can calculate when a collision occurs with the track walls and when the car passes through a distance gate. Upon a collision with the track walls, the car will be reset to its original position and all physics acting upon it reset, so that the next car is able to start off in the same way as all of the other cars, creating a fair environment for each vehicle to compete in. To reset the physics in Unity I set the attribute ‘*isKinematic*’ to **true** when a collision occurs, then **false** after the car had been reset. This attribute is a value a Rigidbody object contains that allows or disallows physics operations to be performed on this object. When true, no physics forces are calculated or affect the object and when false, physics forces acting upon the object are calculated and applied, hence why turning this on then off during a reset resets all physics currently acting on the car.

A Rigidbody is a class of object that allows the Unity physics engine to act upon it, when the correct parameters are set.

Figure 13: Cadwell in Unity (View from above)



As shown, this is a very close recreation of the real track, minus the elevation of the track (this is a flat replica, without any hills).

For the car itself, I found a free 3D model of a Toyota GT86 on the Unity asset store that I then downloaded and imported into the project. This car used WheelColliders mounted onto the wheel objects to control its movement. With these components, I can create a more accurate representation of how a car will act, although they do require more knowledge to control, rather than some other ways of moving a Rigidbody object.

With the wheel colliders in place, I can control a number of aspects of the car such as the suspension stiffness and travel, grip/friction each wheel has before traction is broken, both longitudinally and laterally, (forwards/backwards and sideways (left/right)) and even the mass of the wheel itself. Within the code and setup of these objects, I can also set the values for the 'engine' power and 'brake' power, that when applied to the colliders propel the car in a more realistic manner. Most values of the WheelColliders themselves are default, with a slight increase in the slip values for friction (in the forwards direction) giving more traction to propel the car and an increase in lateral stiffness of the car for turning.

Given that the car's weight is set at 1000Kg (1 tonne), I have given the following constants the values of:

- MAX_STEEL_ANGLE = 45 (45-degree maximum turn angle for the front wheels)
- ENGINE_POWER = 1500 (maximum power available to the rear wheels (since this car is rear wheel drive))
- FRONT_BRAKE_POWER = 1000 (maximum brake power of the front wheels brakes)
- REAR_BRAKE_POWER = 750 (maximum brake force of the rear wheel brakes)

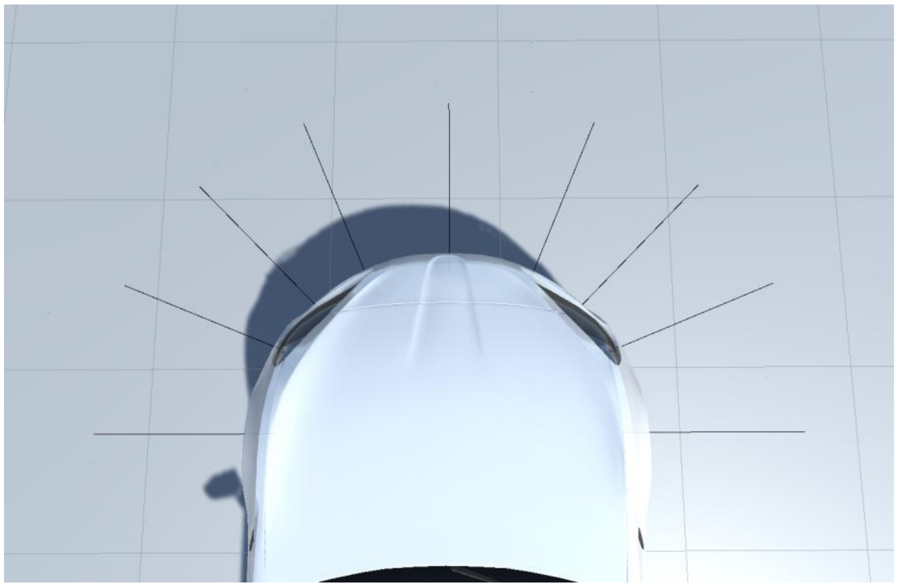
These values are all set as constants via the scripts (programmatically, in C#) rather than in the unity editor itself, to ensure that they are not altered accidentally.

The front wheels have a greater braking force to mimic how cars are designed, with a larger set of brakes for the front wheels, since this is where most of the braking is done, due to the weight transfer to the front wheels during the braking process. The control of the direct components on the car is performed in the '*CarController.cs*' file of the project.

To control all of these components, I created a '*WorldState*' ('*WorldState.cs*') script to keep knowledge of the environment and the car, such as the speed of the car, distance travelled, as well as the starting position and rotation of the car for when it needs to be reset.

Attached to the car's Rigidbody are a set of 9 cubes (that are invisible) that each shoot out a ray in a given direction related to the direction the car is facing. These are the rays used to imitate LIDAR that is often used in real autonomous vehicle situations and will be measuring the distance from each of these cubes (ray origin points) to the nearest first collision point with the track.

Figure 14: Visualisation of rays shot from car



In conjunction with the speed of the car, this will total the 10 inputs into the DNN.

In order to connect the Unity simulation to the Python A.I., I first thought of using HTTP requests, since this is an easy way to send and receive data between two applications. Both Unity and Python have modules that can assist with this technique as well, ensure a simple implementation into the project. However, this method can sometimes be slow, which for a real time application and simulation such as this could cause performance issues.

In the end I decided to proceed with a WebSocket. A WebSocket will be able to transfer between the two applications faster than traditional HTTP since a connection, once started, is always open between the two applications and there is no 'sender and receiver' like in the HTTP method. Both software and languages I am using have support for WebSockets natively, without needing to instal any third-party packages. This makes the implementation a little more complex than standard HTTP client-server architecture due to the nature of WebSockets but is still completely possible with a lot of documentation for both Unity and Python in this area.

To use the WebSocket, I also had to devise a method to determine the end of a data stream sequence, since not all of the data was always sent in one packet. Therefore, I had to use delimiters at the end of each data object that was sent to and from Unity and Python. The data sent was in a JSON (JavaScript Object Notation) object format, which also helped make it easy to determine the start and end of the actual data sent. I used this notation with a delimiter of '|' to indicate the end of the object and when this was encountered, processing of the data on either end could begin, with any extra data being ignored. Using this allowed me to ensure that the correct amount of data would be used without any missing, or superfluous items.

4.2 Python

Constructing the Neural Network and Genetic Algorithm were all performed in Python thanks to the large support online as well as some extremely helpful packages that can be installed and imported to help with the processing, calculations and creating the networks. I only used the package NumPy to aid with the Neural Network aspect of this project, since, if I used more packages to create and control the evolution of the Neural Networks, such as PyTorch, then I would not have had the total control that I had creating it all myself, as I did for this project. I did this to fully understand the A.I. and how to implement and use it, as well as being able to easily view the saved files of the network values created.

To create these networks, I specified the size of each layer and created a NumPy array to this specification to match each of the weights and biases that would be needed for each layer-to-layer connection. For example, a layer of size 5 connected to a layer of size 3, would require 15 weights to connect them all up, into a fully connected layer. I then saved all of these networks to JSON files to easily visualize the network layout and keep the fitness of each car within the car's own network file.

While performing the forward pass of the network, I first needed to normalize the data given from Unity passed via the WebSocket. While testing the car controller, I was able to find theoretical maximum distances each of the rays could detect, as well as the maximum speed of the car, so I used these to normalise the data to between 0 and 1 so that there would not be any value that overpowered the system. I used Min-Max normalisation to normalise the data, for example: $speed = \frac{currentSpeed - minSpeed}{maxSpeed - minSpeed}$.

Once the data had been normalised within the appropriate scale, it was sent through the network. To calculate the outputs of each of the nodes and layers I used the NumPy function '*np.sum*' on the combined array of weights and inputs into the layer (for layer one, these are the inputs from the car) by using the '*np.array*' function. Using NumPy to help with the calculations ensures that during run time the calculations are performed extremely quickly, allowing the simulation to run in real time. Evidence of this is the frame rate of the simulation in Unity that was able to reach over 150 and close to 200 Frames Per Second at points.

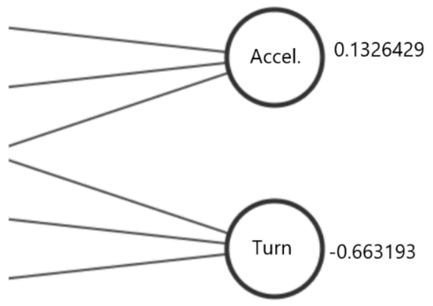
The outputs of each node were also passed through a modified sigmoid function.

$$2\left(\frac{1}{1+e^{-result}} - 0.5\right)$$

This modified sigmoid allows valued between -1 and 1, where any value below 0 is braking, or turn left and above is accelerate or turn right.

The **final output** of the network is **two nodes**, each with a value being between **-1 and 1**. The first node supplies the value to the accelerator/brake and the second node to the turn value. This is then sent back to Unity via the WebSocket where they are then fed to the car controller. For example, the acceleration would be the $enginePower * networkSpeedOutputValue$ (brake power is triggered if the output is less than 0). So, if the output is 0.5 from the network, this will equate to half throttle and -0.5, is half brake power. The same applies for the turn angle, which multiplies the output value from the network by the maximum turn angle. For example, a value of -0.25 from the network is -0.25*45 (the maximum turn angle) and would result in a left turn, a positive output of the network for the turn value is a turn to the right.

Figure 15: Output of the Neural Network



This would indicate a left turn of moderate angle and a slight acceleration.

After a car crashes, the fitness is determined using the fitness function which is the saved to the same car file the network is saved in.

Upon completing a generation, the Genetic Algorithm is run. To perform uniform cross-over, the two parents already determined by their fitness are taken and on a random 50/50 chance (so equal between the two) the current gene for this child individual is picked out from either parent. This is then repeated for every gene/chromosome in the individual (i.e. each weight and bias within the network). Using uniform cross-over will allow for a better mixture of the two parents into the next generation, resulting in a better exploration of the space.

For mutation, I have given a random chance of 1% to each gene such that if this probability is hit, the new gene placed into the child is not of the two parents but a new random weight between whatever the range is that has been specified, which to start will be a common -1 to 1 range – subject to change.

During this time, I also save away the best overall car from the current test iteration. I do this by comparing the best parent's fitness to the current saved best overall car, which is initialised to '*-infinity*'. If this parent has a higher fitness than the current best, it then replaces and becomes the current best. The reason for this is so that I can easily see how the cars are progressing and whether they are becoming stuck in a local optimum therefore not managing the beat the current best. Further, saving this car separately enables me to replay it later, since I may change the current network topology and other functions to explore other avenues for the cars, to see if they can produce better results.

Finally, the generation number is updated so that it is easy to track how fast a car may be progressing, i.e. is it managing to navigate the course within only a few generations (faster), or a large number or generations (slower)?

Section 5: Results

Videos used to showcase behavior and comparisons:

Video 1 [referenced as V1]: https://www.youtube.com/watch?v=NRBrf8UH1Sg (Simulation Evidence)

Video 2 [referenced as V2]: https://www.youtube.com/watch?v=WKem0OjArJs (Real World Comparison)

These will be referred to during this results section to aid in understanding, as well as the use of pictures of the car.

5.1 Observations of the simulation

Iteration 1:

Initially, I started with a network of topology 10, 7, 5, 3 for the hidden layers. A size of 10 and 2 are constant for the actual network inputs from Unity and the outputs for the car, these will not be changing as there will only be 2 outputs possible to control the car and 10 points of data used in the input.

Due to this being the first time running the whole A.I. process, including all of the forward pass of the network and the GA, there were some bugs that did unfortunately reflect in the results, by not allowing the car to behave correctly.

The main issue found, which was corrected for subsequent attempts/tests was that some outputs of the hidden layers did not include the altered sigmoid function, but just the usual sigmoid giving an output of between 0 and 1. This meant that as the values propagated through the network, any ability to turn negative was removed, preventing the car from being able brake, or turn to the left. This is a crucial point that the car needed to learn since the first corner it will encounter is a left-hand corner (actually called Coppice on the real-life track).

[V1]: https://youtu.be/NRBrf8UH1Sg?t=1

As stated, unfortunately this instance was not applicable for further comparisons, as there was a major bug preventing the cars from operating correctly.

Iteration 2:

Upon fixing this bug, I also decided to start with a simpler network and build back up to more complex Deep Neural Networks. So, for this first bug-free run, I used just a basic Neural Network with only 1 hidden layer of 10 nodes. My reason for trying out a non DNN topology is due to seeing some online examples using a simple network such as this for their instances of autonomous vehicles. It should be noted however, that these networks soon returned to multiple hidden layers (a DNN).

I also increased the range the weights can occupy to -5 and 5, since even though the cars did not behave correctly, the movements they did make were very small. For example, they accelerated slowly and only gradually turned right, instead of being able to accelerate at a faster rate or turn a sharp right to avoid walls.

Improvements were visible extremely quickly within this test, as by generation 3 it was able to turn the first few corners.

[V1]: https://youtu.be/NRBrf8UH1Sg?t=11

However, the car was very unstable, meaning it slalomed from side to side in extreme amounts while driving forward. This is likely due to the change in range for weights, now being much larger values, the outputs given will also be much more extreme leading to this erratic behavior of the car. These extreme movements also led to a lot of understeer from the car while turning. Understeer is when the wheels are turned in an attempt to make a corner but lose any grip and traction they may have due to too much speed entering a corner, or too much angle for the speed, such that the car continues to move fairly straight even with the wheels turned.

[V1]: https://youtu.be/NRBrf8UH1Sg?t=45

The car did fail to improve however much after this point There was a very slight improvement in the stability of the car, but not to any adequate condition and there was little to no improvement on the fitness and distance travelled by the car by generation 11.To test my reasoning for the extreme wiggle the car developed in the previous test, I ran the same network topology (the small, simple, single layer of 10) but with the weight ranges reduced to between -1 and 1 again. Upon running the evolution of the car, it was immediately apparent that this was indeed the reason for the erratic behavior of the car, proving my theory/hypothesis correct. The car was very stable, albeit extremely slow since the range was not large enough to allow for outputs in the range to create desirable movements for the car, but this experiment did allow for a better understanding on how these value ranges affected the control of the car showing that I needed to find a suitable range for these weights. By generation 11 again however, the car was not able to adapt to a sharper 90-degree right-hand corner (corner 4 called Park), being stuck here for lots of generations.

Iteration 3:

Moving on from this, I implemented a DNN with layer sizes of 8 and 4 between the input and output. In order to better understand these changes, I did not adjust the weights range, only the topology since changing one thing at a time whilst starting out helps identify how each parameter of the network affects the total outcome. However, I had planned to increase the range of weight values to -2 and 2 within this test. Doing both of these even with the network change unfortunately did not allow the car to make it past the same 90-degree right-hand bend.

In an attempt to remedy this, I again slightly increased the range of weights to -2.5 and 2.5 and a few observations were evident from this change (even if the end result was the same, stuck at the right-hand bend). Initially, some cars developed the ability to understand when a wall was rapidly approaching and begin to brake, slowing the car down. However, instead of braking to slow the car safely for the corner, the cars slowed to a complete stop, meaning that they were reset, as a stopped car is treated as a crashed car otherwise the simulation would be stuck on this stopped car. Once a car was stopped in any position on the track, the inputs would not change, therefore no new actions would be taken by the car hence becoming stuck and not hitting any walls to reset itself. This observation did prove that the network was able to learn more control and have a greater ability to ‘understand’ what it is doing, much more so than the simple, single layer network.

Iteration 4:

A new topology was tried with this same weight range, two hidden layers with sizes of 10 and 5. This change did not bring anything new to the experiment and behaved extremely similarly to how the previous experiment did.

Iteration 5:

Changing the topology again, to an even deeper neural network with topology of hidden layers being 3, 5, 3, whilst sticking with the -2.5 and 2.5 weight ranges. Using this deeper neural network, some of the cars within this attempt were able to finally make it around the 90-degree right turn for the first time, indicating that the DNN and deep learning was producing the expected and desired results from deep learning architectures.

Now that the fundamental issue of navigating the corners was resolved, I was able to progress to other aspects of the A.I., such as the fitness of the cars and how this performed and improved over the course of multiple generations. Observing these values allowed me to spot some unwanted behavior out of the fitness values created by the cars. These unwanted values produced by the fitness function resulted in a car that was able to make it around a good amount of the track and navigate many corners, even if a little slowly, could end with a lower fitness than a car that did not progress far around the track at all, but was very fast over the distance it did travel. This was clearly due to the fitness function selecting the average speed of the car, meaning it did not consider the actual distance travelled by the car, only the speed in the time it was travelling.

With a fitness function like this, a car that went in a straight line and crashed would have a higher fitness than a car that managed to navigate the track (as previously explained) even if that car travelled around the track quickly. This is because the average speed of the car completing the track (or the majority of it) would still be lower due to the need to slow down for corners, thus resulting in a lower average speed metric.

Iteration 6:

A new fitness function was devised to try and solve this problem. The most important aspect of this autonomous vehicle is the ability to navigate the whole track and, while I wanted to push the car to see how similar to a real driver this setup and technique could become, the car is not successful at all if it is unable to navigate a lap of the track.

The new fitness function therefore predominantly used the distance, with the time taken to drive until a crash occurs still being able to affect the fitness value given, but to a much lesser extent than with the average speed, where distance travelled was removed entirely.

This is the new fitness function I created:

$$f = (distance - (time * 10))/100$$

Now with the distance being the key factor in the calculation of a fitness value a new set of cars/networks were generated and the algorithms run again, all with the same topology.

Upon starting this new test, it was evident that the car was performing better and get further than ever before. Now, the cars were consistently able to make it to the same position that only a couple of exceptional and 'outlier' cars could achieve from the previous attempt. Not only this but some individuals were also able to reach further distances than the previous attempt, managing to make it to the 'S' shape bends of corners 9 and 10. However, they were still getting stuck here around generation 19. This was a good improvement showing me that the new fitness function was a step in the correct direction to create a successful lap and a stable autonomous vehicle using Evolutionary Learning. While on this attempt, I also tested out some other parameters in the network, by switching the range of the weights to better understand their effects on this DNN. I retried both -2 to 2 and -5 to 5, both with negative results in comparison to the -2.5 to 2.5, which was proving to be a superior set up for this use case.

To confirm the robustness of the weight range between -2.5 and 2.5 I switched back to this without changing any of the car's network topology, meaning it was still using the DNN of 3, 5 and 3 sized hidden layers. This is where a huge jump in progress was made by the car and the potential of this A.I. method really became apparent.

By only generation 7 a car made it nearly to the end of the lap, only getting caught out on the hairpin corner, an extremely tight right-hand bend. After letting the cars evolve for a few more generations the first fully successful laps were starting to begin and the strengths of this method were very obvious.

Within only 9 generations the cars were able to make it around the track avoiding collisions with any walls, proving that evolutionary learning can create successful autonomous vehicles and produce them extremely quickly. While the cars were making it all the way around, they did still behave slightly erratically. However, as mentioned in some papers referenced in section 2.4 of this report this GA evolution can create extremely capable 'baseline' cars that can then be further tweaked and perfected through small amounts of training and evolution time, much faster than any other method of learning, such as supervised.

The cars were driving slower than desired, but this was to be expected since the fitness function was now judging on distance meaning the speed the car travelled was a secondary factor to the actual navigation and safety (collision avoidance) of the car.

Following the evolution of cars that could navigate the track safely, I continued to let the algorithm run for another couple of generations to ensure that most cars could drive themselves at least a majority of the lap before crashing. After completion of this, I then changed the fitness function to favour speed:

$$f = \frac{\left(\frac{distance}{10}\right)}{time}$$

I also took a slight amount of inspiration from Reinforcement Learning (even though this is not a RL implementation) and implemented a small punishment to a car's fitness if it did not make it a certain distance around the track. This was to remove the chance of the same issue of the first attempts where a car could drive fast but crash and still have a higher fitness than one that drives the track well. I did this because I wanted to keep the cars driving around the whole track, but to also let the parents be the fastest possible cars, so that each generation would slowly start to improve in speed, while not losing any of their navigation and safety potential for driving successful laps.

Allowing this new fitness function to run on the cars (without restarting, continuing from generation 11) showed some improvement to the speed and handling of the car, but only slightly. This only slight improvement could be because the individuals had already been tuned to the distance measure, such that it was then harder to force a change to increase the speed quickly. As a result of this observation, I increased the mutation rate for each new individual from 1% to 5% to potentially increase the learning and explore new territory while tuning the cars for speed with this fitness function.

The cars were increasing their speed and becoming more stable in their behavior. Each car was driving in a more predictable manor (the wiggling behavior was bred out) which is both safer and faster. Constantly turning left and right while trying to drive straight is much slower than a straight line as there is a larger distance to travel with all the extra turns and increased friction slowing the car down also as a result of the extra turning.

The speed managed to improve by 2-3 speed units (measured by Unity) by generation 20, showing improvement, but this was very slow and showed limited improvement per new generation.

[V1]: https://youtu.be/NRBrf8UH1Sg?t=83

Iteration 7:

Building from this experience, a new fitness function was once again devised. Rather than using two fitness functions separately and manually switching between them, I wanted to test whether the same effect could be achieved (without needing any punishment for the cars) by combining the two functions. To do this I took the two separate fitness functions for both distance and speed fitness and summed them into one final fitness value. I expected this would work because as cars get further around the track and complete a full circuit the remaining determining factor would only be the speed they could travel (as the maximum distance the cars can travel around the track had been reached), so only the fastest will then be picked.

When starting out, this compound function should also work well as the values are not biased either way to speed or distance. Therefore, where one car travels a good distance but very slowly and another travels a moderate distance, but fast, the second car would still be chosen as overall it is a much better performing car since it can navigate some of the track and can keep an acceptable pace while doing so. Breeding this individual will yield an overall faster and fitter generation by the end.

It also prevents the case where a car can drive fast but only in a straight line from gaining a fitness value too high and skewing the gene pool.

New fitness function:

$$df = \left(\frac{distance}{10}\right) - time$$

$$sf = \frac{\left(\frac{distance}{10}\right)}{time}$$

$$f = df + sf$$

Within the function, the distance is divided by 10 as it is measured with the distance gates (which have a number greater than 30,000) in Unity so by reducing by a factor of 10, it makes the distance much more realistic and useable for any average speed calculations and comparisons.

The topology remained the same as the previous generation (3, 5, 3 and -2.5 to 2.5), so the only change was the improved fitness function. After 13 generations cars were starting to navigate all the way around the track showing it was a rapid way to evolve an A.I. However, it did take a little longer in terms of the generation count compared to the previous attempt, but this was to be expected as the focus is not only on the single measure of distance, but also the speed.

The result of this iteration was that the cars now showed true potential to deliver a good outcome, as they managed to complete full laps and do so with a relatively good average speed. I also observed some cars accelerating along the straights on the track and then slowing down for corners replicating the behavior of a real car on track, where it will accelerate whilst exiting a corner and onto a straight. The way the cars slowed themselves down though, was not ideal. To slow down for a corner, the car would start to wiggle again, side to side while not applying any throttle, or any brake. This action slowed the car down (effectively braking) due to the increased friction from turning constantly and the decreased throttle input from the network, however, it is not ideal for real use as using the brakes and slowing down in a straight line is both safer and faster

Overall, this car performed better than the previous attempt by maintaining a much higher average speed, plus the acceleration and deceleration (even if performed in a less than desirable manor) on straights and combined with the ability to make it all the way around the track indicated this as being the best attempt and DNN setup found so far.

Iteration 8, The Best Attempt:

Continuing with the 3-layer DNN, I changed the number of genes (nodes) per layer to 10, 7 and 4 in an attempt to increase the ability of the car to learn. Totalling to a network size of (including the inputs and outputs): 10, 10, 7, 4, 2.

Since I already knew how the range of weight values affect the car, I decreased this to -2 and 2 in an attempt to decrease the wiggle when approaching corners to try and force the car to learn how to use the brakes more effectively. That, in conjunction with the larger network, would test whether the ability to learn these more complex actions was improved.

As expected, including more nodes within the network resulted in a longer training time, but by generation 17, a car did learn the ability to use the brakes for Park bend after the long straight showing that this extended network and slight decrease in the extremity of the weight values had improved the cars in the fashion that I had hoped to achieve.

[V1]: https://youtu.be/NRBrf8UH1Sg?t=239

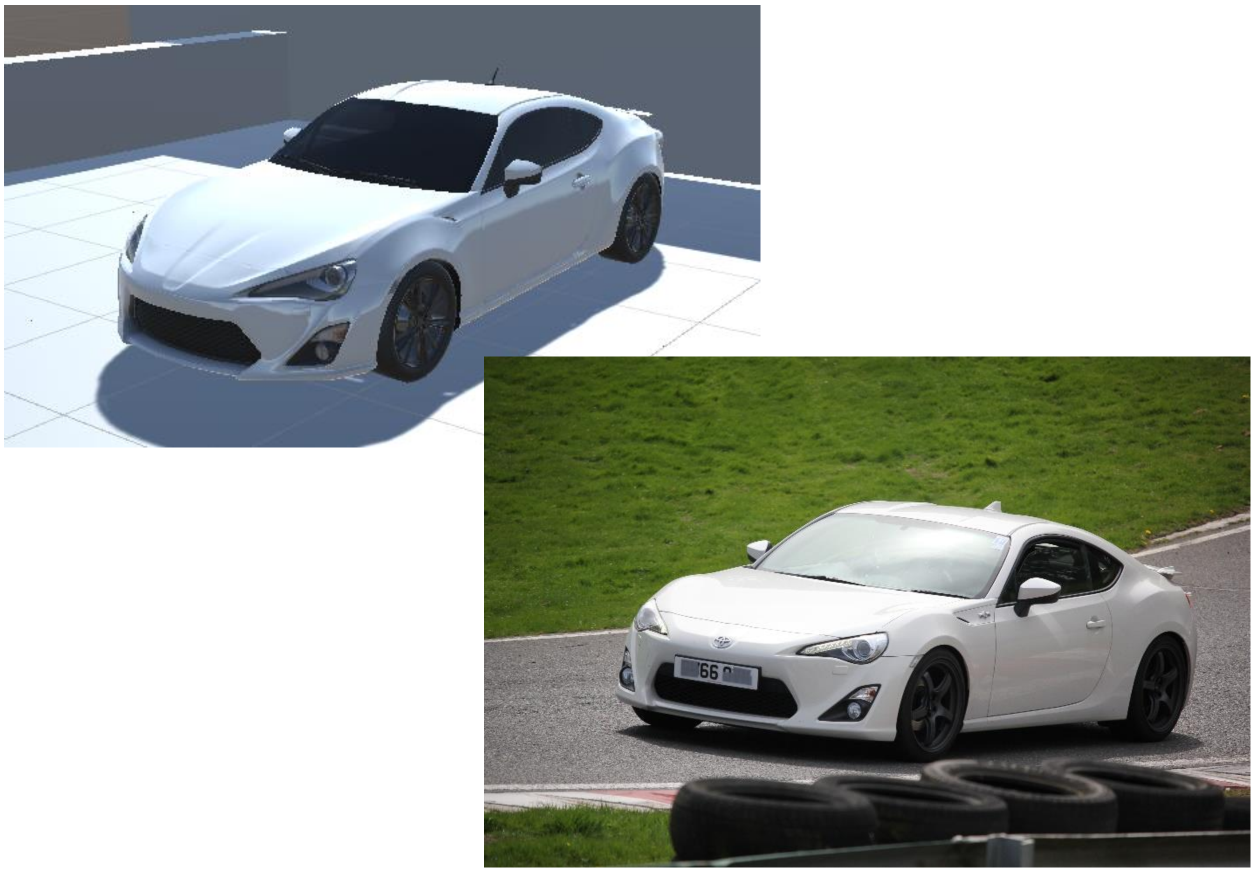
Running for more evolutions after this point saw the cars continue to improve in their control and speed, although some behavior similar to the wiggling for corners did start to return. This was not to the extent of the previous attempt, so an improvement was made and it was also observed that the cars braked during a corner if they were travelling too fast in that corner. Whilst this is not ideal on a real track with real cars at high speeds, for this simulation it did not cause an issue and was still able to show the ability of the A.I. to learn the more advanced controls of the cars such as braking.

[V1]: https://youtu.be/NRBrf8UH1Sg?t=281

Out of the numerous different set ups for parameters and network topologies, this set up delivered the best outcome for the cars being able to produce the highest overall fitness value (i.e. for its best car). Viewing the simulation running, this attempt subjectively produced the cars with the best visual performance, not just delivering a higher fitness value. The cars drive well, they accelerate, brake and drive themselves around the track quickly and safely (avoiding collisions with the walls), with some persistence to the minor imperfections such as the wiggle before entering a corner and late turning.

5.2 Comparisons to the real world

Figure 16: Image of both simulated and real cars



On 20th April 2022, I was able to take my own car (also a Toyota GT86) to Cadwell Park on a track day to capture video so that I could use this as a comparison of a real driver to the autonomous car behaviour on the same track. I was able to capture 2 perspectives of the car whilst on track. One was taken from inside the cockpit of the car (a first-person point of view) and another where I was following a ‘camera car’ which videoed the front of my car to see how the I was positioned on the track.

With these videos I wanted to show where the autonomous car had some similarities as well as highlighting the areas that would need further improvement. I have highlighted potential improvements and developments moving on from the scope of this project in section 6.

I split the track into 4 different sections, to help understanding and comparison of the cars.

[V2]: https://youtu.be/WKem0OjArJs

Here the first section of the lap shows how the A.I behaves and shows the general trend for the rest of the comparison too. The A.I. car likes to stay as centered as possible on the track, leaving as much room as possible between the walls and itself. This is definitely the safest way to navigate a track such as this as the likely hood of any collisions with a wall reduced under normal driving. Ignoring the wiggling, as this has been addressed in the previous section of the results gathered, comparing to the real car shows that whilst the racing line is not following before entering and exiting some tighter corners, on the first long right-hand corner however both cars are relatively centered on the track, showing a slight equality between the two even if this is due to the A.I. favouring the centre for safety reasons.

[V2]: https://youtu.be/WKem0OjArJs?t=103

This next section of the track shows the corners of Park through Gooseneck. Another large behavioural aspect of the A.I. car is first shown here as well. On the approach to Park, I turn in relatively early to prepare for the corner as it is relatively sharp, especially after a long straight where speeds can exceed 100mph (in a real car). Comparatively, the A.I. does not make such a preparation for the corner. The speed the A.I. carried into the corner was not particularly high to cause it to collide with the wall, but the very late turn seems to be a reaction to the wall being very close and this led to very sharp and harsh turns of the wheels, leading to the correction needed to get the car back on track and going straight for the next long right hander corner, heading into the right-left of Gooseneck. The car was very stable throughout this section and again, was able to imitate myself on track more closely by staying central on the road. The A.I. was also more similar to my real drive where it used more of the track for the Gooseneck bends than previously seen and it was able to make less sharp and late steering actions to avoid the wall and giving a better and more realistic driving line through this set of turns.

[V2]: https://youtu.be/WKem0OjArJs?t=192

Next are the couple of corners of Mansfield and The Mountain, so called due to its steep incline. As I have little experience with 3D modelling, when creating the model of Cadwell Park I was unable to capture the elevations changes that occur on track for the A.I to learn. The car's late turn in and extreme turn angle behaviour is very apparent here with this set of turns. On each of the corners here, which are both similar, tight turns with the opposite turn in rapid succession, the car makes its turn late into the corner, which needs an extreme angle to compensate for this tardiness of the turn. This then compounds into the next turn leading to another late and extreme turn making the car behave very erratically, then causing the correction to be even greater, when accelerating onto the next straight.

[V2]: https://youtu.be/WKem0OjArJs?t=279

In the last section of the track there is a set of 'S' bends which for a real driver, as can be seen by the video of myself driving, are very smooth and do not require too much steering input as you can use the whole track to reduce the sharpness of the corners allowing the car to flow over the bends quite elegantly. With the autonomous car, as has already been discussed, the habit of late turn in and an extreme reaction did not allow for this elegance to transpire within the A.I. This sequence of corners was one of the main challenges when trying to replicate the real drive on track. On each of the turns in this set (called Hall Bends and the Hairpin), the autonomous car took a late and extreme turn angle around which it then tried to correct as it was entering the next corner in the sequence. This drastically hinders the performance and highlights the main shortfall of the data and inputs given into the network, rather than the Evolutionary technique itself. Given an adequate amount of data process, the A.I. could likely begin to mimic the driving style and line I took around these corners.

I believe the reason for this behaviour of the car is due to the lack of knowledge about the track and its environment. All that the car can see is what is directly in front of it, therefore making it unable to plan for a longer set of continuous corners, leading to these late and extreme turns sometimes made, especially with tighter corners, where the simulated LIDAR rays cannot see around. In section 6 (Further Work) a potential enhancement for this issue is discussed and that should allow the car to "see" more of the track and plan ahead for sets of corners, thus allowing it to take better, safer and faster lines around the track.

Section 6: Conclusion and Further Work

Creating an autonomous vehicle to navigate Cadwell Park in this project has been a success.

I have been able to employ an Evolutionary Algorithm to train a Deep Neural Network to safely drive itself around a complex track, as per the original concept of this project. As the project developed, I have also been able to employ additional and different concepts I have learnt through my time at university in other related modules, as well as key concepts from other academic papers as seen in the literature review section of this project and report. The further additional scope to the project of seeking to develop the AI to improve the speed of cars around the track was also successful. The car was able to learn when to accelerate after a bend onto a straight and to slow down for corners such that it can successfully make the turn. These aspects together allowed the car to keep a good pace around the track, as this additional scope required.

Researching the different A.I. methods such as supervised, reinforcement and of course, evolutionary in more depth has broadened my knowledge on this topic and as a result allowed me to use these concepts in an application of my own choosing, even though I was not able to find many existing use cases doing a similar thing to learn from and refer to. Therefore, from a personal perspective, this project has enabled me to learn a significant amount about Artificial Intelligence methods and their application and has been a huge success for me.

I have also been able to learn how individual parameters on a Neural Network can influence its actions and outputs. Specifically the range of the weights used and how parameters within a Genetic Algorithm affect the evolution of the Networks within the population. For example, increasing the mutation rate can allow for a larger search space to be explored, but may lead to a longer convergence time. This can result in the potential to use an adjustable mutation rate that decreases as fitness increases to benefit both convergence time and the area explored.

With this project, I have also been able to add to the A.I. space, that being Autonomous Vehicles with Evolutionary Learning. This project has been able to showcase the versatility of this A.I. method to rapidly evolve and develop an autonomous car that can be the foundation for building on top of to create safe self-driving vehicles that could be used alongside other road users.

During the various iterations the cars did however behave in some unexpected ways, such as with the amount of wiggling that occurred during some sections of the track and with the late and extreme turns into some of the corners. While the car was a success and safely navigated the whole track, these aspects leave some room for improvement, which can be addressed as described in the Further Work section below.

To conclude, I have been able to grow my knowledge of A.I. and Evolutionary Learning, apply this technique into a new emerging technology field and explore the benefits of this method, such as its rapid learning. I have also been able to learn about various other methods of Artificial Intelligence and build on my knowledge already embedded through my studies at university.

Further work:

Whilst this project successfully achieved its objectives, some improvements and further developments became clear during testing.

Firstly, I believe the inputs the Neural Network receives could be improved by adding a camera to the car. Using this as input would help with the cars understanding of the track and where it can drive, as well as extra knowledge of what will be following a given corner. Comparing my method to this, my method allowed the car to only see what was directly in front of it and nothing further ahead, which, prevented any planning ahead by the A.I., such as seeing a corner and also being able to tell how sharp the corner is as well as what is following the corner (i.e., is there another bend or a straight).

If this could be implemented into the car, I believe this would have a material impact on the stability of the car and prevent the 'wiggling' that occurred with my method. It would also allow for the car to "see" further ahead on the track and better prepare for the upcoming corners and sets of continuous corners, which may help create a more realistic driving simulation when compared to a real driver around the same track. In this regard, hopefully the car would be able to drive more accurate racing (or optimum) lines and avoid the late and extreme turns into some corners, thus removing the need to then make corrections and causing issues for the next corner, which would reduce the overall speed. Ultimately this would also make the car more stable which will also lead to it becoming safer.

This addition of a camera view would not replace the current ray simulation as these rays can be extremely useful for measuring distances, since they will be much more accurate than a camera view estimating the distance to certain object, or walls etc. The two can be used in conjunction to create a more robust and versatile autonomous vehicle that will be able to adapt and learn more situations making it more suitable for real world use.

Starting to apply this method to more complex scenarios will also be a key area of development for this evolutionary method. Training in scenarios with other cars, traffic signs/signals and some unpredictable events will be needed to test the limits of what this method of learning can be used for within the space of autonomous vehicles. Some other areas of A.I. will also be employed here, such as Image Recognition to be able to tell apart the different objects within the cameras view. For example, the car would need to be able to tell the difference between a pedestrian and traffic lights.

These developments would push the boundaries of Evolutionary Learning and help explore the area of autonomous vehicles even further.

References

[1] Schrider, Daniel R., and Andrew D. Kern. "Supervised machine learning for population genetics: a new paradigm." *Trends in Genetics* 34.4 (2018): 301-312.

[2] LeCun, Yann, et al. "A theoretical framework for back-propagation." *Proceedings of the 1988 connectionist models summer school*. Vol. 1. 1988.

[3] P. Sharma, D. Austin and H. Liu, "Attacks on Machine Learning: Adversarial Examples in Connected and Autonomous Vehicles," *2019 IEEE International Symposium on Technologies for Homeland Security (HST)*, 2019, pp. 1-7, doi: 10.1109/HST47167.2019.9032989.

[4] Fridman, Lex. "Human-centered autonomous vehicle systems: Principles of effective shared autonomy." *arXiv preprint arXiv:1810.01835* (2018).

[5] O'Shea, Keiron, and Ryan Nash. "An introduction to convolutional neural networks." *arXiv preprint arXiv:1511.08458* (2015).

[6] Chiaroni, Florent, et al. "Self-supervised learning for autonomous vehicles perception: A conciliation between analytical and learning methods." *IEEE Signal Processing Magazine* 38.1 (2020): 31-41.

[7] M. Usama et al., "Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges," in IEEE Access, vol. 7, pp. 65579-65615, 2019, doi: 10.1109/ACCESS.2019.2916648.

[8] N. Grira, M. Crucianu, and N. Boujemaa, ‘‘Unsupervised and semisupervised clustering: A brief survey,’’ Rev. Mach. Learn. Techn. Process. Multimedia Content, vol. 1, pp. 9–16, Jul. 2004.

[9] Aristidis Likas, Nikos Vlassis, Jakob J. Verbeek, The global k-means clustering algorithm, Pattern Recognition, Volume 36, Issue 2, 2003

[10] J. Zhang et al., "Evolutionary Computation Meets Machine Learning: A Survey," in IEEE Computational Intelligence Magazine, vol. 6, no. 4, pp. 68-75, Nov. 2011, doi: 10.1109/MCI.2011.942584.

[11] Sacha Gobeyn, Ans M. Mouton, Anna F. Cord, Andrea Kaim, Martin Volk, Peter L.M. Goethals, Evolutionary algorithms for species distribution modelling: A review in the context of machine learning, Ecological Modelling, Volume 392, 2019

[12] F. H. F. Leung, H. K. Lam, S. H. Ling and P. K. S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," in IEEE Transactions on Neural Networks, vol. 14, no. 1, pp. 79-88, Jan. 2003, doi: 10.1109/TNN.2002.804317.

[13] L. Chen and D. Alahakoon, "NeuroEvolution of Augmenting Topologies with Learning for Data Classification," 2006 International Conference on Information and Automation, 2006, pp. 367-371, doi: 10.1109/ICINFA.2006.374100.

[14] Bishop, Chris M. "Neural networks and their applications." *Review of scientific instruments* 65.6 (1994): 1803-1832.

[15] Vogl, T.P., Mangis, J.K., Rigler, A.K. *et al.* Accelerating the convergence of the back-propagation method. *Biol. Cybern.* **59,** 257–263 (1988). https://doi.org/10.1007/BF00332914

[16] Targ, Sasha, Diogo Almeida, and Kevin Lyman. "Resnet in resnet: Generalizing residual architectures." *arXiv preprint arXiv:1603.08029* (2016).

[17] Holland, John H. “Genetic Algorithms.” *Scientific American*, vol. 267, no. 1, 1992, pp. 66–73, http://www.jstor.org/stable/24939139. Accessed 6 Apr. 2022.

[18] Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA 2007, Warsaw, Poland, April 11-14, 2007, Proceesings, Part II, Bartlomiej Beliczynski, Andrzej Dzielinski, Marcin Iwanowski**,** Bernardete Ribeiro

[19] Haldurai, L., T. Madhubala, and R. Rajalakshmi. "A study on genetic algorithm and its applications." *International journal of computer sciences and Engineering* 4.10 (2016): 139.

[20] Bhoskar, Ms Trupti, et al. "Genetic algorithm and its applications to mechanical engineering: A review." *Materials Today: Proceedings* 2.4-5 (2015): 2624-2630.

[21] Arrigoni, Stefano, Francesco Braghin, and Federico Cheli. "MPC path-planner for autonomous driving solved by genetic algorithm technique." *arXiv preprint arXiv:2102.01211* (2021).

[22] Jalali, Seyed Mohammad Jafar, et al. "Optimal autonomous driving through deep imitation learning and neuroevolution." *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, 2019.

[23] P. Caamano, F. Bellas and R. J. Duro, "Augmenting the NEAT algorithm to improve its temporal processing capabilities," 2014 International Joint Conference on Neural Networks (IJCNN), 2014, pp. 1467-1473, doi: 10.1109/IJCNN.2014.6889488.

[24] Chen, Lin, and Damminda Alahakoon. "NeuroEvolution of augmenting topologies with learning for data classification." *2006 International Conference on Information and Automation*. IEEE, 2006.

[25] Ibrahim, Mohamed Yilmaz, et al. "Advances in neuroevolution through augmenting topologies–a case study." *2019 11th International Conference on Advanced Computing (ICoAC)*. IEEE, 2019.

[26] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, Babak Hodjat, Chapter 15 - Evolving Deep Neural Networks, Artificial Intelligence in the Age of Neural Networks and Brain Computing

[27] Inagaki, Toshiyuki, and Thomas B. Sheridan. "A critique of the SAE conditional driving automation definition, and analyses of options for improvement." *Cognition, technology & work* 21.4 (2019): 569-578.

[28] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda and T. Hamada, "An Open Approach to Autonomous Vehicles," in IEEE Micro, vol. 35, no. 6, pp. 60-68, Nov.-Dec. 2015, doi: 10.1109/MM.2015.133.

[29] M. Betke and L. Gurvits, "Mobile robot localization using landmarks," in IEEE Transactions on Robotics and Automation, vol. 13, no. 2, pp. 251-263, April 1997, doi: 10.1109/70.563647.

[30] S. Kuutti, R. Bowden, Y. Jin, P. Barber and S. Fallah, "A Survey of Deep Learning Applications to Autonomous Vehicle Control," in IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 2, pp. 712-733, Feb. 2021, doi: 10.1109/TITS.2019.2962338.

[31] M. Al-Qizwini, I. Barjasteh, H. Al-Qassab and H. Radha, "Deep learning algorithm for autonomous driving using GoogLeNet," 2017 IEEE Intelligent Vehicles Symposium (IV), 2017, pp. 89-96, doi: 10.1109/IVS.2017.7995703.

[32] Sainath, G., et al. "Application of Neuroevolution in Autonomous Cars." *International Virtual Conference on Industry 4.0*. Springer, Singapore, 2021.

[33] Trasnea, Bogdan, et al. "GridSim: A vehicle kinematics engine for deep neuroevolutionary control in autonomous driving." *2019 Third IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2019.

[34] Li, Yuxi. "Deep reinforcement learning: An overview." *arXiv preprint arXiv:1701.07274* (2017).

[35] Sutton, Richard S., and Andrew G. Barto. "Reinforcement learning." *Journal of Cognitive Neuroscience* 11.1 (1999): 126-134.

[36] Watkins, Christopher JCH, and Peter Dayan. "Q-learning." *Machine learning* 8.3 (1992): 279-292.