The ITB Journal

Volume 5 | Issue 1 Article 21

2004

Neural Networks for Real-time Pathfinding in Computer Games

Ross Graham
School oflnformatics and Engineering, Institute of Technology at Blanchardstown, Dublin l5.,
Ross.Graham@itb.ie

Hugh McCabe
School oflnformatics and Engineering, Institute of Technology at Blanchardstown, Dublin l5.

Stephen Sheridan
School oflnformatics and Engineering, Institute of Technology at Blanchardstown, Dublin l5.

Neural Networks for Real-time Pathfinding in Computer Games

Ross Graham, Hugh McCabe & Stephen Sheridan
School of Informatics and Engineering, Institute of Technology at Blanchardstown, Dublin *15*
Contact email: **Ross.Graham@itb.ie**

Abstract

One of the greatest challenges in the design of realistic Artificial Intelligence (AI) in computer games is agent movement. Pathfinding strategies are usually employed as the core of any AI movement system. The two main components for basic real-time pathfinding are (i) travelling towards a specified goal and (ii) avoiding dynamic and static obstacles that may litter the path to this goal. The focus of this paper is how machine learning techniques, such as Artificial Neural Networks and Genetic Algorithms, can be used to enhance an AI agent's ability to handle pathfinding in real-time by giving them an awareness of the virtual world around them through sensors. Thus the agents should be able to react in real-time to any dynamic changes that may occur in the game.

Keywords: Neural Network, Genetic Algorithm, Pathfinding.

1. Introduction

Agent movement is one of the greatest challenges in the design of realistic Artificial Intelligence (AI) in computer games. This challenge is compounded in modern games that are becoming more dynamic in nature as a result of middleware engines such as Renderware [Renderware] and Havok [Havok]. These middleware companies allow game developers to spend more time developing interesting dynamic games because they remove the need to build custom physics engines for each game. But these new dynamic games create a strain on existing pathfinding strategies as these strategies rely on a static representation of the virtual world of the game. Therefore, since the games environment can change in real-time, the pathfinding strategy also has to occur in real-time. The two components for basic real-time pathfinding are (i) heading in the direction of a goal and (ii) avoiding any static and dynamic obstacles that may litter the path to that goal in real-time.

This paper will highlight the need for real-time pathfinding and how effectively a neural network can learn this initially at a basic level. It will then discuss the steps taken to determine the possibility of using neural networks for basic real-time pathfinding and the pros and cons of the results.

1.1 The Need for Real-Time Pathfinding

Traditionally in computer games pathfinding is done on a static scaled down representation of the virtual world that the game presents. This works fine if there is little or no change to the virtual world throughout the course of the game. This was the case in most games up until now as the sophistication of the game's real-time physics engine was limited mainly due to the time required to develop it. However games are now being built using middleware for key components of the game, including the physics engine [Havok]. Middleware is software written by an external source that has hooks that allow it to be integrated into a game developer's code.

Therefore game developers can spend much more time creating more immersible games with real-time dynamic scenes. This sounds exciting however it is being impeded by traditional pathfinding AI that operates off a *static* representation of the games virtual environment. This limits the amount of dynamic objects that can be added to games, as the pathfinding strategy will have to be fine-tuned to handle them thus adding more time to the development of the game.

To allow an AI agent to effectively navigate a dynamic world it would have to be given real-time awareness of the environment surrounding it. To achieve this with traditional methods would require running the pathfinding algorithm at every move, this would be computationally expensive, especially for the limited memory available to the games consoles of today. Therefore the AI agent will have to be given some kind of sensors that can obtain information about its surroundings. This is not difficult to implement, however a key problem arises in making the agent decipher useful information from these sensors and react accordingly in real-time without putting too much of a strain on the computers resources. Artificial neural networks are a well known AI technique that provides a potential solution to this problem.

1.2 Neural Networks for Real-Time Pathfinding

An artificial neural network is an information-processing system that has certain performance characteristics in common with biological neural networks [Fausett94]. Each input into a neuron has a weight value associated with it; these weights are the primary means of storage for neural networks. Learning takes place by changing the value of the weights. The key point is that a trained Neural Network (NN) has to ability to generalise on situations that it has never encountered [Champandard 04]. This is a particularly useful feature that should help considerably with dynamic scenes.

There are many different types of neural networks but the one particularly suited to real-time games is the Feed Forward Neural Network (FFNN) due to the speed it can process data [Fausett 94]. Therefore the extent to which a FFNN could be trained to decipher the information presented to it, from sensors attached to an AI agent in a dynamic virtual world, was investigated. These sensors will receive real-time data from the physics engine therefore giving the agent a sense of awareness about its surrounding environment.

The next stage was to come up with some system that will train the weights of the FFNN. If simple rules are to be learned then it is possible to compile a set of inputs and their expected outputs and train the FFNN through backpropagation. Otherwise reinforcement learning [Champandard 04] will be used to evolve the FFNN's weights through a genetic algorithm

(GA) [Buckland 02]. This is achieved by rewarding AI agents for following various rules that the user specifies at different time intervals. Then the AI agents are ranked according to their respective scores with the top ranking agents putting a mixture of their weights in to a lower ranking agent. This is analogous to nature's survival of the fittest model in the real world that has helped humans evolve to where we are today.

1.3 Evolving the Weights of a Neural Network

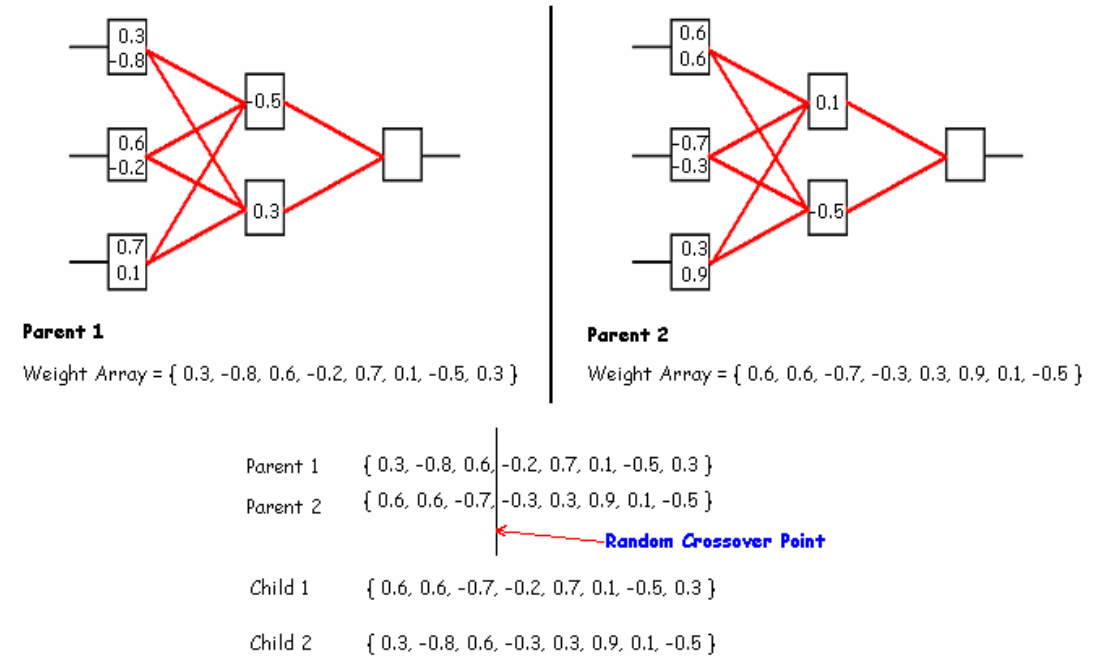


Figure 1.1

The encoding of a neural network which is to be evolved by a genetic algorithm is very straightforward. This is achieved by reading all the weights from its respective layers and storing them in an array. This weight array represents the chromosome of the organism with each individual weight representing a gene. During crossover the arrays for both parents are lined up side by side. Then depending on the crossover method, the genetic algorithm chooses the respective parents weights to be passed on to the offspring as shown in figure 1.1. Training the neural network for basic real-time pathfinding first required it to learn to (i) head in direction of goal and then to (ii) navigate around any obstacles that might litter the path. Therefore the first step will be to see if the NN can learn these two tasks separately and then finally learn both of them combined.

2 Test bed

The test bed for the experiment was a simple 2D environment (grid of square cells) in which the agents can only move either *up*, *down*, *left* or *right* one cell from their present locations. There is a boundary around the perimeter of the grid that has one of the following two properties *solid boundary* and a *wrap- around boundary*. With the *wrap-around boundary* if

the AI agent hits it they will be transported to the opposite side of the grid while with the *solid boundary* the agent is stopped from moving. Objects can also be added to the grid, which will permanently occupy their respective position in the grid thus making it off limits to the agent. The test bed also allows real-time modification to the genetic algorithms parameters so the user can observe the evolution in real-time and change these values due to observations. Thus the test bed offers the NN a simple virtual environment to learn the two different components of basic real-time pathfinding through reinforcement learning, which will be conducted through a Genetic Algorithm (GA). This was done in terms of stages of increasing difficultly that present an AI agent with different situations.

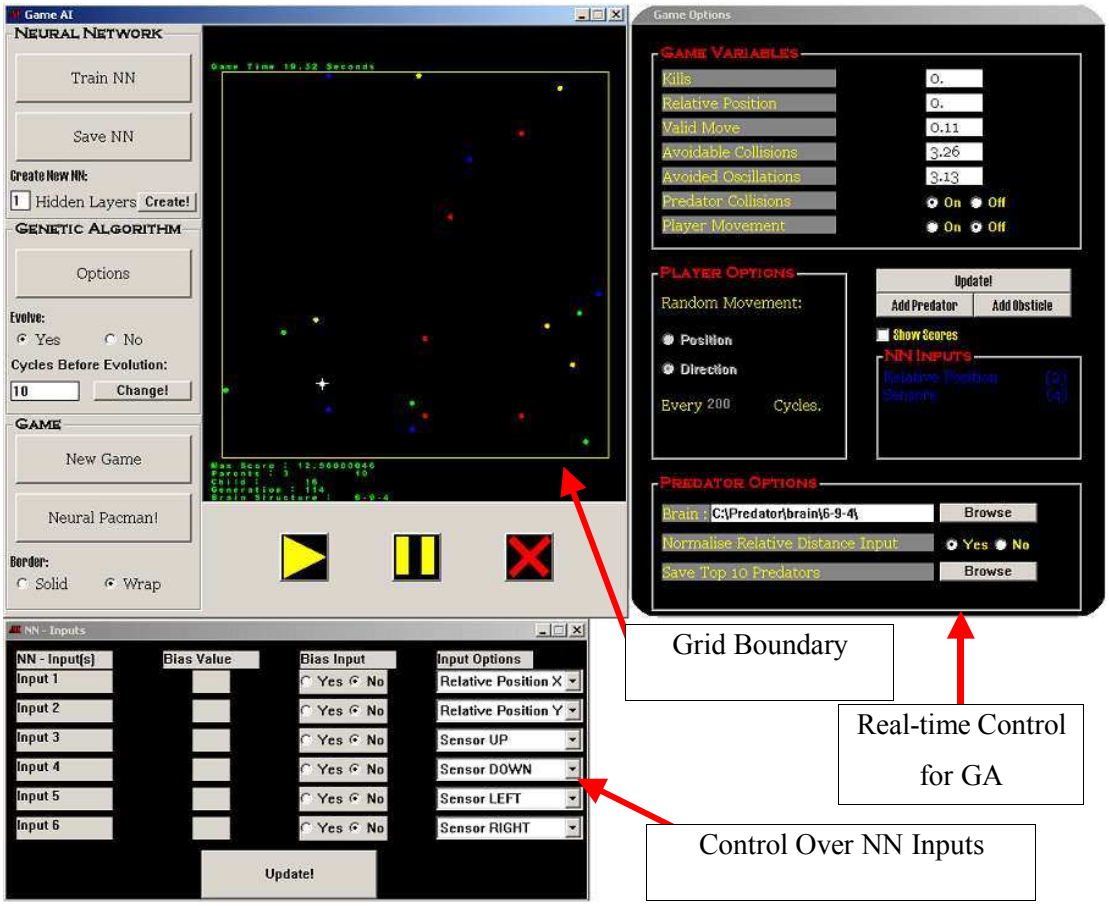
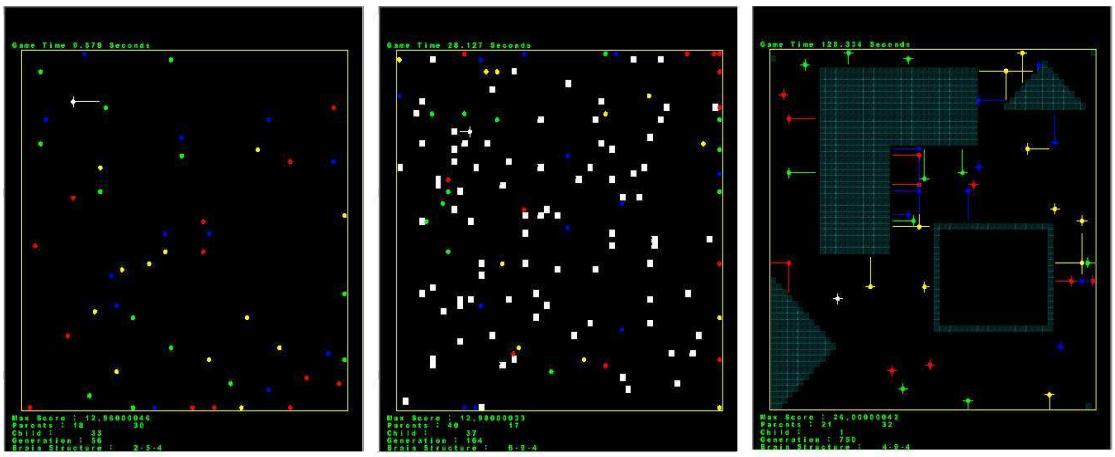


Figure 2.1



(a) Grid With No Obstacles (b) Grid With Small Obstacles (c) Grid With Large Obstacles

Figure 2.2

2.1 Stage One

This stage comprised of a Predator\Prey pursuit scenario with no obstacles as shown in Figure 2.2(a). The relative position of the prey was used as the input to the NN.

Neural Network Information	
Inputs (2)	Outputs (4)
Relative Position of Prey on X-axis	***UP***
	DOWN
Relative Position of Prey on Y-axis	***LEFT***
	RIGHT

The first thing that the NN was tested on was its ability to go towards a goal. The idea here is to have predator that relentlessly pursues its prey around an obstacle free space. Therefore the predator will decide which way to move via a NN that takes the relative position of the prey as its input. The NN had four outputs for each of the possible moves it can make. The output with the strongest signal was selected for the next move.

Num Inputs	Hidden Layer Neurons	Time(Sec)	Generations	Population Size
2	**2**	**230.1**	**1380.6**	**20**
2	**3**	**141.6**	**849.6**	**20**
2	**4**	**83.5**	**501**	**20**
2	**5**	**99.5**	**597**	**20**
2	**6**	**101.9**	**611.4**	**20**

Table 2.1

Table 2.1 shows the length of time and the number of generations it took for the predators to evolve to a satisfactory solution to the behaviour required with different numbers of neurons in the hidden layer. The results were compiled by getting the average time for a satisfactory result after running through the simulation ten times for each of the hidden layer configurations. As

highlighted in table 2.1 neural networks with four neurons in their hidden layer on average evolved quickest to the solution.

Since the objective of the NN was very clear and simple it was possible to train it using back propagation and GA [Fausett 94]. This task was learned very quickly through back propagation and GA but GA had the benefit of creating less predictable solutions. This simple Predator/Prey demonstrated that the NN had no trouble learning to head in the direction of a goal, which is the first requirement of real-time pathfinding.

2.2 Stage Two

This stage comprised of a Predator\Prey pursuit scenario with small obstacles as shown in Figure 2.2(b). The inputs to the NN were the Relative position and the contents of the four surrounding cells.

<table>
<tr><td colspan="2">Neural Network Information</td></tr>
<tr><td>Inputs (6)</td><td>Outputs (4)</td></tr>
<tr><td>Relative Position of Prey on X-axis</td><td>UP</td></tr>
<tr><td>Relative Position of Prey on Y-axis</td><td>DOWN</td></tr>
<tr><td>Cell Above</td><td rowspan="2">LEFT</td></tr>
<tr><td>Cell Below</td></tr>
<tr><td>Cell Left</td><td rowspan="2">RIGHT</td></tr>
<tr><td>Cell Right</td></tr>
</table>

This next stage aims to test if the predator can avoid obstacles that litter the path between it and the prey. To achieve this the predators NN was given more inputs so as to inform it about its immediate surrounding environment. To keep it simple the predator was given four sensors that indicated if the next cell in each of the possible directions from its present location was obstructed.

The task was learned quickly through back propagation and GA for obstacles up to two cells in size, however with obstacles larger than two cells the Predator got stuck. This indicated that the predator did not have enough time/information to avoid larger obstacles as it could only sense one cell ahead. Therefore to possibly overcome this problem the predator would need longer-range sensors i.e. sensors that scan greater than one cell in each direction.

2.3 Stage Three

This stage comprised of a Predator/Prey pursuit scenario with large obstacles as shown in Figure 2.2(c). The inputs to the NN were the relative position of the prey and with four directional sensors that can look ahead more than one cell.

Neural Network Information	
Inputs (6)	Outputs (4)
Relative Position of Prey on X-axis	***UP***
Relative Position of Prey on Y-axis	***DOWN***
Sensor UP	***LEFT***
Sensor DOWN	
Sensor LEFT	***RIGHT***
Sensor RIGHT	

The Predators could not learn the two tasks combined with any real impressive results therefore the search space needs to be reduced. One way of achieving this would be to use a hybrid neural network which is more that one neural network being used to solve the problem. i.e. a NN could work out the direction to travel towards goal and then input this into another NN that checks for obstacles.

2.4 Stage Four

This stage comprised of a Predator with four sensors that can look more than one cell ahead in each direction for obstacle avoidance as shown in Figure 2.2(c).

Neural Network Information	
Inputs (4)	Outputs (4)
Sensor UP	***UP***
Sensor DOWN	***DOWN***
Sensor LEFT	***LEFT***
Sensor RIGHT	***RIGHT***

Since there are two parts to real-time pathfinding it was decided to investigate whether the NN could learn to steer around large and small obstacles. Thus giving similar results to Renoylds steering algorithms [Reynolds99]. This time the predators were only given long-range sensors as inputs to their NN. The Predators learned to steer away from obstacles large and small. Therefore if a NN can learn the two requirements for basic real-time pathfinding separately it is a reasonable assumption that with a bit more refinement into the training procedure that it can be trained to do both of them.

3 Future Work

Future work will involve investigating the use of hybrid neural networks [Masters93] that will help break up the problem into its two components thus reducing the search space for the full problem. Our intention is then to expand the research into 3D games and benchmark the results against traditional pathfinding strategies such as A* and Dijkstra [Graham04] with regard to speed, realistic movement and ability to handle dynamic environments. The move to 3D should

not be much more of a challenge for the NN to handle as gravity constrains in most games will confine the agent to a 2D plane most of the time. Another extension to this research is to look at a path-planning component that can guide the AI agent by supplying it different goals to seek.

3.1 Conclusion

The two main problems associated with traditional pathfinding are they (i) rely on a static representation of the virtual world and (ii) rigid unrealistic movement may be observed unless fine-tuned in someway. This therefore hinders using the full power on offer from dynamic middleware components. Since a neural network (NN) can learn the two main components required for a basic level of real-time pathfinding separately with some more research into refining the training, it will be possible for a NN to learn the two together. The NN should also be able to generalise on dynamic situations that it did not encounter during training. This would offer the possibility of creating a pathfinding Application Programming Interface (API) based on a neural network that will take inputs from a games physics engine in real-time. Thus create the foundation for real-time pathfinding middleware that would allow the next generation of computer games to immerse players into much more unpredictable and challenging game environments.

References

[Buckland02] Buckland, Mat,. "AI Techniques for Game Programming", Premier Press, 2002
[Champandard04] Chapandard, Alex J., "AI Game Development", New Riders Publishing, 2004
[Fausett94] Fausett, Laurene, "Fundamentals of Neural Networks Architectures, Algorithms, and Applications", Prentice-Hall, Inc, 1994.
[Graham04] Graham, Ross., "Pathfinding in Computer Games", In proceedings of ITB Journal Issue Number 8, 2004, Pages 56-80
[Havok] www.havok.com
[Masters93] Masters, Timothy., "Practical Neural Network Recipes in C++", Boston: Academic Press, 1993
[Renderware] www.renderware.com
[Reynolds99] Reynolds, C. W., " Steering Behaviors For Autonomous Characters", In proceedings of Game Developers Conference 1999 held in San Jose, California. Miller Freeman Game Group, San Francisco, California. Pages 763-782
[RusselNorvig95] Russel, Stuart., Norvig, Peter., "Artificial Intelligence A Modern Approach", Prentice-Hall, Inc, 1995