**TRƯỜNG ĐẠI HỌC TRẦN ĐẠI NGHĨA**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

MÔN HỌC: MẠNG NEURON

**ĐỀ TÀI:** Ứng dụng mạng Neuron mô phỏng xe tự lái trên Unity

**G/V hướng dẫn**: Phùng Thế Bảo

**Nhóm 4:**

Sinh viên thực hiện*:*

Phạm Duy Khánh MSSV: 18DDS0803115

Dương Tôn Khánh MSSV: 18DDS0803114

Lê Quang Khải MSSV: 18DDS0803112

Nguyễn Thành Khiêm MSSV: 18DDS0803116

Hồ Chí Minh, 10/2021

MỞ ĐẦU

Trong thế giới hiện đại, máy học (Machine Learning) đang đóng vai trị quan trọng hơn bao giờ hết trong nhiều lĩnh vực khác nhau như: di truyền học, thị giác máy tính, nhận diện giọng nói, xử lý ngôn ngữ tự nhiên, robot, dự đoán thị trường chứng khoán, v.v… Máy học cũng hiện hữu rất nhiều trong đời sống hằng ngày như: Facebook sử dụng máy học để đề xuất kết bạn hay Google sử dụng máy học làm nền tảng cho công cụ tìm kiếm của họ.

Trong ngành công nghiệp ô tô cũng như giới học thuật đã tiến hành rất nhiều nghiên cứu liên quan đến xe tự lái. Xe tự lái là một chủ đề rất thú vị, nó có tiềm năng mang lại nhiều lợi ích cho xã hội như: giảm tai nạn giao thông, giảm ùn tắc giao thông, v.v… Theo báo cáo từ NHTSA, hơn 90% các vụ va chạm ô tô xảy ra do lỗi của người lái xe. Và những thương vong trên có thể giảm thiểu tối đa bằng cách triển khai các tính năng tự động trong ô tô. Do đó, xe tự lái trở thành một phương tiện có khả năng giúp con người tránh được những thương vong đáng tiếc và hiện đang được phát triển bởi nhiều công ty. Với mong muốn tìm hiểu và phát triển công nghệ mới này, nhóm chúng tôi thực hiện nghiên cứu đề tài “Ứng dụng mạng Neuron mô phỏng xe tự lái trên Unity”. Trong bài tập lớn này, chúng tôi ứng dụng máy học để xây dựng một mô hình xe tự lái cơ bản. Cụ thể hơn, chúng tôi xây dựng mô hình xe có thể dự đoán góc lái để giúp xe bám theo làn đường.

LỜI CẢM ƠN

Để thực hiện và hoàn thành tốt bài tập lớn này, em đã nhận được sự giúp đỡ và hướng dẫn rất tận tình của các thầy cô thuộc Đại học Trần Đại Nghĩa. Em xin cảm ơn các thầy cô thuộc bộ môn chuyên ngành đã cung cấp cho chúng em các thông tin, kiến thức vô cùng quý báu và cần thiết trong suốt thời gian qua để em có thể thực hiện và hoàn thành bài tập lớn của mình. Đặc biệt em xin chân thành cảm ơn thầy, người đã trực tiếp hướng dẫn chúng em trong thời gian thực hiện bài tập lớn này. Cuối cùng, xin chân thành cảm ơn các bạn trong ngành công nghệ thông tin đã ủng hộ, giúp đỡ, chia sẻ kiến thức, kinh nghiệm và tài liệu có được giúp chúng tôi trong quá trình nghiên cứu và thực hiện đề tài.

Do giới hạn về mặt thời gian và kiến thức cũng như kinh nghiệm thực tiễn nên đề tài không tránh khỏi những sai sót. Em rất mong nhận được sự thông cảm của quý thầy cô và mong đón nhận những góp ý của thầy cô và các bạn.

Em xin chân thành cảm ơn!

MỤC LỤC

[CHƯƠNG 1. TỔNG QUAN VỀ HỆ THỐNG XE TỰ LÁI SỬ DỤNG CÔNG NGHỆ MẠNG NEURON TRUYỀN THẲNG 4](#_Toc84230750)

[1.1 Tổng quan về xe tự lái 4](#_Toc84230751)

[1.2 Mục tiêu đề tài 5](#_Toc84230752)

[CHƯƠNG 2: CƠ SỞ LÝ THUYẾT CỦA XE TỰ LÁI 6](#_Toc84230753)

[2.1 Chuyển động chất điểm trong vật lý 6](#_Toc84230754)

[2.2 Mạng Neural truyền thẳng 6](#_Toc84230755)

[2.3 Hàm sigmoid & hàm tan 7](#_Toc84230756)

[2.4 Thuật toán di truyền (GAs) 8](#_Toc84230757)

[CHƯƠNG 3: XÂY DỰNG MÔ HÌNH XE TỰ LÁI TRÊN UNITY 10](#_Toc84230758)

[3.1 Nguyên tắc hoạt động. 10](#_Toc84230759)

[3.2 Mô hình xe tự lái trên unity 10](#_Toc84230760)

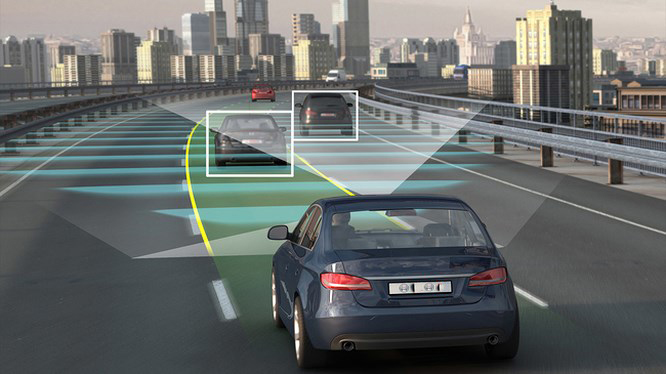
[3.3 Cài đặt chương trình 11](#_Toc84230761)

[3.4 Kết quả thử nghiệm và đánh giá. 11](#_Toc84230762)

# CHƯƠNG 1. TỔNG QUAN VỀ HỆ THỐNG XE TỰ LÁI SỬ DỤNG CÔNG NGHỆ MẠNG NEURON TRUYỀN THẲNG

* 1. Tổng quan về xe tự lái

Cùng với sự phát triển của khoa học công nghệ, việc ứng dụng và đưa trí tuệ nhân tạo vào các sản phẩm công nghệ là một xu thế tất yếu. Không những vậy công nghệ trí tuệ nhân tạo cũng đưa các máy móc mà vốn dĩ chúng ta nghĩ không thể phát triển hơn nữa lên một tầm cao mới hơn. Và một trong số đó là công nghệ xe tự lái.



Hình 1: Minh họa về xe tự lái

Trước đây, đã có nhiều nghiên cứu về điều hướng cho xe tự lái đã được thực hiện. Bao gồm các phương pháp như: phát hiện làn đường theo thời gian thực để điều hướng tự động, … Hiện tại, xe tự lái không chỉ dừng lại ở việc đánh lái, tăng tốc hay phanh trong một số trường hợp nhất định, nó còn có thể tránh né vật cản, xác định các vật thể chuyển động trên đường để có thể hoàn toàn thay thế người lái xe, giúp cho họ có thể thoải mái hơn trên những tuyến đường cao tốc, giảm rủi ro tai nạn giao thông. Nhờ vào sự phát triển của trí tuệ nhân tạo đã giúp cho việc tạo ra những chiếc xe tự lái khả thi hơn bao giờ hết.

Để có cái nhìn rõ hơn về xe tự lái cũng như lý do tại sao lại nói xe tự lái chính là xu hướng của tương lai thì dựa vào những thơng tin sau:

- Trong các trường hợp gây tai nạn giao thông hiện nay thì 90 ~ 95% những vụ xảy ra tai nạn là do lỗi của sự bất cẩn của con người. Chính vì thế, khi xe tự lái được rộng rãi theo như các nhà khoa học tính tốn thì công nghệ này sẽ giúp giảm được hơn 1 triệu người tử vong hằng năm vì tai nạn giao thông.

- Việc xe tự lái phát triển cũng đồng nghĩa với đó là giúp tiết kiệm hàng trăm cho đến hàng tỷ USD (chi phí người dùng phải bỏ ra để sửa ô tô).

- Giúp những người không có khả năng lái xe, người tàn tật và người già di chuyển dễ dàng (như tìm hiểu về xe tự lái ở bên trên thì các bạn cũng thấy đây là một loại xe điều khiển không cần tới con người). Xe tự lái có thể được định nghĩa là một chiếc xe đã được tự động hóa bằng cách sử dụng một số loại phương pháp thuật toán. Nó có thể lái xe đến bất cứ đâu hoặc thực hiện các nhiệm vụ (liên quan đến việc lái xe) mà không có sự can thiệp của con người. Với việc sử dụng phương pháp nhân bản hành vi (là một phương pháp bắt chước hành vi của con người với sự trợ giúp của máy móc) để học các mẫu trong dữ liệu bằng cách quan sát hành vi của con người tại bất kỳ thời điểm nào và sau đó sao chép nó khi được yêu cầu giúp cho hệ thống tự lái xử lý chính xác trong các tình huống khác nhau.

* 1. Mục tiêu đề tài

- Áp dụng được những kiến thức được học về Neural Network qua bài tập thực tế. Nắm rõ cá quá trình của Machine Learning để giải quyết các bài toán có độ phức tạp đòi hỏi có độ chính xác cao và tối ưu hóa kết quả đạt được.

- Xây dựng bộ công cụ phần mềm huấn luyện mạng neuron cho mô hình giả lập xe tự lái trên Unity

- Tăng kĩ năng lập trình qua ứng dụng thực tế, thực hiện làm việc nhóm một cách hiệu quả

# CHƯƠNG 2: CƠ SỞ LÝ THUYẾT CỦA XE TỰ LÁI

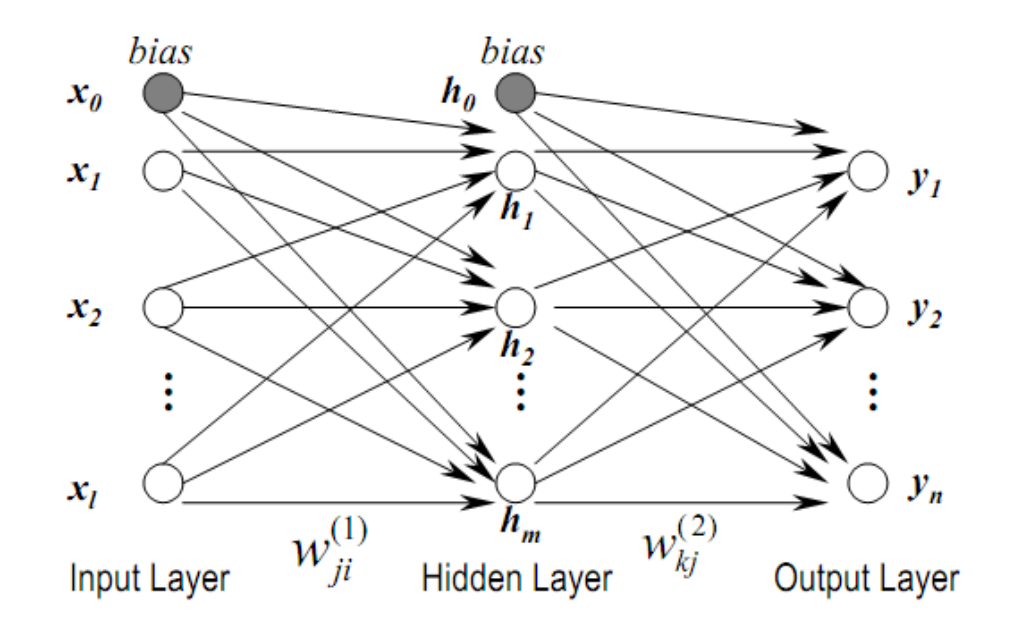
## 2.1 Chuyển động chất điểm trong vật lý

Chuyển động, trong vật lý, là sự thay đổi vị trí trong không gian theo thời gian của chất điểm hay một hệ chất điểm. Trong đó chất điểm là một điểm hình học không có kích thước hoặc kích thước vô cùng nhỏ bé nhưng có khối lượng. Chất điểm không thay đổi vị trí trong không gian theo thời gian thì đứng yên. Chuyển động của chất điểm chỉ có 2 dạng là chuyển động đều (tức là chuyển động với vận tốc không đổi) và chuyển động có gia tốc (tức là có sự thay đổi vận tốc khi chuyển động). Chất điểm không chịu tác dụng của lực thì sẽ đứng yên hoặc chuyển động đều, tức là gia tốc bằng 0.

Tập hợp nhiều chất điểm gọi là hệ chất điểm. Các chất điểm trong hệ chất điểm có thể độc lập chuyển động hay đứng yên, và giữa chúng có thể có sự thay đổi hay không thay đổi khoảng cách tương đối với nhau theo thời gian. Sự chuyển động tương đối của hệ chất điểm có được là nhờ các loại lực tương tác giữa chúng với nhau (trong bốn loại lực cơ bản). Hệ chất điểm mà luôn giữ các khoảng cách tương đối giữa các chất điểm trong hệ không đổi, dù các chất điểm thuộc hệ có chuyển động hay đứng yên, thì được gọi là vật rắn hay ngắn gọn là vật (Như vậy, vật rắn là hệ chất điểm có khối lượng, luôn giữ khoảng cách với nhau không đổi khi chuyển động). Khoảng cách không đổi của các chất điểm trong vật rắn đem lại sự phụ thuộc không tự do của các chất điểm gọi là liên kết của chất điểm trong vật rắn. Các dạng chuyển động cơ bản của vật rắn bao gồm chuyển động tịnh tiến, và chuyển động quay quanh trục hay tâm. Các loại chuyển động phức tạp của vật rắn đều có thể phân tích thành 2 dạng chuyển động cơ bản trên.

## 2.2 Mạng Neural truyền thẳng

Mạng neuron truyền thẳng nhiều lớp bao gồm một lớp vào, một lớp ra và một hoặc nhiều lớp ẩn. Các neuron đầu vào thực chất không phải các neuron theo đúng nghĩa, bởi lẽ chúng không thực hiện bất kỳ một tính toán nào trên dữ liệu vào, đơn giản nó chỉ tiếp nhận các dữ liệu vào và chuyển cho các lớp kế tiếp. Các neuron ở lớp ẩn và lớp ra mới thực sự thực hiện các tính toán, kết quả được định dạng bởi hàm đầu ra (hàm chuyển).

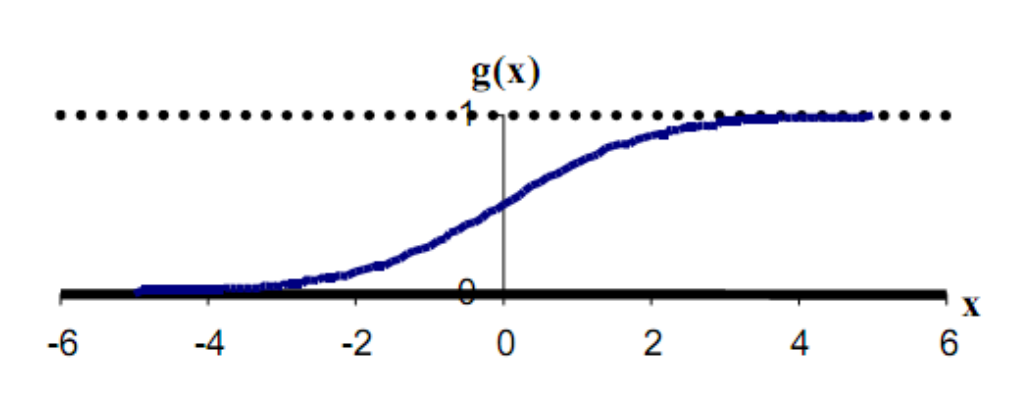


Hình 2: Sơ đồ mạng nerual nhiều lớp truyền thẳng

## 2.3 Hàm sigmoid & hàm tansig

- Hàm sigmoid (Sigmoid function (logsig):

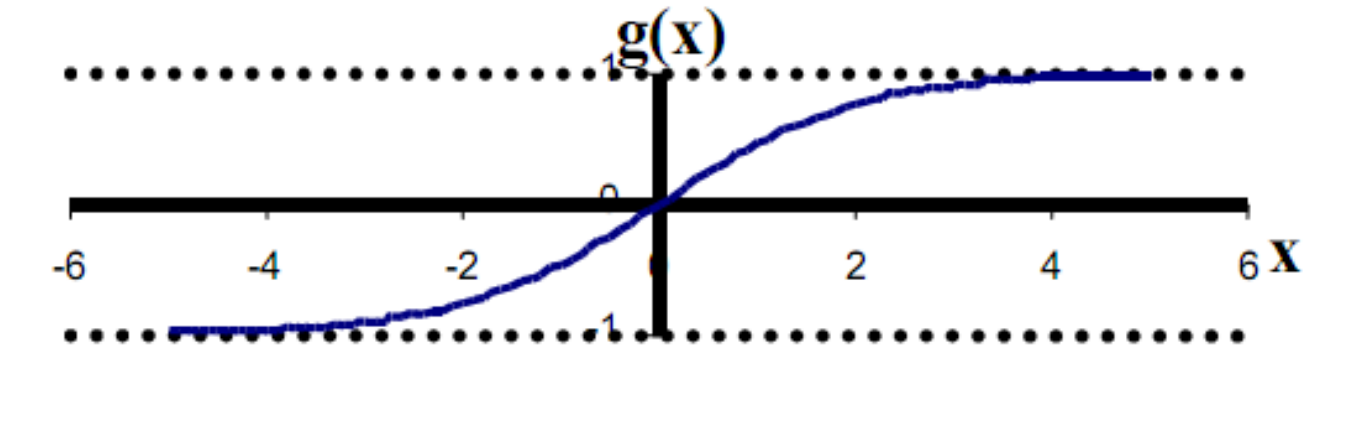
Hàm này đặc biệt thuận lợi khi sử dụng cho các mạng được huấn luyện (trained) bởi thuật toán Lan truyền ngược (back-propagation), bởi vì nó dễ lấy đạo hàm, do đó có thể giảm đáng kể tính toán trong quá trình huấn luyện. Hàm này được ứng dụng cho các chương trình ứng dụng mà các đầu ra mong muốn rơi vào khoảng [0,1].



Hình 3: Hàm Sigmoid

- Hàm sigmoid lưỡng cực (Bipolar sigmoid function (tansig))

Hàm này có các thuộc tính tương tự hàm sigmoid. Nó làm việc tốt đối với các ứng dụng có đầu ra yêu cầu trong khoảng [-1,1].

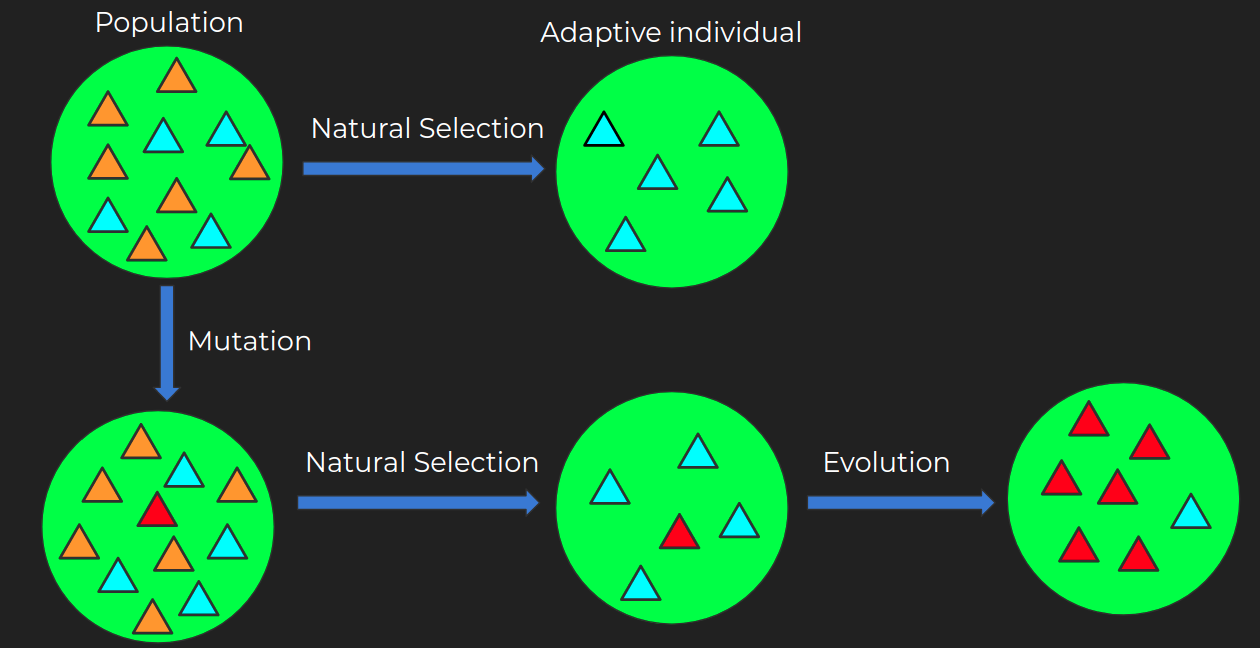


Hình 4: Hàm sigmoid lưỡng cực

## 2.4 Thuật toán di truyền (GAs)

GAs là một kỹ thuật của khoa học máy tính nhằm tìm kiếm giải pháp thích hợp cho các bài toán tối ưu tổ hợp (combinatorial optimization), là một phân ngành của giải thuật tiến hóa, vận dụng các nguyên lý của tiến hóa như: di truyền, đột biến, chọn lọc tự nhiên, và trao đổi chéo. Nó sử dụng ngôn ngữ máy tính để mô phỏng quá trình tiến hoá của một tập hợp những đại diện trừu tượng (gọi là những nhiễm sắc thể), của các giải pháp có thể (gọi là những cá thể) cho bài toán tối ưu hóa vấn đề. Tập hợp này sẽ tiến triển theo hướng chọn lọc những giải pháp tốt hơn.

- Thành phần và mối liên hệ của thuật toán di truyền:



Hình 5: Thành phần và mối liên hệ của thuật toán di truyền

1. Population - Quần thể: Một quần thể ban đầu sẽ có những cá thể nhất định với những đặc tính khác nhau, những đặc tính này sẽ quy định khả năng sinh sản, sinh tồn, khả năng đáp ứng điều kiện môi trường của từng cá thể.

2. Natural Selection - Chọn lọc tự nhiên: Theo thời gian những cá thể yếu hơn, không có khả năng sinh tồn sẽ bị loại bỏ bởi những tác nhân như tranh chấp chuỗi thức ăn, môi trường tác độc, bị loài khác tiêu diệt, … Cuối cùng sẽ còn lại những cá thể có đặc tính ưu việt hơn sẽ được giữ lại - Adaptive individual.

3. Mutation - Đột biến: Như chúng ta đã biết thì mỗi cá thể con được sinh ra sẽ được kế thừa lại những đặc tính của cả cha và mẹ. Sau một thời gian sinh sống, một quần thể sẽ đặt tới giới hạn của các cặp gen của con được tạo nên từ gen của bố mẹ. Để đạt được tới sự tiến hóa, Đột Biến chính là một trong những nguyên nhân chính, có vai trò đóng góp nguyên liệu cho quá trình Chọn lọc tự nhiên.

4. Evolution - Tiến hóa: Những cá thể đột biến không phải luôn là những cá thể mạnh mẽ và có đủ khả năng sinh tồn, Chọn lọc tự nhiên sẽ chọn ra những cá thể đột biến nhưng có thể thích nghi với môi trường sống tốt hơn những cá thể khác trong quần thể. Sau một thời gian sinh sản, những gen đột biến sẽ chiếm ưu thế và chiếm đa số trong quần thể.

# CHƯƠNG 3: XÂY DỰNG MÔ HÌNH XE TỰ LÁI TRÊN UNITY

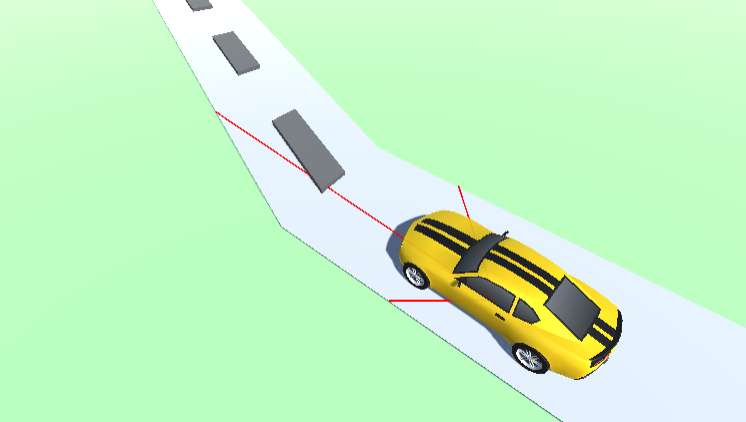
## 3.1 Nguyên tắc hoạt động.

Hiện nay, đã có rất nhiều tập đoàn sản xuất xe hơi và công nghệ lớn trên thế giới đã tham gia vào cuộc chạy đua phát triển xe hơi công nghệ tự lái thông minh (sau đây gọi tắt là xe tự lái, xe tự lái) mà không cần đến bất cư can thiệp nào từ bàn tay của con người, trong đó có những tên tuổi nổi bật như Tesla, Daimler, Google… Đối với việc phát triển xe không người lái thì phần mềm đóng vai trò quan trọng nhất. Nó được ví như não bộ, đảm bảo sự vận hành cho xe.

Ở phần mềm này mô hình xe tự lái này có hình dạng giống những chiếc ô tô thông thường. Được trang bị hệ thống nhận diện làn đường. Sau đó, phần mềm sẽ xử lý tất cả đầu vào và thông qua những thuật toán, phần mềm sẽ vạch ra các cách vận hành và gửi hướng dẫn đến bộ truyền động của ô tô, nơi có nhiệm vụ điều khiển tốc độ, phanh và hệ thống lái.

## 3.2 Mô hình xe tự lái trên unity

Mô phỏng đường và xe ô tô trong không gian 3d của Unity. Dựa vào chuyển động chất điểm trong vật lý ta xác định cho mô hình ô tô 3 véctơ a, b, c có hướng xuất phát từ trọng tâm xe cố định như hình 6.



Hình 6: Mô hình xe tự lái xây dựng trên Unity

Ta được input là tọa độ của ba vectơ a, b, c ta áp dụng vào mạng neural truyền thẳng với các trọng số w ngẫu nhiên qua lớp ẩn => output là gia tốc được lọc qua hàm Sigmoid để lấy giá trị trong khoảng [0,1] và hướng đi của xe được lọc qua hàm tansig để lấy giá trị trong khoảng [-1,1] với -1 là hướng về bên trái và 1 là hướng về bên phải ta thu được một mạng neuron.

Dựa vào thuật toán di truyền ta tạo ra một quần thể với 85 cá thể mạng neuron. Lần lượt áp dụng từng mạng neuron vào mô hình xe ô tô dựng sẵn trên Unity thực hiện đánh giá từng cá thể mạng neuron qua quãng đường đi được thuộc mạng neuron trội hay là mạng neuron lặn. Sau đánh giá toàn bộ mạng neuron ta sắp xếp và chọn ra 8 mạng neuron trội và 3 mạng neuron lặn tăng cường theo tỉ lệ trội và thực hiện chọn ngẫu nhiên các cặp mạng neuron lai với nhau tạo ra 85 mạng neuron mới cứ thực hiện như thế cho đến khi chọn được cá thể mạng neuron ưu tú nhất. Trong các phép lai ta thêm vào một vài phần tử đột biến để tăng xác xuất tạo ra cá thể mạng neuron ưu tú, giảm thời gian cũng như như số lần thực hiện phép lai.

## 3.3 Cài đặt chương trình

Thực hiện cài đặt chương trình bằng C# và Unity

3.3.1 NNet (Neuron Network)

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using MathNet.Numerics.LinearAlgebra;

using System;

using Random = UnityEngine.Random;

public class NNet : MonoBehaviour

{

public Matrix<float> inputLayer = Matrix<float>.Build.Dense(1, 3);

public List<Matrix<float>> hiddenLayers = new List<Matrix<float>>();

public Matrix<float> outputLayer = Matrix<float>.Build.Dense(1, 2);

public List<Matrix<float>> weights = new List<Matrix<float>>();

public List<float> biases = new List<float>();

public float fitness;//độ chuẩn

//Khởi tạo ban đầu

public void Initialise(int hiddenLayerCount, int hiddenNeuronCount)

{

inputLayer.Clear();

hiddenLayers.Clear();//lớp ẩn

outputLayer.Clear();

weights.Clear();

biases.Clear();

for (int i = 0; i < hiddenLayerCount + 1; i++)

{

Matrix<float> f = Matrix<float>.Build.Dense(1, hiddenNeuronCount);//1 lớp ẩn, 10 neuron

hiddenLayers.Add(f);

biases.Add(Random.Range(-1f, 1f));//random biases

// trọng số WEIGHTS với ma trận 3 hàng 10 cột(hiddenNeuronCount)

if (i == 0)

{

// inputToH1 ma trận trọng số truyền vào lớp ẩn

Matrix<float> inputToH1 = Matrix<float>.Build.Dense(3, hiddenNeuronCount);

weights.Add(inputToH1);

}

Matrix<float> HiddenToHidden = Matrix<float>.Build.Dense(hiddenNeuronCount, hiddenNeuronCount);

weights.Add(HiddenToHidden);

}

//OutputWeig ma trận trọng số từ lớp ẩn truyền ra output

Matrix<float> OutputWeight = Matrix<float>.Build.Dense(hiddenNeuronCount, 2);

weights.Add(OutputWeight);

//random biases

biases.Add(Random.Range(-1f, 1f));

RandomiseWeights();

}

//Copy ra một neuron network mới giống với neuron network cũ

public NNet InitialiseCopy(int hiddenLayerCount, int hiddenNeuronCount)

{

NNet n = new NNet();

List<Matrix<float>> newWeights = new List<Matrix<float>>();

for (int i = 0; i < this.weights.Count; i++)

{

Matrix<float> currentWeight = Matrix<float>.Build.Dense(weights[i].RowCount, weights[i].ColumnCount);

for (int x = 0; x < currentWeight.RowCount; x++)

{

for (int y = 0; y < currentWeight.ColumnCount; y++)

{

currentWeight[x, y] = weights[i][x, y];

}

}

newWeights.Add(currentWeight);

}

List<float> newBiases = new List<float>();

newBiases.AddRange(biases);

n.weights = newWeights;

n.biases = newBiases;

n.InitialiseHidden(hiddenLayerCount, hiddenNeuronCount);

return n;

}

//copy HiddenLayer qua HiddenLayer mới

public void InitialiseHidden(int hiddenLayerCount, int hiddenNeuronCount)

{

inputLayer.Clear();

hiddenLayers.Clear();

outputLayer.Clear();

for (int i = 0; i < hiddenLayerCount + 1; i++)

{

Matrix<float> newHiddenLayer = Matrix<float>.Build.Dense(1, hiddenNeuronCount);

hiddenLayers.Add(newHiddenLayer);

}

}

// Tạo ngẫu nhiên giá trị trong ma trận trọng số từ -1 -> 1

public void RandomiseWeights()

{

for (int i = 0; i < weights.Count; i++)

{

for (int x = 0; x < weights[i].RowCount; x++)

{

for (int y = 0; y < weights[i].ColumnCount; y++)

{

weights[i][x, y] = Random.Range(-1f, 1f);

}

}

}

}

//Khởi động Neuron Network được output

public (float, float) RunNetwork(float a, float b, float c)

{

inputLayer[0, 0] = a;

inputLayer[0, 1] = b;

inputLayer[0, 2] = c;

inputLayer = inputLayer.PointwiseTanh();

hiddenLayers[0] = ((inputLayer \* weights[0]) + biases[0]).PointwiseTanh();

for (int i = 1; i < hiddenLayers.Count; i++)

{

hiddenLayers[i] = ((hiddenLayers[i - 1] \* weights[i]) + biases[i]).PointwiseTanh();

}

outputLayer = ((hiddenLayers[hiddenLayers.Count - 1] \* weights[weights.Count - 1]) + biases[biases.Count - 1]).PointwiseTanh();

return (Sigmoid(outputLayer[0, 0]), (float)Math.Tanh(outputLayer[0, 1]));

}

private float Sigmoid(float s)

{

return (1 / (1 + Mathf.Exp(-s)));

}

}

3.3.2 GeneticManager (Di truyền)

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System.Linq;

using MathNet.Numerics.LinearAlgebra;

public class GeneticManager : MonoBehaviour

{

[Header("References")]

public CarController controller;

[Header("Controls")]

public int initialPopulation = 85;//dân số ban đầu

[Range(0.0f, 1.0f)]

public float mutationRate = 0.055f;//chỉ số đột biết

[Header("Crossover Controls")]

public int bestAgentSelection = 8;//số lượng phần tử trội

public int worstAgentSelection = 3;//số lượng phần tử lặn

public int numberToCrossover;//số lần lai (39)

private List<int> genePool = new List<int>();// Tạo vùng chứa phần tử trội, lặn

private int naturallySelected;//Chỉ mục của phần tử đang được chọn

private NNet[] population;// quần thể Neuron network

[Header("Public View")]

public int currentGeneration;//Thế hệ hiện tại

public int currentGenome = 0;//Phần tử đang được chọn trong quần thể

private void Start()

{

CreatePopulation();

}

//Tạo quần thể

private void CreatePopulation()

{

population = new NNet[initialPopulation];

FillPopulationWithRandomValues(population, 0);

ResetToCurrentGenome();

}

//Đặt lại neuron network mới cho xe

private void ResetToCurrentGenome()

{

controller.ResetWithNetwork(population[currentGenome]);

}

//Tạo ngẫu nhiên các cá trong quần thể

private void FillPopulationWithRandomValues(NNet[] newPopulation, int startingIndex)

{

while (startingIndex < initialPopulation)

{

newPopulation[startingIndex] = new NNet();

newPopulation[startingIndex].Initialise(controller.LAYERS, controller.NEURONS);

startingIndex++;

}

}

//Thực hiện phép thử cho từng cá thể nếu là cá thể cuối cùng thì tạo quần thể mới

//Hàm thực hiện khi mô hình có va chạm

public void Death(float fitness, NNet network)

{

if (currentGenome < population.Length - 1)

{

population[currentGenome].fitness = fitness;

currentGenome++;

ResetToCurrentGenome();

}

else

{

RePopulate();

}

}

//Khởi tạo quần thể mới từ quẩn thể cũ

private void RePopulate()

{

genePool.Clear();

currentGeneration++;

naturallySelected = 0;

SortPopulation();

NNet[] newPopulation = PickBestPopulation();

Crossover(newPopulation);

Mutate(newPopulation);

FillPopulationWithRandomValues(newPopulation, naturallySelected);

population = newPopulation;

currentGenome = 0;

ResetToCurrentGenome();

}

//Đột một cài cá thể trong quần thể với giá trị random

private void Mutate(NNet[] newPopulation)

{

for (int i = 0; i < naturallySelected; i++)

{

for (int c = 0; c < newPopulation[i].weights.Count; c++)

{

if (Random.Range(0.0f, 1.0f) < mutationRate)

{

newPopulation[i].weights[c] = MutateMatrix(newPopulation[i].weights[c]);

}

}

}

}

//Khởi tạo ma trận đột biến

Matrix<float> MutateMatrix(Matrix<float> A)

{

int randomPoints = Random.Range(1, (A.RowCount \* A.ColumnCount) / 7);

Matrix<float> C = A;

for (int i = 0; i < randomPoints; i++)

{

int randomColumn = Random.Range(0, C.ColumnCount);

int randomRow = Random.Range(0, C.RowCount);

C[randomRow, randomColumn] = Mathf.Clamp(C[randomRow, randomColumn] + Random.Range(-1f, 1f), -1f, 1f);

}

return C;

}

//Thực hiện lai tạo cá thể mới

private void Crossover(NNet[] newPopulation)

{

for (int i = 0; i < numberToCrossover; i += 2)

{

int AIndex = i;

int BIndex = i + 1;

if (genePool.Count >= 1)

{

for (int l = 0; l < 100; l++)

{

AIndex = genePool[Random.Range(0, genePool.Count)];

BIndex = genePool[Random.Range(0, genePool.Count)];

//lấy ngẩu nhiên 2 cá thể lai

if (AIndex != BIndex)

break;

}

}

NNet Child1 = new NNet();

NNet Child2 = new NNet();

Child1.Initialise(controller.LAYERS, controller.NEURONS);

Child2.Initialise(controller.LAYERS, controller.NEURONS);

Child1.fitness = 0;

Child2.fitness = 0;

for (int w = 0; w < Child1.weights.Count; w++)

{

if (Random.Range(0.0f, 1.0f) < 0.5f)

{

Child1.weights[w] = population[AIndex].weights[w];

Child2.weights[w] = population[BIndex].weights[w];

}

else

{

Child2.weights[w] = population[AIndex].weights[w];

Child1.weights[w] = population[BIndex].weights[w];

}

}

for (int w = 0; w < Child1.biases.Count; w++)

{

if (Random.Range(0.0f, 1.0f) < 0.5f)

{

Child1.biases[w] = population[AIndex].biases[w];

Child2.biases[w] = population[BIndex].biases[w];

}

else

{

Child2.biases[w] = population[AIndex].biases[w];

Child1.biases[w] = population[BIndex].biases[w];

}

}

// thêm 2 cá thể con vào quần thể mới

newPopulation[naturallySelected] = Child1;

naturallySelected++;

newPopulation[naturallySelected] = Child2;

naturallySelected++;

}

}

// Thêm các cá thể vào bể lai

private NNet[] PickBestPopulation()

{

NNet[] newPopulation = new NNet[initialPopulation];

for (int i = 0; i < bestAgentSelection; i++)

{

newPopulation[naturallySelected] = population[i].InitialiseCopy(controller.LAYERS, controller.NEURONS);

newPopulation[naturallySelected].fitness = 0;

naturallySelected++;

int f = Mathf.RoundToInt(population[i].fitness \* 10);//số lượng thêm vào bể lai

for (int c = 0; c < f; c++)

{

genePool.Add(i);

}

}

for (int i = 0; i < worstAgentSelection; i++)

{

int last = population.Length - 1;

last -= i;

int f = Mathf.RoundToInt(population[last].fitness \* 10);//số lượng thêm vào bể lai

for (int c = 0; c < f; c++)

{

genePool.Add(last);

}

}

return newPopulation;

}

//sắp xếp lại quần thể

private void SortPopulation()

{

for (int i = 0; i < population.Length; i++)

{

for (int j = i; j < population.Length; j++)

{

if (population[i].fitness < population[j].fitness)

{

NNet temp = population[i];

population[i] = population[j];

population[j] = temp;

}

}

}

}

}

3.3.3 CarController (Bộ điều khiển ô tô)

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[RequireComponent(typeof(NNet))]

public class CarController : MonoBehaviour

{

//khai báo startPosition: tọa độ ban đầu

//khai báo startRotation: góc ban đầu

//khai báo network: mạng neuron

private Vector3 startPosition, startRotation;

private NNet network;

[Range(-1f, 1f)]

public float a, t;//output

public float timeSinceStart = 0f;

[Header("Fitness")]

public float overallFitness;// chỉ số đánh giá tổng thể

public float distanceMultipler = 1.4f;//độ chia nhỏ nhất của chất điểm

public float avgSpeedMultiplier = 0.2f;//độ chia nhỏ nhất của vận tốc trung bình

public float sensorMultiplier = 0.1f;//độ chia nhỏ nhất của 3 vecto cảm biến

[Header("Network Options")]

public int LAYERS = 1;// số lớp ẩn

public int NEURONS = 10;// số neuron trong lớp ẩn

private Vector3 lastPosition;//điểm cuối

private float totalDistanceTravelled;//khoản cách thực di chuyển được

private float avgSpeed;//tốc độ trung bình

private float aSensor, bSensor, cSensor;//3 vecto

//Hàm khởi động

private void Awake()

{

startPosition = transform.position;

startRotation = transform.eulerAngles;

network = GetComponent<NNet>();//tạo neuron network mới

}

//reset network

public void ResetWithNetwork(NNet net)

{

network = net;

Reset();

}

//reset mô hình xe trở về trang thái ban đầu

public void Reset()

{

timeSinceStart = 0f;

totalDistanceTravelled = 0f;

avgSpeed = 0f;

lastPosition = startPosition;

overallFitness = 0f;

transform.position = startPosition;

transform.eulerAngles = startRotation;

}

//Kiểm tra va chạm. khi xe va chạm hàm được gọi

private void OnCollisionEnter(Collision collision)

{

Death();

}

//Cập nhật thông số mỗi lần chạy

private void FixedUpdate()

{

//tính giá trị 3 vecto cảm biến

InputSensors();

lastPosition = transform.position;

//out put được lấy ra từ RunNetwork

(a, t) = network.RunNetwork(aSensor, bSensor, cSensor);

MoveCar(a, t);

//Thời gian di chuyển của mô hình

timeSinceStart += Time.deltaTime;

CalculateFitness();

//a = 0;

//t = 0;

}

private void Death()

{

GameObject.FindObjectOfType<GeneticManager>().Death(overallFitness, network);

}

//Tính chỉ số tổng thể của xe

private void CalculateFitness()

{

//Tính khoản cách thực của xe đã đi được

totalDistanceTravelled += Vector3.Distance(transform.position, lastPosition);

avgSpeed = totalDistanceTravelled / timeSinceStart;//v=S/t

//Tính chỉ số đánh giá tổng thể của mô hình

overallFitness = (totalDistanceTravelled \* distanceMultipler) + (avgSpeed \* avgSpeedMultiplier) + (((aSensor + bSensor + cSensor) / 3) \* sensorMultiplier);

//print("thong so: " + overallFitness);

//print("thoi gian: " + timeSinceStart);

if (timeSinceStart > 20 && overallFitness < 40)

{

print("Death at 103 " + overallFitness + " Time: " + timeSinceStart);

Death();

}

if (overallFitness >= 1200)

{

print("Death at 108 " + overallFitness);

Death();

}

}

//tính giá trị 3 vecto cảm biến

private void InputSensors()

{

Vector3 a = (transform.forward + transform.right);

Vector3 b = (transform.forward);

Vector3 c = (transform.forward - transform.right);

// Tạo tia truyền đi

Ray r = new Ray(transform.position, a);

RaycastHit hit;

// Vẽ tia truyền đi

if (Physics.Raycast(r, out hit))

{

aSensor = hit.distance / 20;

Debug.DrawLine(r.origin, hit.point, Color.red);

}

r.direction = b;

if (Physics.Raycast(r, out hit))

{

bSensor = hit.distance / 20;

Debug.DrawLine(r.origin, hit.point, Color.red);

}

r.direction = c;

if (Physics.Raycast(r, out hit))

{

cSensor = hit.distance / 20;

Debug.DrawLine(r.origin, hit.point, Color.red);

}

}

private Vector3 inp;

//Thực hiện di chuyển mô hình xe

public void MoveCar(float v, float h)

{

//v:gia tốc

//h:gốc

inp = Vector3.Lerp(Vector3.zero, new Vector3(0, 0, v \* 11.4f), 0.02f);

inp = transform.TransformDirection(inp);

transform.position += inp;

transform.eulerAngles += new Vector3(0, (h \* 90) \* 0.02f, 0);

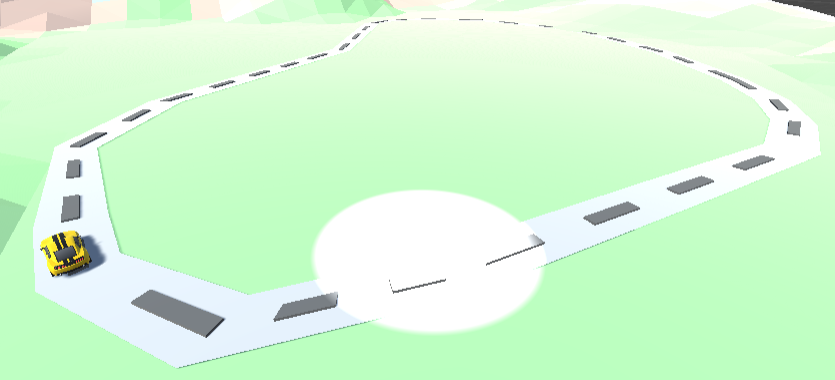
}

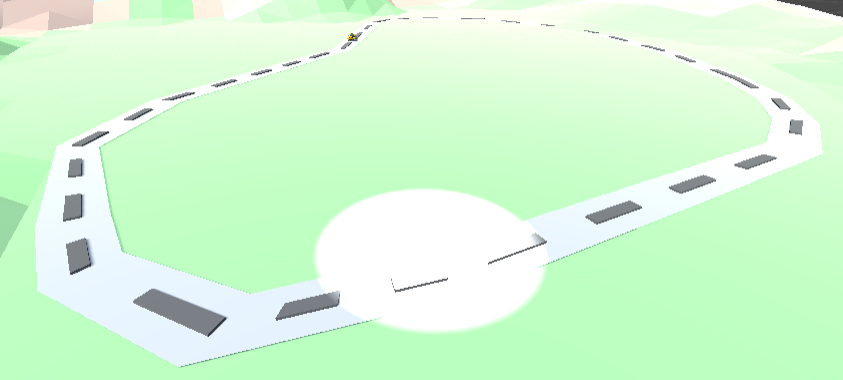
}

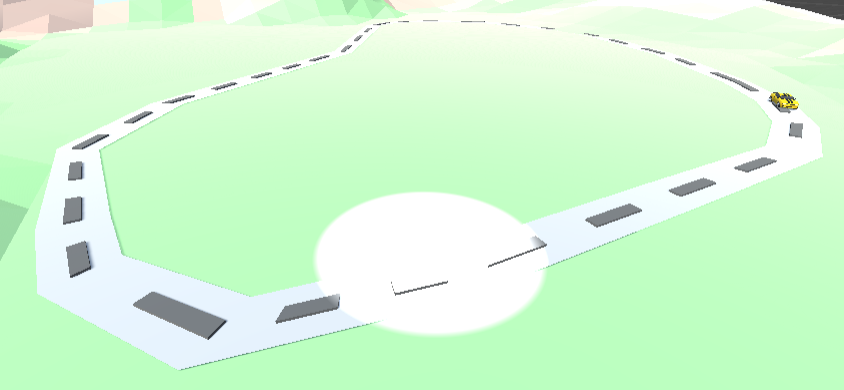
## 3.4 Kết quả thử nghiệm và đánh giá.

3.4.1 Kết quả thử nghiệm

Sau khi nhiều lần huấn luyện thì mô hình thực hiện thành công kết quả mong muốn là đi hoàn tất quảng đường được mô phỏng.







3.4.2 Đánh giá

- Ưu điểm

* Mô hình hoàn thành tốt được việc huấn luyện tăng cường qua mạng neuron.
* Thời gian huấn luyện của mô hình ngắn, ổn định.
* Mô hình có khả năng ứng dụng trong thực tế cao.

- Nhược điểm

* Mô hình còn sơ khai
* Chưa xử lý được việc tránh vật thể, xe lưu thông có trên đường
* Không có khả năng nhận biết biển báo cũng như tín hiệu đèn giao thông