The use of Neuroevolution for the optimisation of car controllers in a simple racing game against pre-evolved controllers.

Lee Kempton (10520595) - Plymouth University

Abstract:

This paper describes the use of Neuroevolution, along with neural networks, multilayer perceptrons and general evolutionary computation algorithms in the training of car controllers for a simple racing game against a pre-evolved controller, within the context of the technology presented. The first part of the paper will talk about the advantages and disadvantages of neuroevolution, both from the context of classical artificial intelligence research and from the perspective of games that either rely on, or make use of such technologies. The commercial viability of such methods of computational evolution will also be looked at briefly, taking examples from successful games, whether their focus by on the AI technologies at hand or simply on the possible applications for fun games media.

With this paper, the aim is to present a short write up of the technology at play and then talk about it from a game's side of the industry, offering related texts and work as a basis for the claims that are made. Also, the paper will be used to present some experimental data taken from tests done on a car controller. This data will be evaluated in the later parts of the paper and links will be made based on the graphical data demonstrated. Finally, relationships between two pieces of data will be highlighted and discussed, along with collections of both average and best in class data, this paper should provide a short yet concise idea of the applications of neuroevolution for a car controller in a simple racing game.

Introduction:

Overview of technology:

Neuroevolution is a current and well understood part of the field of computational intelligence; with common usage in a wide variety of AI applications including games, robotics and procedural generation[1]. The biological motivated technique has proven very popular and is becoming increasingly more researched as new use cases are becoming viable, both technically, and from a consumer want perspective. I will discuss this more in the next section, which aims to look at current NE applications that exist, in both games and otherwise. This section will cover some background information of what NN is and it’s applications and issues within both the game, and non-game related fields.

Neuroevolution is primarily inspired by Darwinian evolution, meaning the whole technique is built around the kind of biological and evolutionary theories that Darwin was renowned for [1]. Neuroevolution works by using Neural Networks that use ‘weightings’ between different nodes to turn a fixed number of input nodes into some tangible output. Whether that be a command for acceleration in either direction, in the case of the subject car racer, or something completely different, like what would be expected from a NN based procedural terrain generation program [2]. One of the major advantage of NN is the huge capacity for

learning, along with the ability to deal with ever rising demands for control systems [3]. The main key term here being ‘learning’, which is obviously an interesting concept when it comes to artificial intelligence and can allow for use of the technique on a wider range of problems with increasingly large search spaces required in order for an effective solution to be acquired [1]. NNs also provide the benefit of being extremely effective at certain problems [1] and due to the fact they are often modelled after the human brain; they can be very effective in non-game based fields, such as medicine and medical diagnostics [4].

Despite these generally positive and advantageous reasons to use NE, it does have a downside. Namely, the fact that it can be arduous to debug and predict what is being learned by the network [1]. Another potential issue with using NE with NNs is the fact that they have to be trained before use, meaning if the training data is not diverse enough then the final solution can be lackluster at best [5]. This is an issue in non-game applications such as for misuse detection in IT security [5] and also once again has a ‘black box’ effect mentioned before, where it is extremely difficult for humans to understand what exactly is going on [5][1].

The experiments that this paper will talk about later on use a Multilayer perceptron, where input is converted into outputs, or actions, that the car controller will use. These input pass through a series of weights and hidden nodes before being turned into output commands for the controller to use. This is known as a ‘feed-forward network’ and is one of the more popular uses of networks for both researchers and general users of an NN [6]. The main thing that separates a single layer network from a Multilayer Perceptron is the inclusion of ‘hidden nodes’. These hidden nodes act as an intermediate layer between the input and output nodes [7], with weights that provide an idea to what could be the correlation between each neuron activity and the neurons that they connect with [7]. Outputs typically represent what a Human player would normally input on a controller or keyboard/mouse and is always in response to external stimulus from the game, for example, dodging out of the way of an attack, or driving towards an end goal. This can change depending on the game that is being run, but follows the same general idea. Inputs, like stated, come from in game events that can be acted upon; each of these events has a weight which offer a form of bias towards certain actions. It may be optimal to collect more health before trying to make an attack, in an FPS, for example.

One reason that NE differs from other methods is the fact that it can produce results with only knowledge of how well the network is doing [1], this means that individual strategies do not have to be programmed in by a Human. The NN will adapt over many generations, slowly increasing the average fitness of a population. The fitness is the direct measure of how well the network is doing, in the case of this experiment it is car racing, but can also be many other things which will be explored in the related texts section. Fitness can be defined by a variety of parameters, from score to speed to angular turning velocity; in the case of non-game implementations it can be almost anything that defines a result as either ‘good’ or ‘bad’.

Related Application and Text:

While the previous section covered the idea of Neuroevolution and its application in some wider fields, this section aims to compare and contrast the use of the technique in a closer, more related game scope. NE within games has been influential enough to spawn the creation of whole new genres; from the squad tactics based evolution of NERO (2005) [8], to the literal breeding and raising of virtual companions in Creatures (1996) and other interesting games that dip their proverbial toes into the pool of artificial neural networks and evolution. Later on in this section each of these games and a few more of varying genres will be the focus of scrutiny, both as a way to measure success of said games as 'games', and to deduct whether the use of NN in general is desirable for the industry; both financially and technologically.

Currently, Neuroevolution in video games are mostly being used to learn either how to play a game, or to control an NPC (non-player character) in the game [1]. This might be so the game can put up a convincing and challenging 'fight' for the player, or to replicate what activities an NPC might want to take part in, without the need for explicit Human commands to do so. The two main roles of NE for this is the analysis of states to select the best algorithm for a script to call; or the direct and immediate selection of actions that can take place during a specific state [1]. However, there are some alternative ways that NE is being researched into being used. One of these is procedural content generation [1], which was mentioned earlier on, and the other is a kind of 'smart game director', which can evaluate the behaviour and preferences of the player to deliver an ever changing game experience [1]; Left 4 Dead (Valve 2008) [9], makes use of a similar 'director' system, along with procedural generation algorithms and is probably the most well known and commercially successful game that uses such AI systems to dynamically change gameplay.

Valve's use of 'the director' in their games are relatively recent in context of the field; it would almost be a crime to talk about game AI without mentioning the original benchmark to a computer's ability to be *artificially intelligent,* Chess. An ancient game still beloved by intellectuals everywhere, it was also the main focus of AI researchers back in the 60s, 70s and beyond [10]. The idea being to get a computer to perform as well as, or better than a proficient/master level Human by effectively using learning algorithms and other similar techniques. Specifically, estimating the 'value' of board positions and making seemingly educated decisions on what moves an agent can take [1]. This was a reality in 1983, when the first computer program earned the US title of master [10], breaking the perceived notions of what a machine could do; opening the floodgates to a rush of computer scientists and psychologists trying to beat increasingly difficult problems, with larger and more complex ranges of search spaces. These worked in a very similar way to the car controllers experimented with here, with neural networks building up nodes; selecting individuals; evaluating fitness; and varying the population based on this in a recognisable way [10].

Go, is yet another ancient game that is got the attention of researchers around the world. However this is a much more difficult game to master for a computer, with even the most advanced computer algorithms struggling against average players [11]. This is the catalyst behind the research and development of new AI techniques such as SANE (Symbiotic, Adaptive Neuro-Evolution), this is outside of the scope of this paper though. Nearly all of these classical games' difficulty of computation comes down to the amount of possible moves that are available. For example, the game tic-tac-toe is an early and more basic use of artificial computation techniques and serves as a good start point due to its varied yet relatively small number of possible moves in each iteration [12]. These too use neural networks and can actually boast impressive results, with evolved computer controlled players being able to win or draw the majority of times [12]. Checkers is another test bed for Neuroevolution, as like the games mentioned before it, offers a wide number of possible moves and states that a network of evolving topologies thrives [12].

Classical board games which have existed for a long time are an obvious target for the research of advanced AI methods, however, these are typically turned based games and while the techniques proposed are impressive without doubt; it can be equally as notable to look at a more modern application of NE. Left 4 Dead was previously mentioned, however this makes a rather 'watered down' use of the systems at play, not to say the implementation is not outstanding; but instead it says more about how 'fun' NE is as a way of game design currently. A good case study would be Neuro-Evolving Robotic Operatives (NERO). NERO is game where a squad of 'dumb' units are trained in various test situations in order to then be put up against a 'real' threat, which requires them to use all that has been learned. Behaviour in NERO is actually learned very quickly [13], meaning that unlike Chess and the other similar classical games, this evolution can be seen to happen in real time, as the game is played [13]. This makes for an extremely exciting game, as the player can actively witness their soldiers gain knowledge and become a formidable force based on the user's inputs into basic behavioural parameters; like tendency to run, fight or the units prefered physical configuration [13]. Another potential benefit of NERO as a platform for AI techniques is the abstracted nature of the way a user evolves a force, meaning it could be a good foot in the door for teaching about NE to people who may have no prior interest [13], which in turn increases the popularity and therefore demand for such games by the general public.

From the games listed so far it might seem that Neuroevolution for games is close to completion. This is not however true, as there is still lots of research to be done into games which have huge amounts of decisions and states. An example of a game like this is the game series Civilization (1991 - 2016), these offer a huge number of possible game states, with a ridiculous number of turn options to decide from [1]. In fact, grand strategy games in general are a large 'bump' in the road to truly generalised Neural Networks that can play anything. Generally, a work around for this issue is an AI that cheats; this can be a good way to build up a facade of believability in the player[14], but is not actually using NE methods, rather

faking them in a clever way. In addition, there are other cases in which it is difficult to use NN in a game context, for example if no source code for a game exists, it will suffer from the same 'black box' effect that was referred to earlier [1]. Finally, NE can produce some unexpected results due to the nature of selection, mutation and crossover algorithms that are used, this can make it tricky to QA test a game, as there is a very large number of outcomes and some realistic AI response can be seen as buggy or 'unfun' by the average player [14].

Neuroevolution in games are best when the games being created are designed with NE in mind. This helps to avoid all of the pitfalls of NN while providing an experience that plays directly with the technology in question. The next section of this paper aims to evaluate and then discuss a series of experiments taken on the car racing game; designed for specific testing of optimisation of controllers using NNs, it should provide some insight into the intricacies of MLPs, NE and NNs.

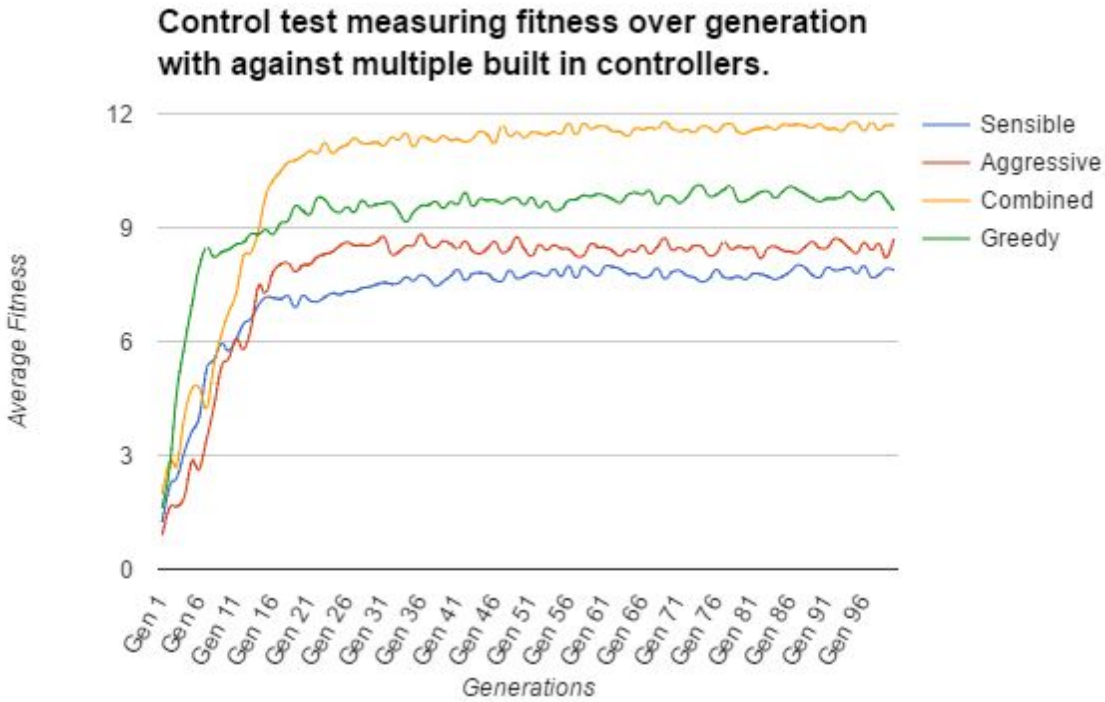
Experiments and Results:

In the car racer a number of variables are available to change and test with. The experiments presented in this section each aim to evaluate the result of changing a single or small number of variables against a control version, which are default in nature. The overall experiments are tested against one of the pre-evolved car, specifically the Heuristic Combined Controller. This was chosen to be the test controller after the control tests returned the greatest fitness value for that controller.

The main thing being tested in these experiments was explorative data, specifically mutation. The crossover variables were not changed throughout but instead variables that are affected directly or indirectly from the mutation magnitude.

Control Test:

The first control test consisted of 4 different individual evolutions, one for each of the non-human controlled controllers. The variables that were entered into the program were kept to mostly default values, apart from the inputs and population size, which were increased to 3 and 100, respectively. The inputs selected were first person inputs regarding the evolved controller's speed, distance to the next waypoint, and the angle to the next waypoint. The population size was increased to 100 by default as it offers a wider range of data and therefore more reliable data.

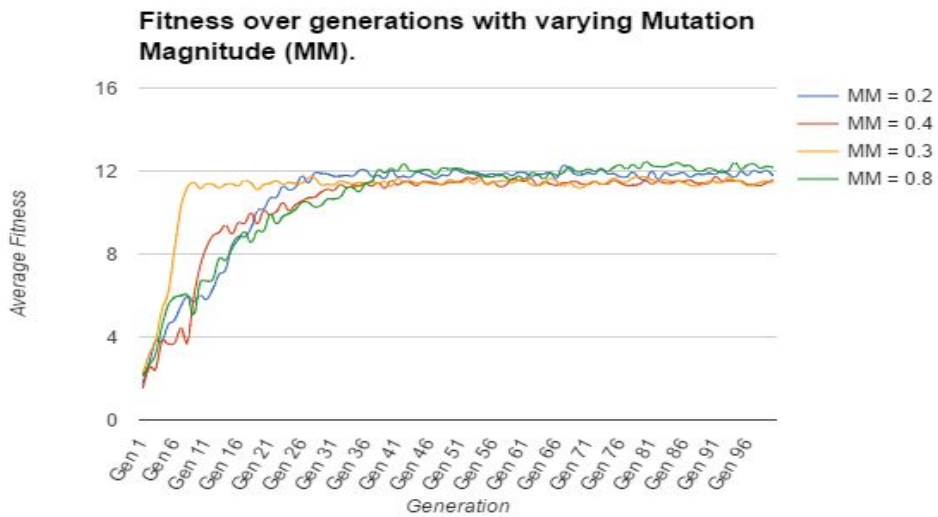


From the data gathered by the control group it was clear that the combined controller was the best controller for the evolved car to compete against. It offers a significant increase in average fitness when compared to the other three controllers. In addition, the combined controller was the only controller to result in an overall increase to average fitness once the initial plateau of data points after approximately generation 20. This makes sense as the combined controller is designed to have small bits of functionality from all the other controllers, and therefore would be a good test bed for experiments that change other variables.

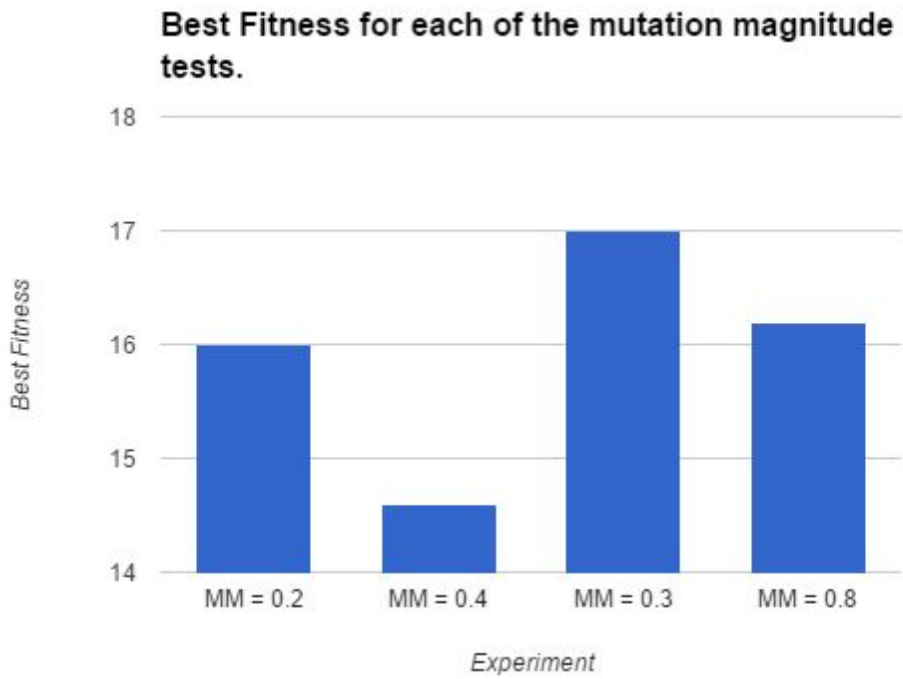
The best fitness of the population was also measured for each experiment. This shows how well the very best individual did and is also the individual used during the play mode. Here it is obvious that the combined controller was superior, with a way higher fitness; shortly followed by the greedy controller.

Mutation Magnitude Test:

The second experiment took all of the variables of the control test, along with the test bed of the combined controller, but changed the mutation magnitude throughout the test; making sure to test both extreme and intermediate values. The following values were tested against the control value of 0.1: 0.2 (double the control); 0.3 (an intermediate value); 0.4 (4 times the control); 0.8 (8 times the control).



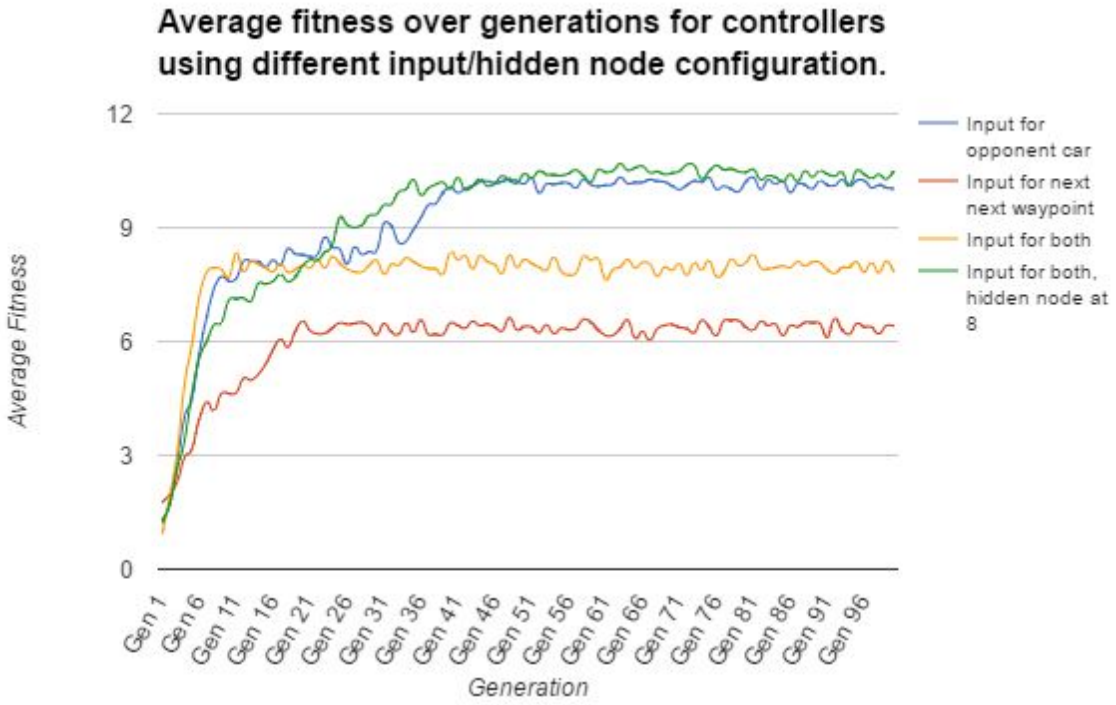
The average fitness data was rather inconclusive, although it did show that the intermediate value of 0.3 was the quickest to plateau, gaining it's max average fitness quickly and then leveling out along with all other values. However, the best fitness is much more useful to use when deciding on an optimal value.



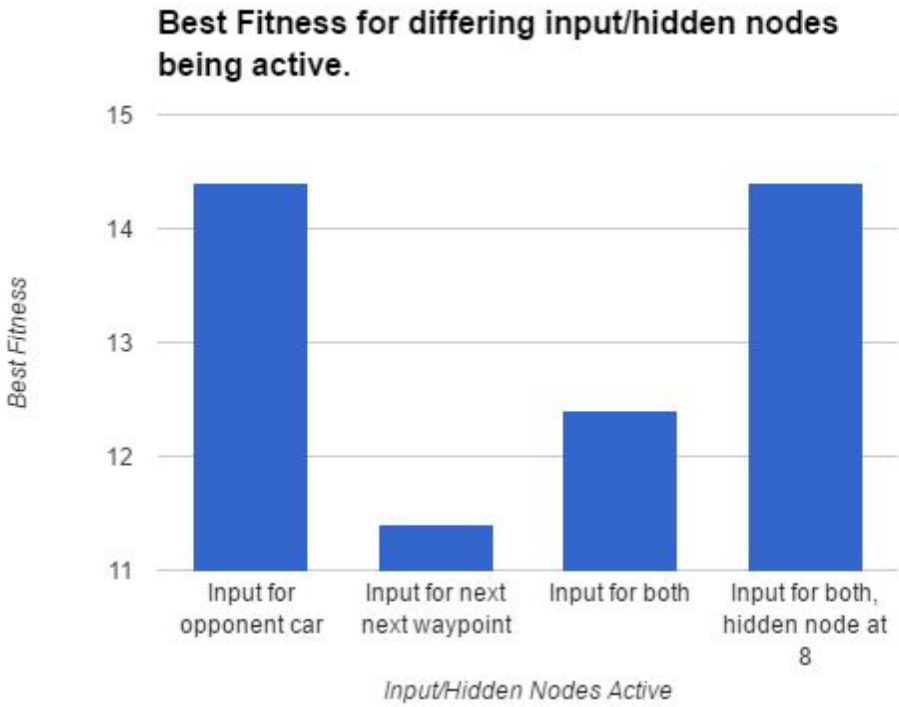
This chat shows that the most optimal mutation magnitude is 0.3, followed by 0.8 and 0.2. A mutation magnitude of 0.3 was optimal in both the average fitness data and the best fitness data, so this is likely to be the best setup for the variables used.

Input/Hidden Nodes Test:

The next experiment that was taken was on the input and partially on the hidden nodes active on the evolved controller. For the input nodes the experiment tries to see if adding more information to the car allows it perform better. These two pieces of extra information was the distance and angle to the opponent car and next next node, for a total of 4 extra inputs. Finally, two more experiments were taken, for both piece of information together and for both pieces of information along with a 1.5* increase to the number of hidden nodes active on the controller.



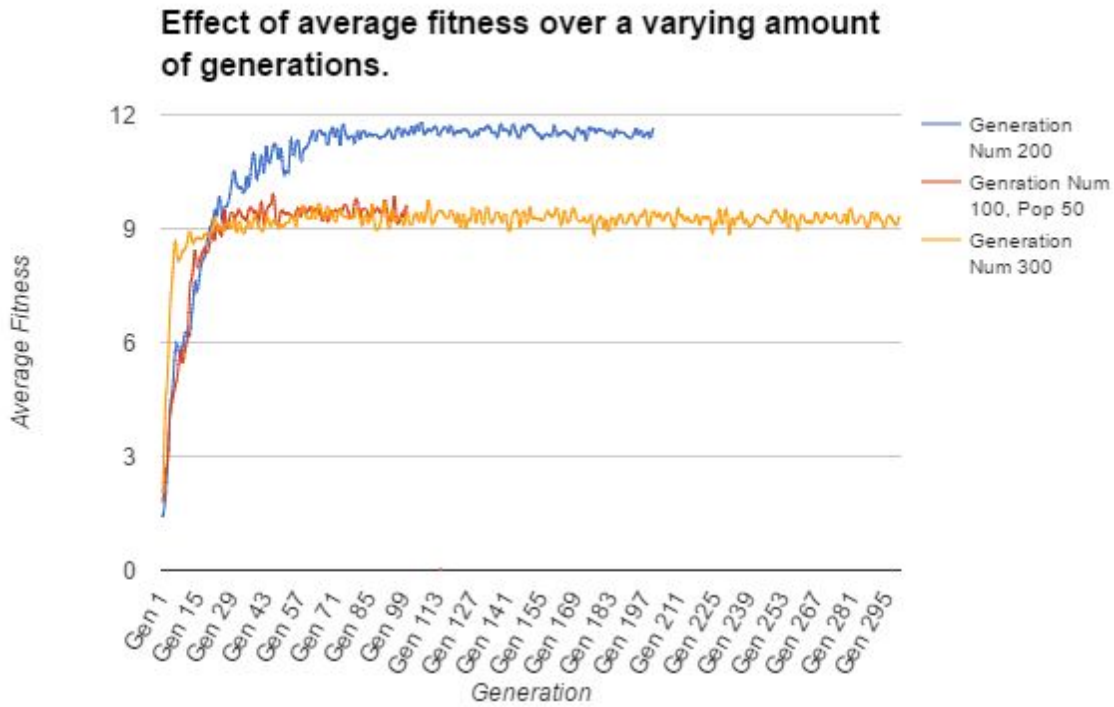
The best average fitness observed was from when both new inputs were implemented along with a higher number of hidden nodes. However, despite this run doing the best out of the set, it was still worse than the control test, which showed both a better average fitness and best fitness. Observed behaviour for the evolved controller during play mode showed the car struggling to decide between which points to go for, slowing down way too soon, and attempting to 'drift' through points, which was ineffectual in its execution.



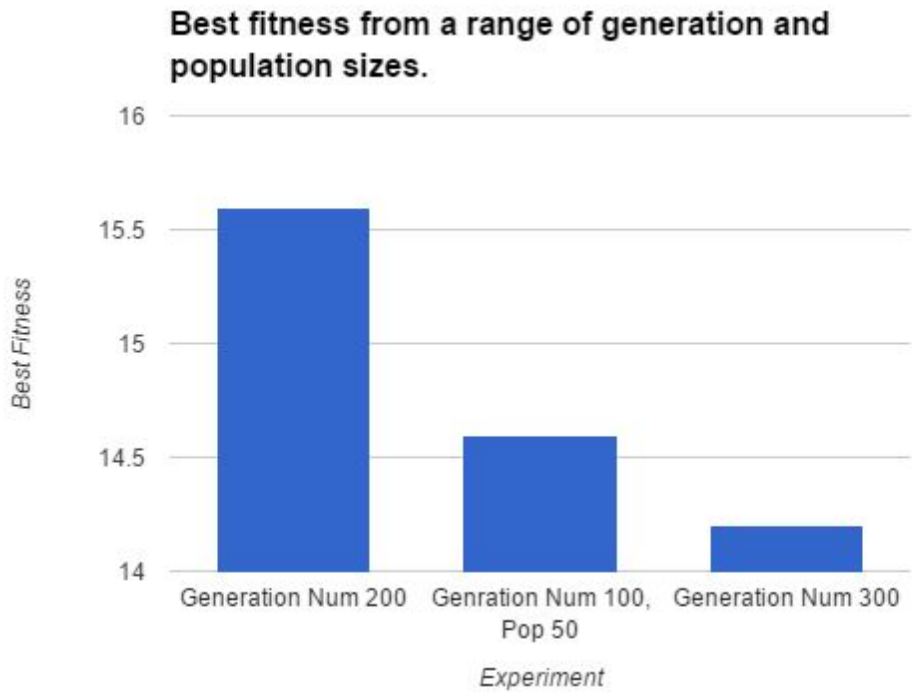
The best fitness data shows that the changing of the hidden nodes made very little difference to the best result obtained. It also shows that the other two inputs were way less effective, scoring only around 11 to 12.5 for their fitness. Overall the data received from these tests was lower than that of the control test, this will affect the optimal controller experiment performed later on in this section.

Generation/Population Number Test:

The next test was based once again on the control group, but this time with the number of generation and population changed throughout. Firstly, beside the result obtained the biggest issue with increasing either of these variables is the increased processing time for a single run through an evolution. Doubling and then tripling the generation size caused the time to take experiments to increase by the same factor.



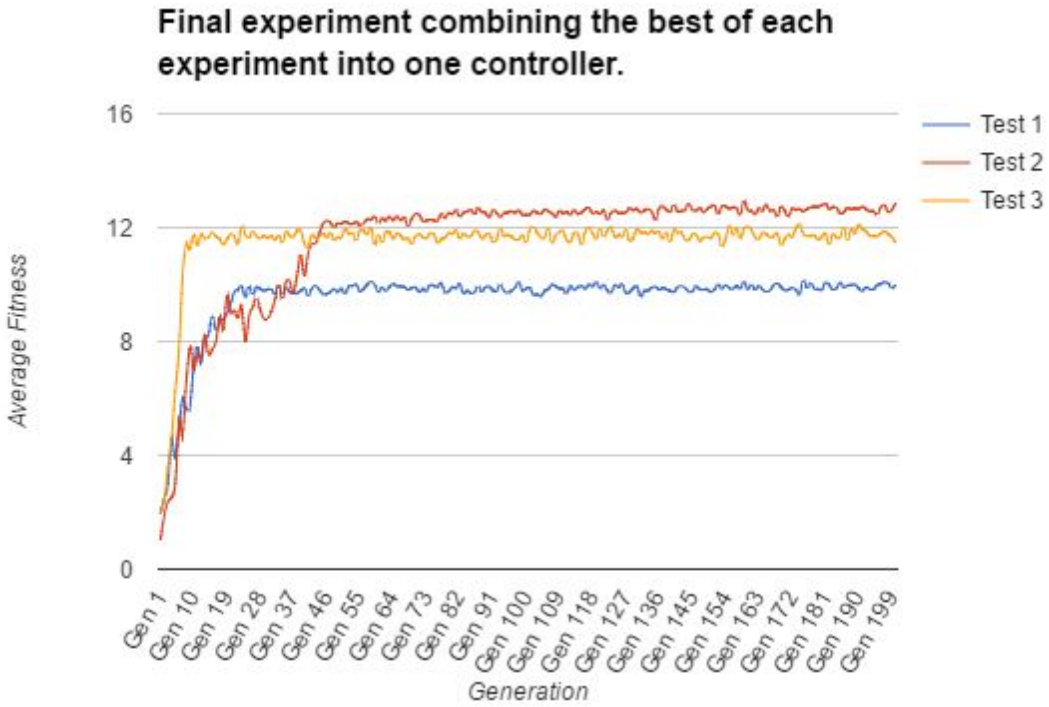
The car was tested with three different variable sets. One with the generation number at 200 (doubled); one where the generation number was the same as the control, but with the population halved; and another where the generation number was set to 300 (tripled). The average fitness overall is clearly higher when the generation is set to 200. This will be discussed in the next section.



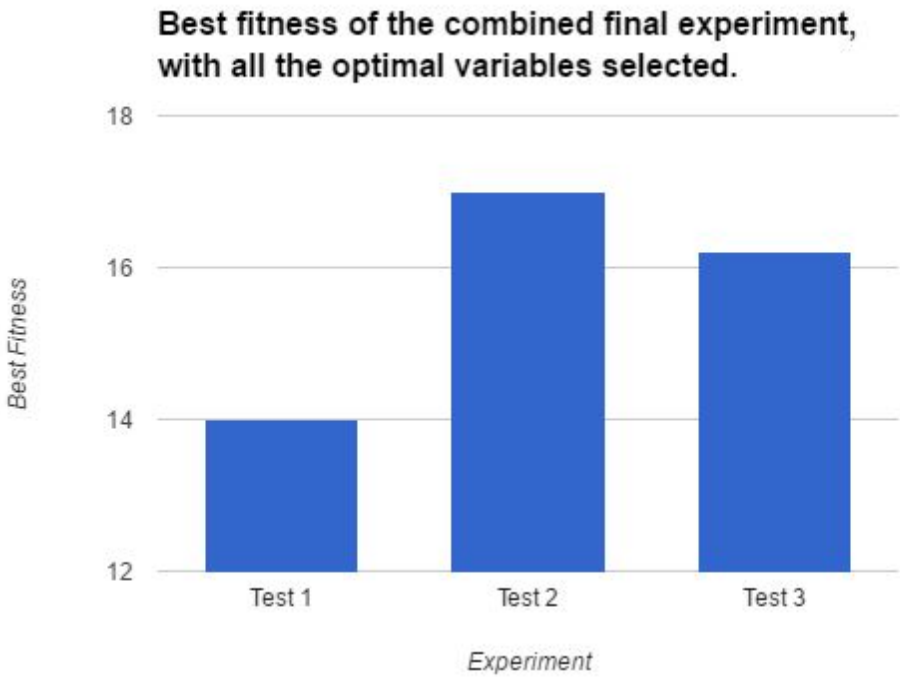
Along with the average fitness, the best fitness is also given to the test with generation set to 200, because of this being the optimal value the generation was set to 200. Despite this value being lower than the control fitness for the same tests, it was hypothesised that along with the increase to the mutation magnitude that was deemed beneficial before, it would combined to make a much stronger fitness than the control.

Optimal Variables Test:

The previous tests were designed to isolate which variables affected the evolved solution in which way. The optimal variable test was the last test to be carried out and aims to take all the best solutions and put them into a single controller. The mutation magnitude was set to 0.3, this was in line with the results gathered from experiment 2. The input and hidden nodes were left at default, no observed improvement could be seen from changing these. The generation size was increased to 200 and the population stayed the same, this followed the tests done above which proved that this was the most optimal configuration. Finally, the combined controller was trained against, as this was the best opponent as gathered from the control test. Three tests were carried out on this experiment, but no changes were made between each of these.



The average fitness was generally higher than that observed from the other 4 experiments, with the second of the three iterations performing the best later on and the third test having the best initial average fitness gain. The increased generation can be seen to have been useful from this data chart; small increases to the average fitness can be seen even after generation 100, where it would have been stopped previously.



Test 2 in the optimal variable experiment produced the best fitness not only in its scope, but also out of every other experiment. The end result was 17 fitness, which is quite high considering the control best fitness of 15.8. Test 3 was still a higher value than the control and test 1 was the only value observed to be lower than the control, for the best fitness in this experiment.

Overall, the tests performed were very useful in understanding which pieces of data were affecting what in the system. In the next section, the data presented here will be evaluated in order to get an idea of the meaning behind it, what conclusions could be made, and the ways the experiment could have been improved.

Discussion of Results:

This section of the papers aims to give a more subjective look at the results gathered above and the general applicability of evolutionary computation, neural networks, and neuro evolution. The final best evolved controller ended up with a fitness of 17, when compared with the scores obtained from the car racing competition, which has a best score of 20 fitness[15], it can be said that it is not a bad score. From observation of the car on the track it performs well against the opponent controller. In addition, the controller still beat over half of the submitted controllers in that competition [15].

One of the predicted outcomes of changing the mutation variable to be a larger amount is the increased likelihood of a controller being able to evolve a superior fitness later on in its generation cycle. This behaviour was rarely observed in the control tests, where the majority, if not all of the increase in best fitness happened within the first 20 generation. The first test was making the mutation magnitude large and this showed that even at 0.2 the controller would continue to create new bests as late on as 60 generations, almost 3 time longer than the control tests showed. Much more aggressive fluctuation can be observed in the controllers with a higher mutation magnitude, this is likely due to the values being changed during mutation gaining way more from the Gaussian noise applied to it. The best fitness data is inconclusive as it doesn't really follow any trend or expected pattern; this isn't unusual for a variable that makes such use of a normal distribution and only a small sample size. Way more experiments would have to be performed on this data set in order to conclusively state a relationship between mutation magnitude and fitness.

The next experiment was the tests regarding the input/hidden nodes. This overall was shown to be inconclusive. It is expected this is the case due to too many inputs that cause the controller to get confused. During the live tests this was observable as often the player can got stuck or slowed down way too soon. The average fitness data for this test actually shows it going down after around 70 generations. This happened during the test with increased hidden nodes, which is a sign of overfitting in the data. This could be due to having too many hidden nodes, or too many input nodes.

After this was the experiment on the generation number. By itself this tests seems inconclusive, but if combined with the knowledge gained earlier about the mutation magnitude's effect on late fitness increases it can be used to generate an optimal controller using these two variables. That is exactly what was done. The heightened mutation magnitude along with the increased number of generation allows more time for more possible nodes to be explored. The data for the final optimal controller experiment shows that the average fitness is still slowly increasing way past generation 100, which was lacking for when either the mutation magnitude was low, so the controller couldn't reach new areas of possible values, or the generation number was lower, meaning the controller wouldn't be given enough time to find these new possible values before the evolution was complete.

Overall, the experiments performed were useful for realising the relationship between generation size and the mutation magnitude, namely that increasing one has little effect by itself, but increasing both together can have good results. Also, while this is true from the data presented, it should be noted that too many generation can be negative, increasing the time it takes to process the evolution; the mutation magnitude too has an upper limit, as going too high with this variable can lead to decent data sets being completely scrapped in favour of new, wildly far out mutations.

Neuroevolution as a process can be said to have been appropriate for the problem at hand. The car racing works well by using NE and NN to evolve cars based on parameters and weights. Even only making use of part of the available parameters, it was possible to change the way a controller evolved drastically and to use this information to optimise the controller to output a high fitness value that could be considered competitive when compared to other controller results [15]. In order to improve the experiments performed a larger amount of tests would be an obvious choice. It was also be beneficial in future tests to use different algorithms for selection, crossover and mutation; it would also make for a more complete set of data if all of the variables were compared against the control group, so that the importance of those could be evaluated. Finally, more tests could have been taken against some of the other controllers, this would help to confirm readings and test if the final controller is generalised enough to challenge all controllers.

References:

[1] - Risi, S. and Togelius, J., 2015. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*.

[2] - Shaker, N., Togelius, J. and Nelson, M., 2014. Procedural Content Generation In Games.

[3] - Antsaklis, P.J., 1990. Neural networks for control systems. *IEEE Transactions on Neural Networks*, *1*(2), pp.242-244.

[4] - Tu, J.V., 1996. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of clinical epidemiology*, *49*(11), pp.1225-1231.

[5] - Cannady, J., 1998, October. Artificial neural networks for misuse detection. In *National information systems security conference* (pp. 368-81).

[6] - Jain, A.K., Mao, J. and Mohiuddin, K.M., 1996. Artificial neural networks: A tutorial. *Computer*, *29*(3), pp.31-44.

[7] - Pal, S.K. and Mitra, S., 1992. Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on neural networks*, *3*(5), pp.683-697.

[8] - Stanley, K.O., Bryant, B.D. and Miikkulainen, R., 2005. Evolving neural network agents in the NERO video game. *Proceedings of the IEEE*, pp.182-189.

[9] - Booth, Micheal. "The AI Systems Of Left 4 Dead". *valvesoftware.com*. N.p., 2008. Web. 13 May 2008.

[10] - Fogel, D.B., Hays, T.J., Hahn, S.L. and Quon, J., 2004. A self-learning evolutionary chess program. *Proceedings of the IEEE*, *92*(12), pp.1947-1954.

[11] - Richards, N., Moriarty, D.E. and Miikkulainen, R., 1998. Evolving neural networks to play Go. *Applied Intelligence*, *8*(1), pp.85-96.

[12] - Chellapilla, K. and Fogel, D.B., 1999. Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE*, *87*(9), pp.1471-1496.

[13] - Stanley, K.O., Bryant, B.D., Karpov, I. and Miikkulainen, R., 2006, July. Real-time evolution of neural networks in the NERO video game. In *AAAI* (Vol. 6, pp. 1671-1674).

[14] - Nareyek, A., 2004. AI in computer games. *Queue*, *1*(10), p.58.

[15] - Togelius, J., Lucas, S., Thang, H.D., Garibaldi, J.M., Nakashima, T., Tan, C.H., Elhanany, I., Berant, S., Hingston, P., MacCallum, R.M. and Haferlach, T., 2008. The 2007 ieee cec simulated car racing competition. *Genetic Programming and Evolvable Machines*, *9*(4), pp.295-329.