Research into ANNs and GAs

Introduction

This a report to supplement my Artificial Intelligence Assignment where I outline some of the research I did into ANNs GAs and ultimately the GA i developed to solve the problem of pathfinding through a randomly generated map.

ANN

Artificial Neural networks are computers mimicking, biological neural networks. The way they work is very similar both consisting of lots of inputs and an output. The inputs are weighted and thus the inputs can be tweaked so that a mixture of specific input results in a desirable output. There are two main ways of doing this. Back propagation and reinforcement. Back propagation is where the network first feed forward, being run, and then back propagated back with an expected pattern. These two then can be compared and find adjustments for the weighting.

The other is to not calculate individual error but an overall rating for how the program did and use large amounts of data to run the program lots of time. Over many iterations changing the weights toward more desirable outputs.

GA

Genetic algorithms involve imitating nature again but in regards to the natural selection. GAs rely on creating multiple solutions that regardless of their feasibility. The solutions are tested and ranked by a concept of “fitness”. Fitness is a simple way of gauging the overall goal for an algorithm and giving the ability to rank them. A process of breeding would breed functions and the more successful functions would be more likely to “reproduce”. Further adding “mutations” to generate more concepts makes either better or worse combinations that will be thrive or weeded out respectively.

ANN & GA Comparison

ANNs are easier to accomplish smaller goals with smaller neural networks. This Can be easily backpropagation and produce a fast and efficient way to solve these problems. The larger the problem the larger the backpropagation task becomes in an exponential way. If there are double the number of neurons attached to the end node that those are are attached to double you have 4 neutrons you would have to account for. This scaled up to 10’s of thousands of nodes for a complicated task would be absurdly time consuming to do by hand.

Using a reinforcement method would be far less confusing but also require lots of data as the size of the possibilities for the answer corresponds to the size of the problem. The Benefits of GA for pathfinding is that this require no data and only takes time. The GA will run select better and better randomly mutated path not relying on data and just on the computing power.

This is the main downside. Randomly mutation and breeding are random and the breeding is expensive for most systems. On top of this the time taken for completing simple tasks can be a lot longer that of ANNs. I selected a GA for the purpose of creating a pathfinding AI as the not having suitable data or the effort to train this program made me less keen of the other options and the fact modern computers are very suitable for GAs.

Designing My GA

My program is designed in C++ as it something i am familiar with and will be very efficient as the control over memory can lead to small overhead.

The first thing I decided to do was to design a class-based way to do all of the tasks that would be required for this program. This meant a class for loading maps, a class of storing them, a class for A\* and a class for the Genetic Algorithm.

-File reader

The files that i would be using contained number 0-3 and whitespace characters. The first two numbers where the size of the file and the rest data. The first task was to separate the size of the data and the data itself. A nested for loop and if statements were used to see the potential two numbers for each the x and y component of the size. The data then could be formatted into a vector of int vectors. This the can be output to a maps class.

- Maps

The maps class contains the location of the maps in the form of a vector of “maps”.  As well as this it contains instructions on how to obtain the data legally. The maps being a vector of vector of int vectors of which the outer two are pointers looks very confusing. This could have been stored differently the maps being stored all in one place means it is easy to keep track of and them being pointers means that the maps can be accessed anywhere but still have as little flow of data as you are only passing the pointer rather than the whole array.

-GA

The Genetic algorithm as a class has the gene data, and functions that deal with the aspects of the breeding and mutation of the data. The first functions to do would be initialization and declaration. So, in the constructor and first called function, It creates an initial 4 random genes and calculates the fitness. The First thing to do in the main loop of the class is to Breed. Breeding creates temp genes, has a chance to crossover and set the originals to the new breed genes. After the genes are breed there random thing that could happen to them. The first is to extend the size of the gene adding a new chromosome to the end of the gene. As Well as this to make sure the gene pool isn’t stagnant, the genes have a chance to mutate by randomly changing their chromosomes.

Finally, the fitness is calculated for all genes. Fitness is calculated by going through the entire chromosome and testing the “walls” of the map for collisions and calculating and coming to the end point of the map. Another way that I could have done the fitness calculating would be to do a general end position and then possibly count collisions. I theorise this could be quicker as the time taken to calculate the overall length would be far less than doing every single chrome wall collision. With this program being limited by speed heavily increasing speed is key.

A Boolean exit variable is checked it a chromosome has a perfect fitness of 1 to end the algorithm.

A\*

The Other program is an A\* pathfinder. This is a deterministic method of pathfinding and it check surrounding positions of the starting position giving value to “better” positions. The program then iterates on the value assignment for a path through the map traveling back through its path at obstacles.

Comparison between A\* and GA

The A\* program given was a tested piece of code that was very durable and the times it hardly failed. If given a route that was impossible it would return a message signifying it. Comparatively, would go until it was told to stop or hit a threshold for number of iterations.

Furthermore, GA took upwards of 100000 generations summating over 2 minutes on a testing computer. The A\* code on the other hand took ~1ms.

As for resources the GA took a lot, the number of genes and chromosomes being used and the times these variables are checked and changed are in the 100s per iteration for 100000’s of iterations. Conversely the A\* code only does 9 checks per turn and on average would have up to the size of check in a map for fitness calculations. This difference is in the 10s-100s times larger difference.

Conclusion

In Conclusion, GA was much slower, more intensive and seemed to be more costly to create in terms of classes and framework. GAs and potentially ANNs are not well fitted for this task. AI for this problem will excel at larger problems where the size of the complexity of the map gets to the point where the A\* method take too long.

When it comes to Neural networks, the lack of data and need to test vast amounts of weights makes them hard to train. One thing that I would think would be to use a GA to train weights and add neutrons to a simple neural network and calculate fitness to be time taken and efficacy of the program.

Institute of Technology Blanchardstown, 2018. Pathfinding in Computer Games [online]. Poole: Bournemouth University. Available from: <https://arrow.dit.ie/itbj/vol4/iss2/6/> [Accessed 14 11 2018].