Generative Adversarial Imitation Learning & Racing game

Cheng, Ye A00923048

COMP-8037 & Major Project Proposal
2019 10.3

Table of Contents

1. Student Background

1.1. Education

British Columbia Institute of Technology September 2018 –Present

Computer Systems – Games Development Option

https://www.bcit.ca/study/programs/862bbtech

Projected to graduate with distinction in January 2020

Developed games for IOS, PS4, and PC

Nanjing Vocational College of Information Technology 2013-2016

https://www.njcit.cn/

Software technology

Studied subjects including Computer Science, Math, and English

Skills:

Languages: C++, C#, C, Objective C, Java, HTML, PHP, SQL, UML.

Tools: Microsoft Office, Visual Studio, CLion, Git, XCode

Libraries/APIs: OpenGL, Win32, GLFW, Android, ImGui, SDL

Editors: Unity3D

2. Project Description

Since the development of modern racing games, considerable progress has been made in the intelligence of computer opponents against players, and there have been many attempts. The

human-machine confrontation in the game is becoming more and more real and the gameplay is getting stronger. How can non-player-controlled cars avoid the other cars on the track, the track fence, reach the finish line as fast as possible, and even interfere with the player's car through driving skills? It is the most complex and core thing. The challenge is that NPC racing should be able to perform high-speed real-time calculations and predictions for the road conditions, and let the car react appropriately through analysis and decision making. We extracted this core processing and computational problem, mapping it to a simple racing game using Unity, applying certain analytical decision algorithms, and finally enabling the car to drive automatically on the game's track.

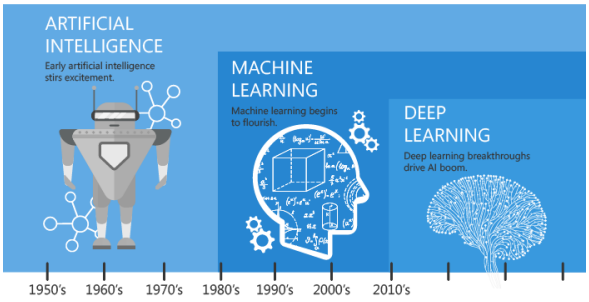
In terms of calculations and algorithms, there are many ways to implement such automatic driving methods, but in the past three or four years, the effect is quite good. The more popular method is to use the generation of confrontation network simulation learning developed by convolutional neural networks, with practical problems. Some other auxiliary means to deal with.

This project focuses on verifying the feasibility and practical effect of the GAN algorithm applied to autonomous driving. As an undergraduate graduation project, the so-called autopilot is achieved to the extent that the car can be automatically controlled by the computer, along the track, to avoid collisions. The walls on both sides of the track should also try to avoid colliding with other vehicles on the track by slowing down and changing lanes.

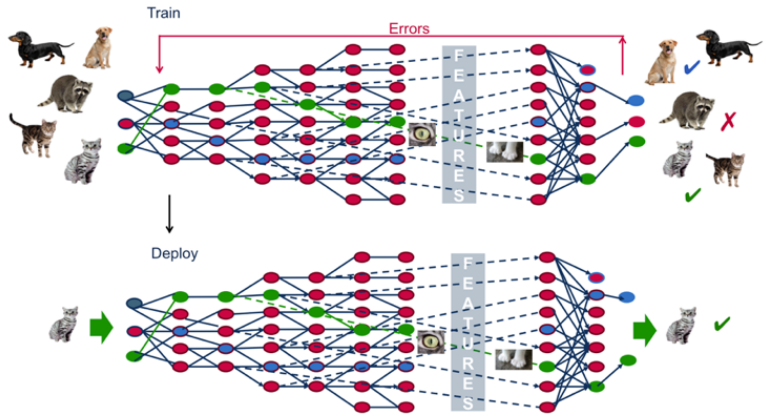
This project uses an imitation learning algorithm based on generating an anti-network to implement the autopilot function. In order to reflect better learning performance and accelerate learning, it also considers other auxiliary learning and provides some human driving. The data is pre-trained, which should result in better generalization performance with less data.

3. Problem Statement and Background

Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input. It is an algorithm that uses artificial neural networks as a framework to perform characterization learning on data. There are several deep learning frameworks, such as neural networks, convolutional neural networks, and recurrent neural networks. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

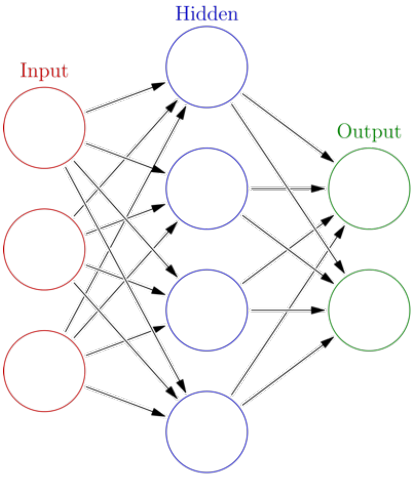


To put it simply, deep learning is to treat what the computer wants to learn as a lot of data, throw this data into a complex, multi-level data processing network (neural network), and then check what is processed by this network. Does the result data meet the requirements? If it meets the requirements, the network will be kept as the target model. If it does not meet the requirements, the parameter settings of the network will be adjusted repeatedly persistently until the output meets the requirements.



Deep learning has been widely used in computer vision, speech recognition, natural language processing, audio recognition, bioinformatics, and other fields, and achieved good results. Artificial neural network is a neural network that simulates the human brain in order to implement artificial intelligence-like machine learning technology. The neural network in the human brain is a very complex organization. There are an estimated 100 billion neurons in the adult brain.

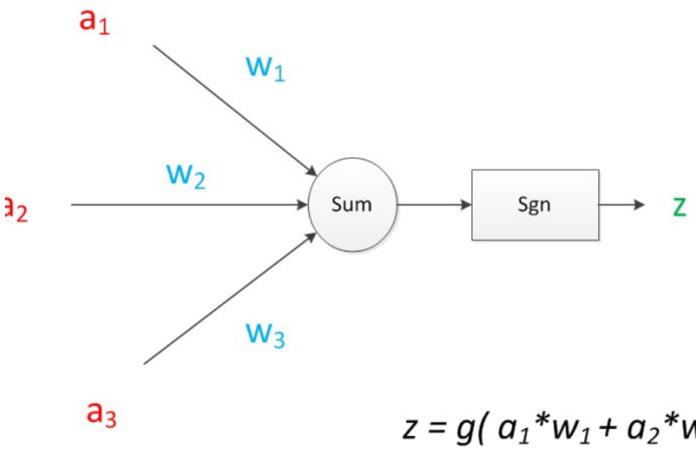
Let's look at a classic neural network. This is a neural network with three levels. The red is the input layer, the green is the output layer, and the purple is the middle layer (also called the hidden layer). The input layer has 3 input units, the hidden layer has 4 units, and the output layer has 2 units.



When designing a neural network, the number of nodes in the input layer and the output layer is often fixed, and the intermediate layer can be freely specified. The key in the structure diagram is not circles (representing "neurons"), but connection lines (representing connections between "neurons"). Each connection line corresponds to a different weight (its value is called a weight value), which needs to be trained. The connection is the most important thing in a neuron. Each connection has a weight.

We use a for input and w for weight. A directed arrow indicating the connection can be understood as: At the beginning, the size of the transmitted signal is still a, and there is a weighting parameter w in the middle. The weighted signal will become a * w, so at the end of the connection, the size of the signal will become a * w.

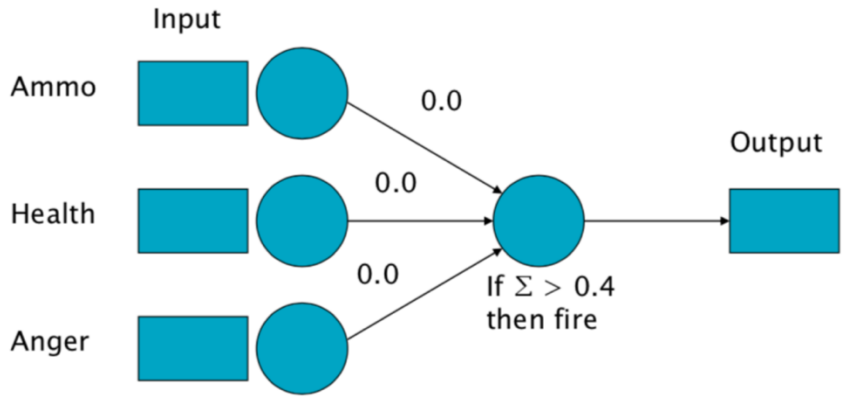
If we represent all the variables in the neuron graph with symbols and write the calculation formula of the output, it is the following figure.



$$z = g(a_1 * w_1 + a_2 * w_2 + a_3 * w_3)$$

Visible z is a linear weighting on the input and weights and superimpose the value of a function g. The function g is a sgn function, that is, a symbolic function. This function outputs 1 when the input is greater than 0, otherwise, it outputs 0. For example, the picture below if all a*w values

are greater than 0.4 then the return value is 1 and vice versa is 0(threshold is 0.4)

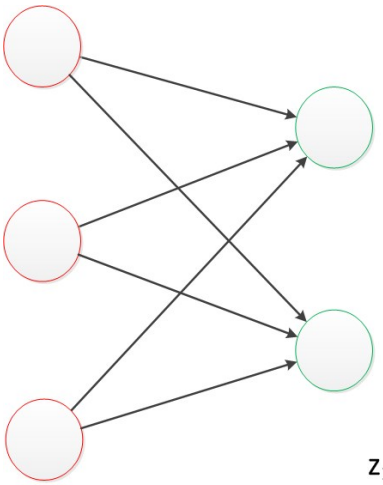


After that, we continue to train and update the weight value using gain term (η), train.out-actual.out, and Node output.

$\Delta w = \eta * (\text{train. Out} - \text{actual. Out}) * \text{node output}$

Suppose that the target we want to predict is no longer a value, but a vector. For example, [2,3].

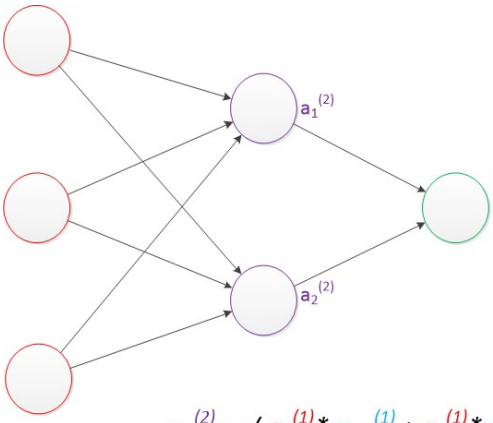
Then you can add another "output unit" in the output layer. (the picture below)



$$z_1 = g(a_1 * w_1 + a_2 * w_2 + a_3 * w_3)$$
$$z_2 = g(a_1 * w_4 + a_2 * w_5 + a_3 * w_6)$$

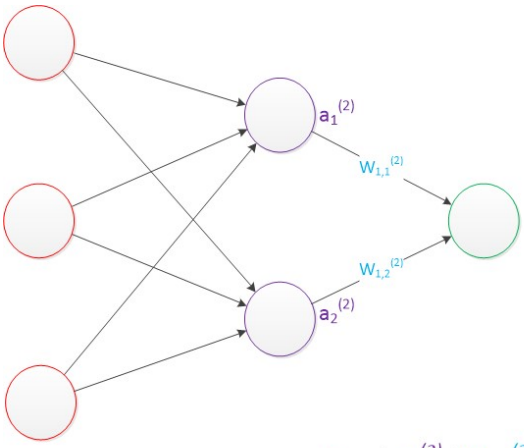
A single-layer neural network cannot solve the XOR problem. But after adding a calculation layer, the two-layer neural network can not only solve the XOR problem and has a very good nonlinear classification effect.

The two-layer neural network includes an input layer, an output layer, and an intermediate layer. At this time, the intermediate layer and the output layer are both calculation layers. We extend the single-layer neural network from the previous section and add a new layer on the right (containing only one node).



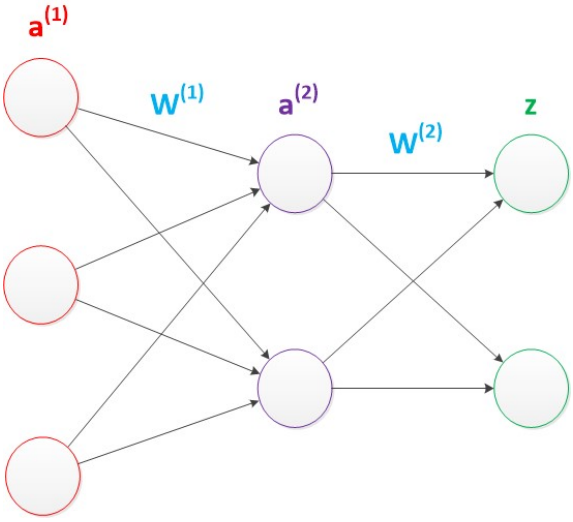
$$a_1^{(2)} = g(a_1^{(1)} * w_{1,1}^{(1)} + a_2^{(1)} * w_{1,2}^{(1)} + a_3^{(1)} * w_{1,3}^{(1)})$$

$$a_2^{(2)} = g(a_1^{(1)} * w_{2,1}^{(1)} + a_2^{(1)} * w_{2,2}^{(1)} + a_3^{(1)} * w_{2,3}^{(1)})$$



$$z = g(a_1^{(2)} * W_{1,1}^{(2)} + a_2^{(2)} * W_{1,2}^{(2)})$$

We use vectors and matrices to represent variables in the hierarchy. a (1), a (2), z are the vector data transmitted in the network. W (1) and W (2) are matrix parameters of the network. As shown below.



The calculation formula is as follows:

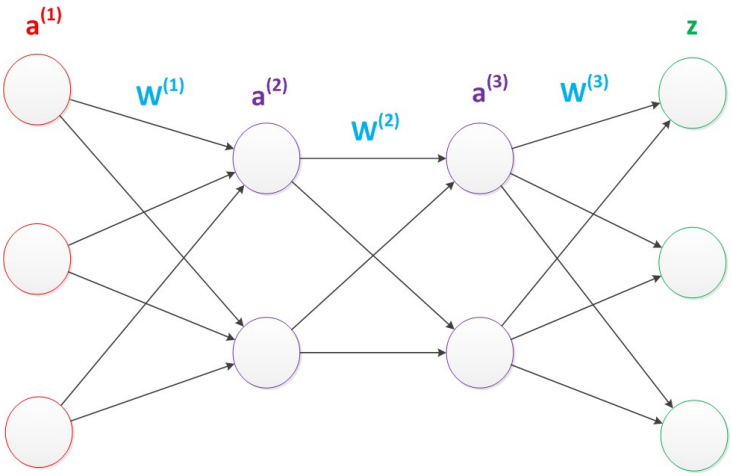
$g(\mathbf{W}^{(1)} * \mathbf{a}^{(1)}) = \mathbf{a}^{(2)}$

$g(\mathbf{W}^{(2)} * \mathbf{a}^{(2)}) = \mathbf{z}$

It should be noted that in the two-layer neural network, we no longer use the sgn function as the function g, and we use the smoothing function sigmoid as the function g. We call the function g an active function.

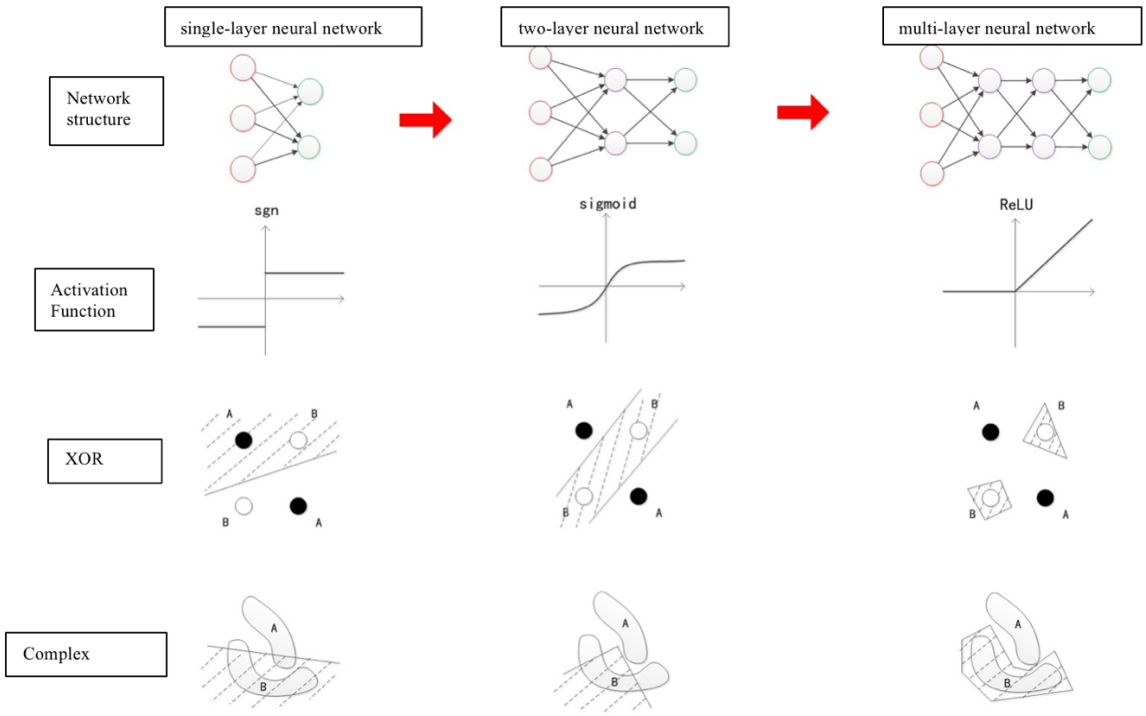
We continue the two-layer neural network approach to design a multilayer neural network. Behind the output layer of the two-layer neural network, continue to add layers. The original output layer becomes the middle layer, and the newly added layer becomes the new output layer, so you can get the following figure.

When it comes to multilayer neural networks, through a series of studies, it has been found that the ReLU function works better when training multilayer neural networks.



In a multilayer neural network, the output is also calculated in a layer-by-layer manner. Starting from the outermost layer, after calculating the values of all cells, continue to calculate the deeper layer. Only after the values of all units in the current layer have been calculated, the next layer is counted. It's a bit like computing is moving forward, so this process is called "forward propagation".

From a single-layer neural network to a two-layer neural network to a multi-layer neural network, the following figure illustrates that as the number of network layers increases, its performance is getting stronger and stronger.

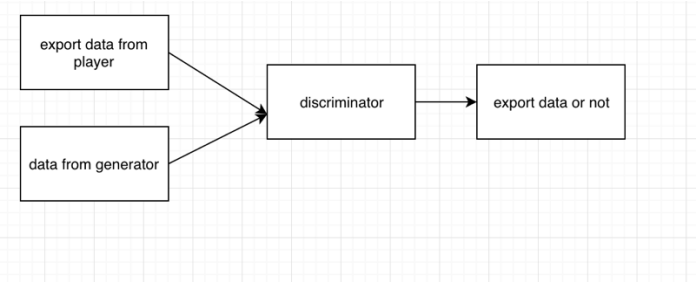


Imitation learning refers to learning from the examples provided by teachers (expert data), and generally provides decision data of human experts (in this project is players). Each decision data contains a sequence of states and actions. Then, pick all state action to construct a new set. Then, we can use state as feature and action as label to classify (for discrete action) or regress (for continuous action) to get the best model.

Gan (Generative Adversarial Network) algorithm will generate an anti-network consisting of two opposing neural networks (discriminator and generator). Generator networks can generate data. Discriminator networks can discriminate whether the data is expert data. The two networks compete against each other to produce the ideal data.

In this project, the player's data is collected as expert data. The generator will generate trajectory data for the AI car running and make the trajectory data as close as possible to expert data. Then,

the generated data and expert data will pass to the discriminator. The discriminator will determine if the trajectory data generated by the expert data or not (is it possible for the car to run normally?), and the generator wants the discriminator to judge the generator's data is expert data.



The core issue involved in automatic driving is automatic road recognition, perception and decision making.

In the racing game, during the driving process, the vehicle needs to judge the current road conditions, such as whether there is a vehicle in front and rear, the relative speed of the vehicle, and the driving direction. What strategy should be adopted to ensure that no collision occurs? Furthermore, can the own vehicle make a certain prediction through the real-time road conditions, the new road conditions that will be encountered in the front, and the interference of the player's car to the own vehicle, and take reasonable countermeasures in advance?

We break the problem down into two parts, the project implements the first part first. Then, discusses the solution of the second part:

In the first part, the project will implement a simple but realistic car physics engine. As the basis of autonomous driving, the car first needs to have basic functions such as acceleration, deceleration, forward, backward, and steering, as well as the effects of ground friction and air resistance. According to the momentum theorem, for collision, it is also necessary to distinguish the collision force to achieve and true. The world's similar impact slows down and is knocked

out. This part will be realized by modeling Newton's law of motion and computing the vector. There will be a certain section in the project to explain in detail how to model and how to program the vector.

The second part is the core of the project and the part that is focused on discussion and implementation. To achieve an NPC-controlled car, you can drive automatically and avoid hitting the track fence and the player-controlled car. Then, just that is not enough, the car will only follow an optimized path, which is too rigid and boring. In this project, we try to realize the self-learning function of the car to achieve the purpose of simulating human players. In this way, the player will always compete with an opponent whose strength is similar. As the player level continues to rise, the NPC racing car It will also constantly improve your driving level, which is a fun racing game.

4. Complexity

For B. Tech students, this project has great implementation difficulties. Due to lack of studying time, physical and mathematical knowledge and long programming experience, the following skills may not be available:

1. Skilled programming skills. Skilled in using C# to program in the Unity environment.
2. Although this project will use Unity for physics, it needs clear physics, mathematical concepts, and the ability to model and program the corresponding concepts. For example, the acceleration, deceleration, and collision of a car require the use of Newton's second law, the law of conservation of momentum, the representation and operation of vector algebra, matrix calculations, etc., and they need to be modeled and coded.

3. Basic knowledge of mathematical analysis, knowledge of advanced algebra and probability and statistics, basic knowledge of machine learning, and a certain understanding of neural networks. It may also require courses in mathematical logic and discrete mathematics.

The above-mentioned questions need to be completed after the B. Tech course.

The real difficulty of this project is:

1. How to combine and model the GAN simulation learning with the auto-driving problem of the racing agent.

2. How to adjust the parameters and use TensorFlow to train the agent, so that the NPC-controlled racing car has the driving characteristics of the human opponent.

3. How to make this project can train the cars in a short time, less sample space with good training results.

In addition, in order to complete this project, I should have some understanding of game artificial intelligence, such as potential function, clustering, pathfinding, genetic algorithm.

5. Scope and Depth

This project will use neural networks to make a racing game. It will Use the GAN network to simulate learning, the generation and confrontation parts are all implemented by neural networks. The game is divided into beginner difficulty, normal difficulty, and master difficulty three difficulty levels. This game will support keyboard operation. It will have simple controls that are easy to learn how to play. When manipulating the car, the four buttons of the up, down, left and right control the movement of the racing agent. It will use artificial intelligence, which can help make the game more challenging and provide a good game experience. The game will be suitable for all ages and be attractive to both men and women. The project will use rich and

rhythmic sound effects, music and exquisite models with rendering. The physics part of the game will simulate the real world to provide a good gaming experience. Players can observe the progress of artificial intelligence and compete with artificial intelligence on different levels of difficulty. In this project, there will not be many racing maps. If time permits it may have some different maps and car models. Due to lack of time, this project may not have various styles of maps and car models.

6. Test Plan

It will use some test tools such as NUnit, Visual Studio, and Unity to do the unit testing. I will use unit tests to test whether the input direction buttons and output direction are consistent. Also, it will test the after input the collision data and the performance of the car whether it meets the requirements. In addition, I will test TensorFlow can train three different levels (easy, medium, and hard level) of autonomous driving techniques, and players can pick the different levels of difficulty to play the game. The testing will find errors and bugs in the project and ensure there are no serious bugs affect the operation of the project. It will make sure the project does everything in the right way and achieve the expected results.

The following are the main items that need to be tested

1. Test of racing driving game content:

1) The car can control the car to move forward and backward through the up, down, left and right buttons.

2) After the car hits the track wall, it will stop and the car body will be damaged.

3) After the car arrives at the finish line, stop and the game prompts to the end.

2. Performance test after simulated learning:

1) The car can still drive automatically, from the starting point to the end.

2) After the player selects the difficulty, the system can automatically match the driving level similar to the player's driving skills, and automatically drive the vehicle to the end point according to this level.

Test item	Function	Tool	Verification	Passing the standard
1.1	Unit test + observation verification		Write a unit test to test whether the data of the input direction of the input button and output is always consistent with the requirements; and observe whether the actual performance of the racing model in the game is in line with expectations.	Press: Car advance; press: car back; press left, front-wheel turns left; press right, front-wheel turns right. When the car is stationary: When pressing up and down, there is a significant acceleration process until it accelerates to the highest speed. When the car is moving: When the up and down buttons are pressed, the speed is maintained after the acceleration to the highest speed. When pressing left or right, the car turns in a direction corresponding to a certain turning radius. Release the button to start decelerating or stop turning.
1.2	Unit test + observation verification	NUnit Visual Studio Unity	Write a unit test to test whether the performance of the car meets the requirements after entering the collision data. Also observe whether the actual performance of the car model in the game is in line with expectations.	Regardless of the speed and angle of the car, the car hits the track wall. According to Newton's laws of mechanics, the car appears to be immediately hit and stopped, and there are obvious injuries and bumps on the body.
1.3	Unit test + observation verification	NUnit Visual Studio Unity	Write a unit test to test whether the performance of the car meets the requirements after entering the end point data. Also observe whether the actual performance of the car after the arrival of the car is in line with expectations.	After the car arrived at the end of the track, the car was forced to slow down and the system prompted to reach the end
2.1	Observation verification	Visual Studio Unity	Write code to test whether the car controlled by the system can be	Under the premise of meeting the requirements of 1.3, the car can be controlled by the system,

			started at the starting point, whether the performance of the car meets the requirements; whether it meets the requirements of 1.3 after reaching the end point. Also observe whether the performance of the actual racing model in the game is in line with expectations	automatically accelerate from the starting point to the highest speed, can quickly pass the curve, avoid hitting the wall, and finally reach the endpoint quickly.
2.2	Observation verification	Visual Studio Unity TensorFlow Sharp	Write code to test directly in the game whether the car has obviously adopted different driving strategies to achieve the function of 2.1; by inputting different training data, directly observe whether the test system can train the driving strategy that matches the human opponent driving technology.	TensorFlow is able to train three different levels of autonomous driving techniques in the middle and upper levels through imitative learning. The three levels should cover the level of game technology at the level of human primary, ordinary, and specialized. The three levels of driving strategies are significantly different from each other and are clearly proficient and advanced ascending incremental steps. After the player chooses different difficulty levels of the game, it can clearly understand that the driving technology of the system-controlled racing car first meets the driving strategy of this difficulty level. The second is the human driving level that meets this difficulty. The most intuitive is that there is a win or lose and winning or losing will not be particularly large.

Test case:

Test item	Test Case
1.1	Press the up button, when the maximum speed is not reached, gradually accelerate straight ahead at a constant acceleration until the highest speed, maintaining the highest speed. When the key is pressed, when it is not at the highest speed, it will gradually go back straight down at a constant acceleration until the highest speed, maintaining the highest speed. Press the left and right button to turn the wheel to the left or right, but the car does not move. Press the up or down button or the left and right buttons at the same time: no action.

	Press the up and direction keys at the same time. When the maximum speed is not reached, gradually accelerate with a constant acceleration. At the same time, turn to the left/right direction with a certain turning radius until the highest speed, maintain the highest speed and maintain the steering. Press the arrow keys at the same time. When the maximum speed is not reached, gradually accelerate with a constant acceleration. At the same time, turn to the left/right direction and retreat according to a certain turning radius until the highest speed, maintain the highest speed and maintain the steering. Any other use cases that do not meet the above rules are considered to be untested.
1.2	When the vehicle collides with the track wall, consider four forces: the forward power of the car, the friction between the wheel and the ground, the air resistance, and the frictional resistance of the wall to the car body. According to Newton's laws of mechanics, the car is considered as the mass point, the above four forces are vector superimposed to calculate the final force, and then the car performs corresponding acceleration and deceleration along the direction of the force. After the car collides with the wall, it will not be tested according to the direction and size of the force when it is directly still, through the wall, and rubbed against the wall. After the car collided with the wall, the collision part did not show visual damage, and it was considered that it did not pass the test.
1.3	When the car arrived at the end, the operation of the car immediately failed, and the system controlled the car to automatically decelerate and finally stop. Wait for other cars to reach the end. When all the vehicles arrive at the finish line and stop, immediately show the end of the game and give the results, such as the first few. It can still be operated after reaching the end point, or the car does not decelerate without stopping, and stops immediately without deceleration. The end of the game is prompted without waiting for other vehicles to reach the end. The ranking was not calculated correctly after the game. Both are considered to be untested.
2.1	On the basis of 1.3, after the start of the game, the car accelerates at the starting point, enters the track and starts the game. Did not enter the track, or did not accelerate after entering the track, or immediately at the highest speed without the acceleration phase. Both are considered to be untested.
2.2	After data collection and calibration, use TensorFlow and Gan model to learn and train, and finally put the trained model into unity. Players choose games of different difficulty, and computer-controlled opponents can compete with corresponding difficulty. Confrontation, and there are wins and losses, and winning or losing is not static, and the gap between winning and losing is not too big. If the computer opponent apparently uses a different difficulty than the player's choice to play the game, or play the game on an unreasonable route, or play the game in a obviously overly clumsy and overly confusing strategy, it is considered as not passing.

In the data collection and analysis part I will divide into the following steps.

1. This project will create two Brains (Teacher and Student). Teacher Brian will use type player Brain.
2. Customize the racing class. This class implements its own data collection function Collect Observations by inheritance the Agent class. Then, I will use the AddVectorObs to record the car's current position, acceleration, speed, end distance, track position, and other data.
3. Re-implement the AgentAction function to express the action, rotation, etc. of the racing agent we need for the racing action.
4. The custom Agent class attach with the custom External Student class brain.
5. Set Student's Brain in python / trainer_config. yaml. set the trainer parameter is imitation. Brain_to_imitate is set to Teacher's Brain name. Batches_per_epoch sets how many trainings are performed at each moment. Increase max_steps to make training for a longer period of time
6. Training through command tool and train in the editor with python3 python / learn.py -train -slow.
7. Operate Teacher to observe the output
8. Observe Student for imitations
9. Once Student has a better imitation, we can the training
10. Put the generated binary file (. bytes) back into the Unity project, change the Brain type of student to Internal, and re-run to see the results of the simulation learning training
11. Let the system NPC car race and observe whether the car's driving path is similar to the player's style.

This project also will test Gan model in three parts (train/validation/test set). I will use a data set, one part of which is used for training, the other part can be used for verification, which is called training set and verification set respectively, and the ratio of training set to verification set is 7:3.

Then, I can take another brand-new dataset to verify the training results, which is called test dataset

The test set can judge the quality of a model based on the error (generally, the difference between predicted output and actual output).

7. Methodology

The project will use neural networks to train the AI car in the game. It will use C# as language and Unity 3D as game engine. Also, it will use TensorFlow Sharp API, TensorFlow Framework, and unity ml-agent. The data is represented as Json. After the development of the game body is completed, collect enough player game data, then start to set up the environment, use the external model of the unity ml-agent to call TensorFlow for training, use TensorFlow Sharp for programming docking, and assist the necessary python script. Then, using the internal mode of Unity ml-agent test the training model and related data. The NPC car is controlled by the computer. It will automatically drive one circle and then compete with the human players. Observe whether the results are in line with expectations. If necessary, it may assist in other reinforcement learning methods. The algorithm will use genetic algorithm and pathfinding algorithm and other algorithms. In addition, this project software development model is rapid prototype model.

8. System/Software Architecture Diagram

It is a racing game. Game design standard and machine learning are part of this game. Game design standards include modeling of the real world, such as: vector math, acceleration, collision, velocity. The right side is artificial intelligence, which includes game data collection (such as the player's accelerated turn preferences, avoidance preferences), racing track data, and behaviors

imitated from the player, such as reinforcement learning and generation of anti-network related models and algorithms. Unity is the entire development environment and editor for implementing these functions. It also stores and localizes the relevant data in game design standard and machine learning (such as player behavior, racing data), because the data belongs to the external system. Unity also uses the ml-agents plugin, TensorFlow sharp plugin, communicates with TensorFlow, reads, writes, and stores files in ckpt and pb formats required by TensorFlow and training model.

Unity's ML-Agent component implements data collection and training through Brain, Agent, and Academy. ML-Agents mainly include the following parts:

1. Learning Enviroment: including game scenarios and characters

a. Agents: tied to GameObject, giving observations and performing actions; must correspond to a Brain.

b. Brains: have strategy, receive observations and give actions; can correspond to multiple Agents.

c. Academy: Control global variables and arrange Brain update order

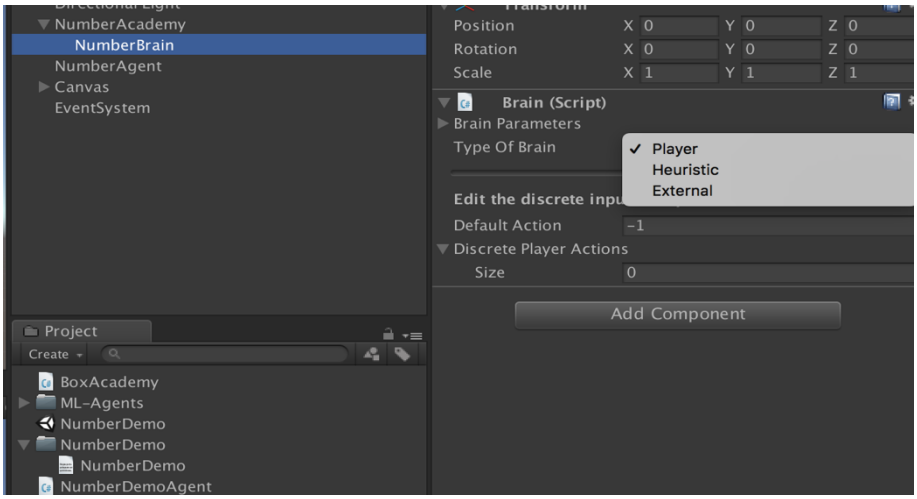
Brain type:

1. External: The strategy is input by the external python API

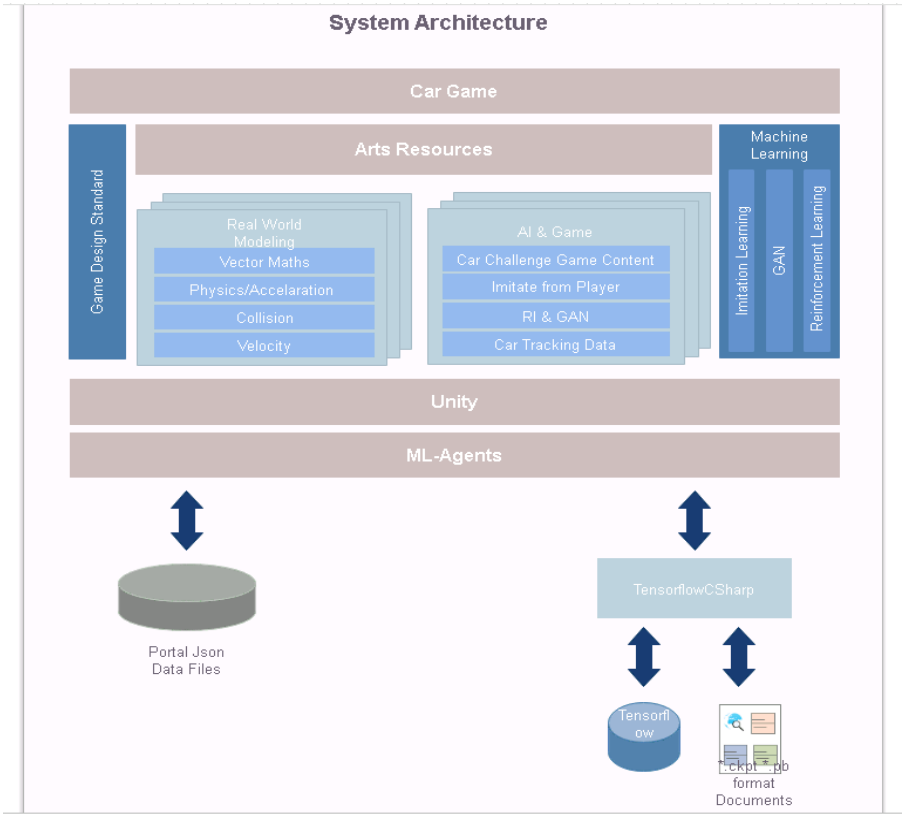
2. Internal: The strategy is determined by the internal TensorFlow model

3. Player: The strategy is entered by the player with the keyboard and mouse

4. Heuristic: The strategy is determined by hard coding; it can be used for testing, comparison.



The AI part uses the ML-Agent component, which is mainly used to attach Agent scripts to racing GameObject objects and rewrite member functions as required and attach defined brain class and attach the defined brain and regulate the environment through the Academy. Json reads and writes, serializes, and implements Newtonsoft class library. The art resources to fbx format, Sprite sprite.



9. Innovation

The key innovation of this project is to use GAN to generate an imitation learning method against the network to learn the game mode of human players and improve the driving skills and driving strategies of computer-controlled NPC racing cars to produce better and more realistic confrontation effects. The player's immersion, enhance the game's playability and adhesion. In addition, almost racing game do not use GAN to make the AI car, and it is also an innovation. Some racing car racing intelligence is often based on a fixed track map, by adjusting the car's own acceleration and turning radius, driving speed, constant acceleration and deceleration in the track, collision with the track wall, Driving strategies such as cornering in corners to achieve confrontation with the players, so that after playing for a period of time, players can explore the fixed driving route and mechanical driving strategy of the NPC car, which will be boring, the game Freshness will also disappear quickly.

If you add a driving strategy after deep learning, NPC cars will not always use the same strategy to fight against each other. You can use the dynamic change of tit-for-tat to win the game's game habits, then the player will experience the real game. This enhances the player's game interest.

In addition, the innovation of this project lies in the engineering implementation method. Generally speaking, the game uses neural network for deep learning, and the algorithm that dynamically matches the player's technical level is relatively small. Usually in large 3A games, especially in SLG strategy and business simulation games, it is not common. For small casual games. Limited to R&D strength, development environment, technology stack and time period, it is less likely to be used by small and medium-sized entrepreneurial teams, and the specific implementation companies are not publicly available and are rarely seen. It is likely that you will

need to write data collection tools, training scripts, game intelligence logic, and so on from scratch.

The implementation of this project uses Unity's ml-agents, which was born only two years ago. Its original purpose is to provide a smart agent to implement a common artificial intelligence game framework. It provides a complete and versatile machine learning programming framework for games of all sizes and can be compared through several machine learning algorithms (intensive learning, course learning, neural networks, etc.) provided by it. It's easy and fast to train the model you want with a specific learner. Through the industry's most commonly used game development methods and technology stacks, individuals or small teams can quickly get started and implement this functionality.

10. Technical Challenges

The technical challenges of this project have physical and mathematical. it is necessary to have a certain understanding of Newton's laws of mechanics under the premise of mastering the basics of C#. It is possible to programmatically implement vector mathematics and correctly implement the acceleration model based on this. Also, GAN is an application evolution of neural networks. Our AI course will learn the basic concepts of neural networks. This will give me some understanding of imitation learning, but what is certain is that the practical part of the project may not have been learned in class. In addition, such as using TensorFlow Sharp to dock native python scripting systems, data collection, and transformation, data format analysis and read and write processing. There is my first time to use, and this is a challenge.

11. Development Schedule and Milestones

The first part of the racing game content development will be 2 months. I will finish the project proposal.

The second part of data acquisition. It will include art resources collected and processed for 1 week. GAN training, TensorFlow programming docking, Unity test verification they will take 3 months. Also, data collection will take 1 week, GAN adjustment will take 3 weeks. Programming docking and verification test will take for 4 weeks. The major proposal will take 2 months.

Car Game		
Milestone	**Effort**	**Hours**
UML use case diagram	Low	6
UML class diagram	Low	6
UML sequence diagram and activity diagram	Medium	16
Development environment set up	Low	6
Vector modelling of velocity	Medium	16
Mechanical model modelling	Medium	16
Racing 3D model import	Low	8
Track map model processing	Medium	16
Racing steering and acceleration programming	High	32
Racing collision logic programming	High	32
Art resources and code integration	Medium	16
Test	Medium	16

Total hour of milestone car game car	186	
AI		
Milestone	**Effort**	**Hours**
Write player data record script	Low	6
Unity machine learning agent configuration	Medium	16
Script TensorFlow sharp	Medium	16
Model training	High	48
Code NPC game logic	High	24
Adjust machine learning parameters and feedback data	High	48
Conduct one-on-one competition with players to test the training results	Medium	16
Total hour of milestone Car game car	174	
Total hours of project	360	

12. Deliverables

This project can finally deliver a complete Unity project with all source code and art resources. Also, it will have player driving data in Json format and the savedModel data in Protobuf format may also contain Ckpt files. In addition, it will have a final project report and project proposal.

13. Conclusion and Expertise Development

This major project will improve my expertise in artificial intelligence. I have not written a whole game with artificial intelligence and I expect that implementing neural networks will be

complicated. The project requires learning and using many new algorithms, and also involves using OpenGL, which will enhance my ability to write rendering code. Also, I will learn to use TensorFlow to learn to collect data and train models. A deep understanding of what neural networks and imitative learning can do, and in the future can complete more complex deep learning tasks independently. This project will use Unity 3D to make a racing game. Therefore, this project will improve my ability to use Unity3D and enable me to use unity skillfully. While I learned Unity3D, OpenGL, physics, and artificial intelligence in my B. Tech. and CST courses, this project will enable me to extend what I learned.

14. References

Artificial neural network. (2019, September 19). Retrieved October 1, 2019, from https://en.wikipedia.org/wiki/Artificial_neural_network.

TensorFlow Pros and Cons - The Bright and the Dark Sides. (2018, September 15). Retrieved October 8, 2019, from https://data-flair.training/blogs/tensorflow-pros-and-cons/.

Haskins, A. (2018, July 23). TF Jam - Shooting Hoops with Machine Learning. Retrieved October 8, 2019, from https://medium.com/tensorflow/tf-jam-shooting-hoops-with-machine-learning-7a96e1236c32.

Saunders, A. A. (2017, November 6). Top Five Use Cases of TensorFlow. Retrieved October 8, 2019, from https://www.exastax.com/deep-learning/top-five-use-cases-of-tensorflow/.

Neural network racing cars around a track. (2019, January 4). Retrieved October 8, 2019, from https://www.youtube.com/watch?v=wL7tSgUpy8w.

如何丝滑地入门神经网络？写个 AI 赛车游戏，只训练 4 代就能驾驶. (2019, February 7) [How to code a sample neural networks example? Write an AI racing game and train for only 4 generations]. Retrieved October 8, 2019, from https://new.qq.com/omn/20190207/20190207A0B5Y1.html.

Chung, T. (2018, March 24). Setting up a Python Environment with Unity ML-Agents and TensorFlow for macOS. Retrieved October 8, 2019, from https://medium.com/@indiecontessa/setting-up-a-python-environment-with-tensorflow-on-macos-for-training-unity-ml-agents-faf19d71201.

TensorFlow Core. (n.d.). Retrieved October 8, 2019, from https://www.tensorflow.org/tutorials.

Juliani, A., Berges, V.-P., Gao, Y., Henry, H., Mattar, M., Lange, D., & Vckay, E. (2018, September 7). Unity: A General Platform for Intelligent Agents. Retrieved October 31, 2019, from https://arxiv.org/abs/1809.02627.

Morton, A., Wheeler, J., Kochenderfer, T., & Kochenderfer, M. (2017, January 24). Imitating Driver Behavior with Generative Adversarial Networks. Retrieved October 31, 2019, from https://arxiv.org/abs/1701.06699.

Lin, J., Zhang, Zongchang, Jiang, Chong, & Hao, Jianye. (2019, September 17). 基于生成对抗网络的模仿学习综述. [A review of imitation learning based on GAN]. Retrieved October 31, 2019, from https://kns.cnki.net/KCMS/detail/11.1826.TP.20190917.1358.002.html?uid=WEEvREdxOWJmbC9oM1NjYkcyQzZ4VHpvVFg2Y25zbDNQY1FkN2xoTWxmVFA&v=MTExODhGeUhrVzcvSkpGWT1MejdCZHJHNEg5ak1wbzVDWk9zUFl3OU16bVJuNmo1N1QzZmxxV00wQ0xMN1I3cWVidVpt.

Lőrinczs's, Z. (2019, September 19). A brief overview of Imitation Learning. Retrieved November 24, 2019, from https://medium.com/@SmartLabAI/a-brief-overview-of-imitation-learning-8a8a75c44a9c.

Generative adversarial network. (2019, November 22). Retrieved November 24, 2019, from https://en.wikipedia.org/wiki/Generative_adversarial_network.

写给机器学习工程师：如何测试 TensorFlow 模型.[Writing to Machine Learning Engineers: How to Test TensorFlow Models] (2018, April 17). Retrieved December 8, 2019, from https://juejin.im/entry/5ad5a055f265da2384411546.

神经网络浅讲：从神经元到深度学习. [Talking about neural networks: from neurons to deep learning](n.d.). Retrieved December 8, 2019, from https://www.cnblogs.com/subconscious/p/5058741.html.

15.Change Log

2019. Nov 5th- Initial writing proposal.

2019.Nov 24th- Minor revisions to improve clarity in section 1(Student Background Page2), 3(Problem Statement and Background Page 4),4(Complexity Page7),6(Test Plan Page9),8(System/Software Architecture Diagram Page 12),9(Innovation Page13).

2019. Dec 9th- Minor revisions to improve clarity in section3(Problem Statement and Background Page 4), 6(Test Plan Page15), and section14 (References page28).