Performance Evaluation Report

Table of Contents

Differences to Brief

Style of Network

When creating the neural network, the initial thought was to use backpropagation and gradient descent with given data sets to teach the network. This was briefly touched upon in the initial brief, but the implementation was not working as intended and was scrapped. This was supposed to be paired with a genetic algorithm which would improve the learning process of the neural network. The genetic algorithm system works as intended and that was the new setup of the neural network. Another neural network structure that I looked at was the NEAT network which attempts to create its own network by adding or removing nodes and connections to the hidden layers which would work perfectly for a modular package. However, the depth of research for this structure was not as intensive as a gradient descent network.

Modularity Issues and Coding Background

A major issue that came from implementation occurred when the system was being made modular. Since AI is a very broad topic and many different games will need them, making a modular neural network was very difficult. A way to combat this is to make a neural network specific for a game and the user can freely edit variables in the inspector that will change how the neural network is developed. This can be changing the number of nodes per hidden layer or increasing the amount of ray casts for inputs. Instead, I made the neural network system “code reliant”. This means the core functionality of the network remains, but the user must decide on their own inputs and outputs through writing their own code. Users must decide how many inputs, outputs and nodes in the hidden layer there are and feed the correct amount of inputs into the system and take out the correct amount of outputs and have the network perform actions based on the outputs. This made

the core neural network modular but required the user to have coding knowledge. This can be intimidating for developers that want a self-learning AI system but have no coding knowledge. Specialised neural network packs will be more appealing to those users.

Manipulation of Variables

I also noted that many of the variables must be manipulated in the inspector. This ultimately changed and the only variables of the core network systems that can be changed in inspector via a scriptable object, is the amount of inputs, outputs and nodes in the hidden layer. This is stated in the above paragraph where functionality for the game had to be separate from the network itself. This also addresses the base script having functionality for the demo. The functionality was removed from the base script to ensure modularity could exist for this package.

Overcomplicated Fitness Score

Another issue that was found after some attempts at teaching the neural network was the fitness score created in the brief. This fitness score was too specific and overcomplicated the system. Each time another factor that I wanted to be monitored came up, I added it to the fitness score. This lead to the networks not learning as intended and they mutated to cheese the fitness score system implemented. To overcome this, the fitness score system was reduced to monitoring the simple statistics and compare those.

Activation Function

The activation function that was initially going to be used was the sigmoid function. After many attempts at getting a vehicle to work with it, I found it easier to use another activation function. This was the tanh function. This also opened up a modularity issue where the user was only given the sigmoid function as an available choice. Many different activation functions are now implemented for usage. The user just needs to specify which one they want to use in the script.

Performance and Comparison to other implementations

CPU vs GPU

The performance of the system is very inefficient currently because a lot of the calculations are done on the CPU. This was very noticeable when there were many agents in the scene doing their calculations and acting upon them. This impacted training times heavily because the timestep feature of unity could no longer be used. Setting it higher would cause the whole project to lag and have unwanted behaviours such as teleporting and clipping. If this was moved to the GPU using threading and compute shaders, larger time steps can be used to simulate a faster training environment. Though training times were longer, they would still come to the same conclusion. This may not be an issue on the training end but in a large-scale game, if many of these agents are running with different actions, it could be a huge load on the system and slow down the game. Moving to Unity's job system can also help processes running on the CPU by splitting the workload into small self-contained units of work.

Performance Against Intended Solutions

The solution that the system comes down to can be very different from the intended output. This is in comparison to the backpropagation and gradient descent systems where they are trained on data sets to give them a direction to learn in rather than learning randomly. Gradient descent systems allow for neural networks to learn in a more supervised environment which will produce better working networks. Gradient descent systems still fall to this issue but is significantly easier to direct

the way the network learns. This is a large performance difference for the implementations as you could spend more time randomly training your network rather than providing expected data sets.

Comparison to Similar Assets

After browsing the Unity asset store, many other neural networks are similar in what they provide compared to the one I have created. Major factors that make other packages look appealing is the visualisation of the neural networks and their nodes and connections. There is a small prototype for this being created in the current package and is not yet complete but having that will greatly affect the appeal of my system. Many of them also have a customisable immediate GUI that can be used to change values of their network. This is much more modular than the one I have created but seem more specific to the situation they are creating a network for.

Changes to the System

Decoupling System and Game

To get this system modular and working for many other situations, separating the functionality of the network and the required functionality for the game was needed. This problem was difficult to tackle because the networks system had to be decoupled from the racing game that was created. This caused major problems and the feeding of inputs and taking outputs system had to be moved to an abstract class. Changing this caused a bunch of the functionality to stop working because the class had no functionality in it, and you needed to grab the script inheriting from the base class so that functionality could be used. The solution to this was stating in the comments and the readme that the template class must be replaced with the created class for it to work as intended. This ensured that the system became modular but puts more work on the user using the package.

Complexity of System

Initially, the network was supposed to include many hidden layers to increase the complexity of the neural network. However, this was cut down to a single hidden layer to reduce the difficulty of making it modular. After looking back at the system, adding this feature is not hard. Altering the core scripts and how the network is built will accommodate for more layers. This also gives users another variable to mess around with. They can influence the amount of hidden layers there are and how many nodes are in each. The number of nodes will preferably be unique for each layer instead of having all hidden layers have the same number of nodes.