Dexter R Shepherd

Examining How Network Architecture Affects Performance of Reinforcement Learning

AI&AB G6042

10-04-2021

Abstract

This paper describes how network architecture effects the performance of a genetic algorithm (GA) as weight and bias training of a neural network converging on a solution. A Genetic algorithm is an alternative to conventional methods such as brute force and greedy algorithms. The genetic algorithm is applied to a robotic chassis which learns to walk using a Microbial algorithm. It is found that a network of two hidden layers and 5 nodes on each, performed best. A perceptron performs the worst out of the given trials. For inputs we used the current position of the servos and the x, y and z angles of a gyroscope.

1 Introduction

Robotic walking is a complex task. To write every eventuality of movement into one program takes a lot of time and effort. We can use evolutionary strategies such as Genetic Algorithms to evolve action, however this does not allow intelligent behaviour to adapt to a dynamical world. Open AI developers have been able to use evolutionary strategies [1] to evolve intelligent behaviour such as the ability to play hide and seek, and keep improving as time goes on [2]. In this experiment we will evolve a neural network using a Microbial GA [3] to develop walking abilities. In this paper we outline how the network architecture effects converging on a solution by trialing different network architectures on a biped chassis.

We predict that one hidden layer with a small amount of nodes will perform better than alternatively sized networks and perceptrons. This is because a perceptron will not take into consideration all inputs for the prediction of accurate outputs. A network with multiple layers is likely to see the vanishing gradient problem therefore give less accurate outputs [4].

After experimentation we found that our hypothesis was partially correct where the perceptron would not outperform a network with the multi-layer design. Our prediction was incorrect where the 1 layer did not out perform 2 layers or 3 layers.

1.1 Problem Statement

The aim of the experiment is to drive a chassis of 4 servos to optimize a method of travel. Each servo can turn from 0 to 180 degrees, however constraints in servo rotation will be added to prevent hardware from colliding with one another. This could cause damage to the chassis if metal parts push against each other with too much force. It is the job of the algorithm to mutate a genotype which will optimize a neural network of different architectures. This will include a perceptron, a neural network with one layer and five nodes, a neural network with one layer and 15 nodes, a neural network with 2 layers and a neural network of 3 layers. This scope of architectures will help plot which style of network performs best. Having a series of layers or a series of nodes will give better insight to architecture and parameters needed for the task of walking.

1.2 Previous work

The use of GA neural networks to evolve walking has been done before both hardware and software [5] [6]. The issue with simulations is that it reduces real life 'noise' which a physical robotic system must overcome to learn. This would include attributes such as weight distribution, floor texture and more.

2 Apparatus

This experiment will be using larger quantities of RAM and processing power, therefore will need to use a Raspberry Pi controller [7]. Circuit Python will not be suitable for a task like this due to its limited memory. The Raspberry Pi will be attached to a biped chassis of 4 motors. 4 motors will be enough to provide a low centre of mass to create good balance, while still showing a walking motion. Using more motors than this will take much longer to converge on a solution and for the purpose of this experiment be unnecessary.

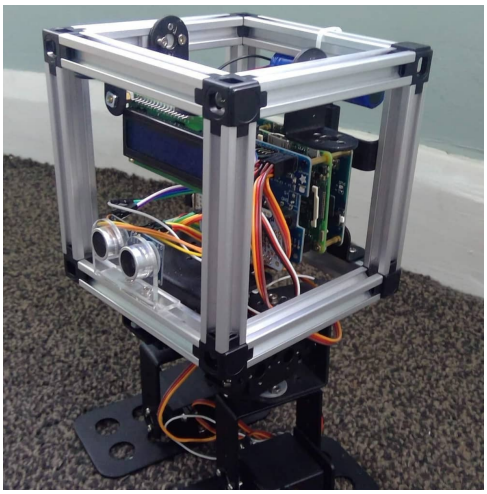

Figure 1: The Biped chassis used was constructed from aluminium for both the hull and biped legs. Electronic components were secured on using M3 screws.

The use of a gyroscope and ultrasound distance sensor will provide fitness of each generation. These are both cheap and small, therefore will be easy to incorporate into the hardware. They will also give insight to where the robot is within its environment without the complexity of a computer vision task.

Due to the larger physical size of the Raspberry Pi, larger servo motors will be needed. We will be using the MG-996R servos which require around 500ma each [8].

$$4 * 500 = 2000ma \tag{1}$$

We used a battery of 3.7V and 3350ma which has plenty of charge for many experiments. It provides a constant flow to the servos. A second battery of 3.7v and 3800ma will be attached to the Pi in order to make the robot mobile. Using the same battery will cause too much of a strain and may damage the battery. The battery was chosen due to it being the most similar to the required amount on offer for compatibility with the Raspberry Pi. The battery cannot go under 2000ma, and it would be pointless paying for more current than needed.

3 Application of GA

Throughout this experiment we will use the Microbial algorithm. A series of random genotypes, representing the weights and biases of a network will be generated. This initial population size will be of size 20 to give enough variation but not too much that behaviour doesn't start to evolve in 150 epochs. At each generation these weights will be mutated and trialed. Two genotypes will be put through a tournament selection and the winner will overwrite the loser. The algorithm will use elitism to decide which genotypes continue into the next generation of populations. Elitism was chosen for its simple design with the code. Using crossover is more complex and random in nature. Either could be used for this experiment.

Algorithm 1 Microbial Algorithm

Require: *population*
  $current1 \leftarrow population[randomInt()]$
  $current2 \leftarrow population[randomInt()]$
  $PerformActions(current1)$
  $fitness1 \leftarrow fitness()$
  $PerformActions(current2)$
  $fitness2 \leftarrow fitness()$
  if $fitness1 > fitness2$ **then**
    $population.AtIndex(fitness2) \leftarrow fitness1$
  else
    $population.AtIndex(fitness1) \leftarrow fitness2$
  end if

3.1 Fitness

Fitness will be measured from three different components, gyroscopic data, distance gained and diversity of movement. In initial tests with neural networks using GAs we found that it would not predict complex movement until many generations in the experiment. By adding a diversity of positions generated score, the robot will evolve more complex movements at an earlier stage. The gyroscopic data will act as a penalty if the chassis is titled too much, which will limit the risk of falling over.

Fitness will be displayed in graphs within the results section. This fitness will be made up of the distance travelled in millimeters added to how many values change in the motor positions. After this, we subtract the penalties mentioned previously. We expect a good fitness in the time to perform an action will measure around 900 to 1000 units. Fitness classed as under good is likely to be around 300 to 800, and anything lower than that is not good. Advancing 10cm within the 20 instructions the robot can use to move its motors is reasonable to achieve given the hardware design.

Through experimentation it was found that the EXPECTEDX and EXPECTEDY was 9.8 and 0.2, respectively, in a plus/minus factor of two.

The values EXPECTEDX and EXPECTEDY represent where the gyroscope stands in 3D space when the robot is standing. There will be given a bound of values that the robot tilt can be within before it is classed as in danger.

3.2 Mutation

Mutation occurs at a standard deviation of 20%, and mean 0. 20% mutation rate has been found to be the optimum rate for GAs [9]. It uses a normal distribution of values to add to the current gene, increasing and decreasing its values by a small

Algorithm 2 Fitness function

Require: $startDistance, stepsTaken$
  $current \leftarrow getDistance()$
  $ground_gained \leftarrow startDistance - current$
  $diversityScore \leftarrow numberOfChanges(stepsTaken)$
  $angleXOff \leftarrow EXPECTEDX - currentXAngle()$
  $angleYOff \leftarrow EXPECTEDY - currentYAngle()$
  if $ground_gained - (angleXOff + angleYOff) + diversityScore > 0$ **then**
    return $ground_gained - (angleXOff + angleYOff) + diversityScore$
  end if
  return 0

amount. It further constrains these values so it does not exceed out of the bounds -4 to 4. This prevents the weights and biases getting too big or too small and making it difficult for the algorithm to evolve.

Algorithm 3 Mutation function

Require: $gene, standardDeviation$
  $gene \leftarrow gene + generateRandomNormalArray(standardDeviation, mean = 0)$
  $gene[values > 4] \leftarrow 4$
  $gene[values < -4] \leftarrow -4$
  return gene

4 Application of neural network

The neural network will use pyTorch. This is due to having more experience with it than TensorFlow or other neural networks. Using pyTorch over creating our own network will save time and offer more functionality than we can provide. Throughout this experiment the networks will change architecture to help us learn what works best and how this evolves behaviour. A gene will carry all the network weights and biases in the order of:

$$layer1weights, layer2weights, ..., layerxweights, biases \tag{2}$$

This will vary in size based on the architecture but will adopt this format. This is the most methodical way of encoding the gene.

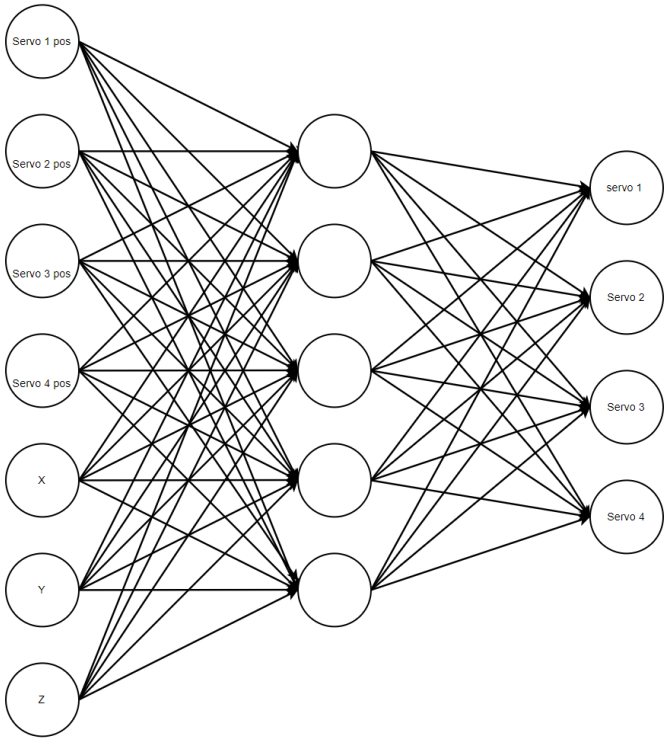

Figure 2: Diagram of the network showing the input nodes and output nodes. Inputted within are the servo positions and gyroscope readings, and outputted are predictions of whether each servo will move backwards, forwards or stay. This will then be converted into an output angle to each servo.

4.1 Implementation

We will be using 5 different network architectures, and 3 trials of each to give a fair average. All neural networks will have 7 inputs, 4 of which will be the current positions of the servos, and 2 will be the x, y and z angles reported by the gyroscope. This will lead through to 4 outputs to predict whether or not the next servos will move forward 30 degrees, back 30 degrees or not move at all. Having these inputs is important as servo position alone is not enough to predict movement in the way we want, especially considering that movement is very symmetrical and will likely perform the same motor positions twice but require different outputs. Involving the tilt will help the agent distinguish when an output is related to the current position. 30 degrees of movement was chosen as it was large enough to be significant but small enough not to topple the robot easily.

The first network will have one hidden layer with 5 nodes. The second will have one hidden layer with 15 nodes. The third will be a perceptron. The fourth will be network of 2 layers, 5 nodes on each. The fifth is a network of 3 layers with 5 on each. These were picked for the reasons previously mentioned.

5 Results

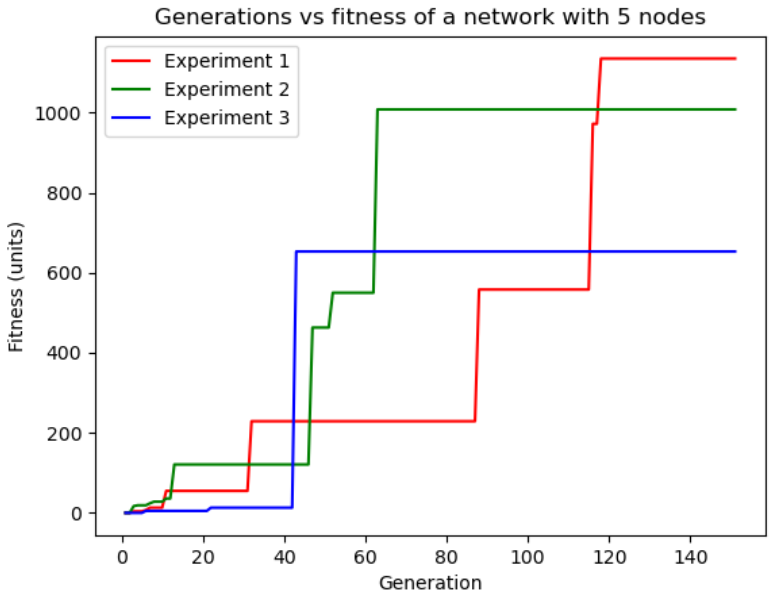

Figure 3: 3 trials of Generations vs fitness of a network with 5 nodes on a single hidden layer. Seen here is the solution which started off lower, but converged on a much higher solution in the last 30 generations.

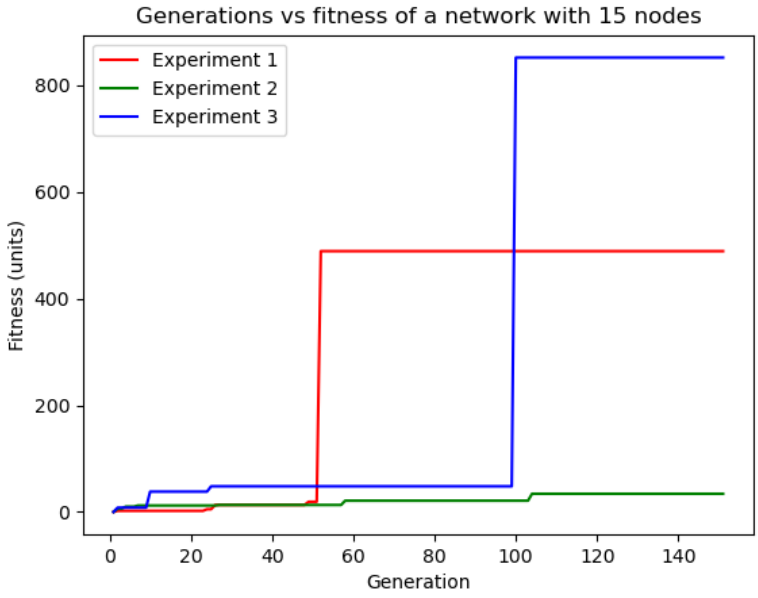

Figure 4: 3 trials of Generations vs fitness of a network with 15 nodes on a single hidden layer. The second experiment never converged on a real solution.

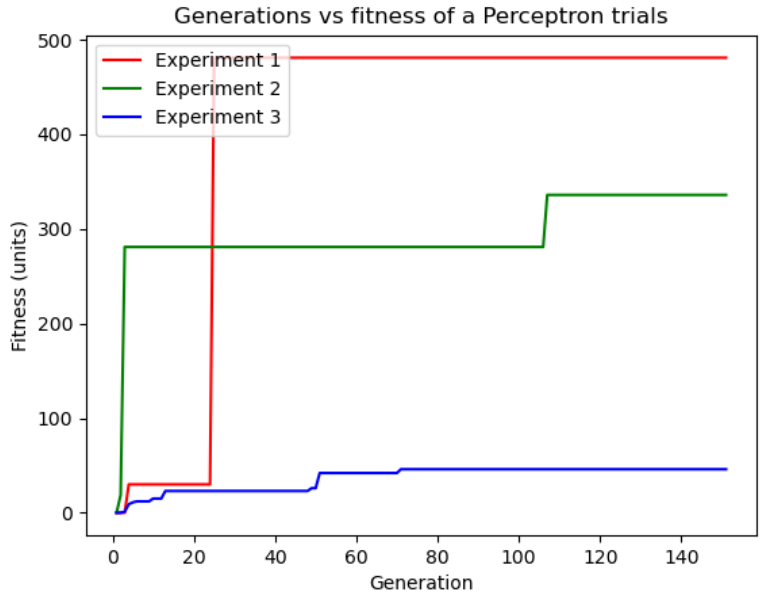

Figure 5: 3 trials of Generations vs fitness of a perceptron. The third trial converged on a sub optimal solution which was very far off a real solution.

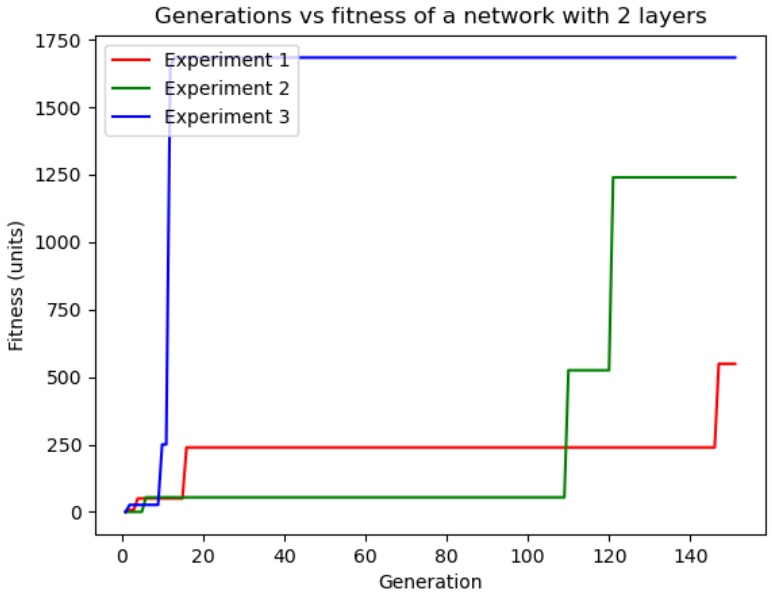

Figure 6: Average of 3 trials of Generation vs fitness of a network of 2 hidden layers, both with 5 nodes on each.

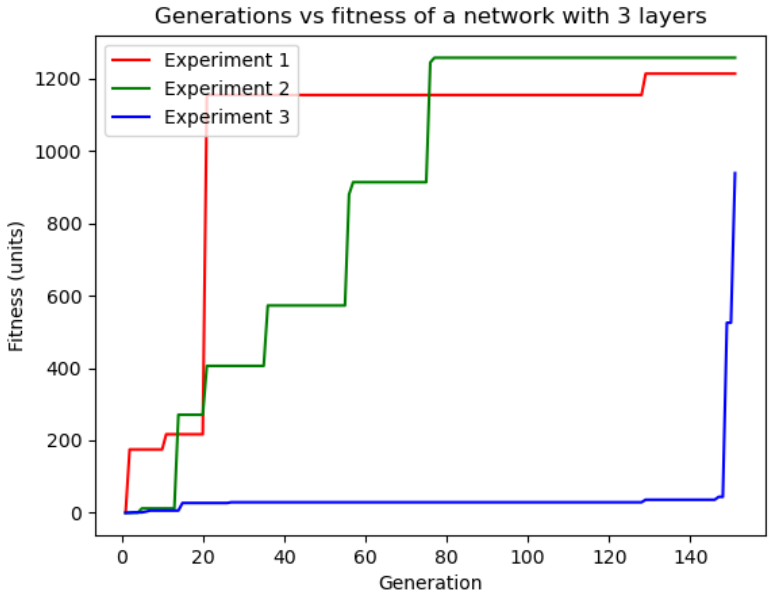

Figure 7: Average of 3 trials of Generation vs fitness of a network of 3 hidden layers, both with 5 nodes on each.

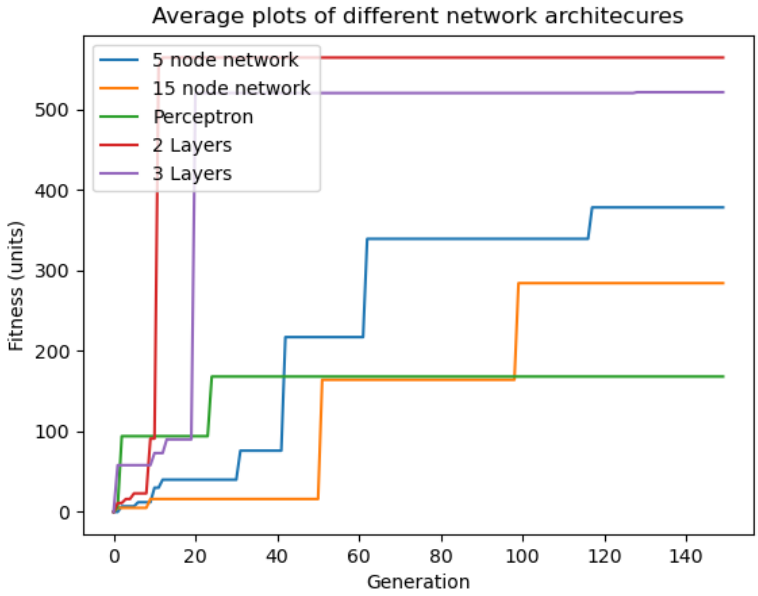

Figure 8: After 150 tournaments we calculate the average fitness at each tournament of each of the 3 trials for all architectures and plot the best fitness of the algorithm. Plotting the averages together shows which architecture proved itself best within the 3 trials. As seen, all algorithms only go up in fitness.

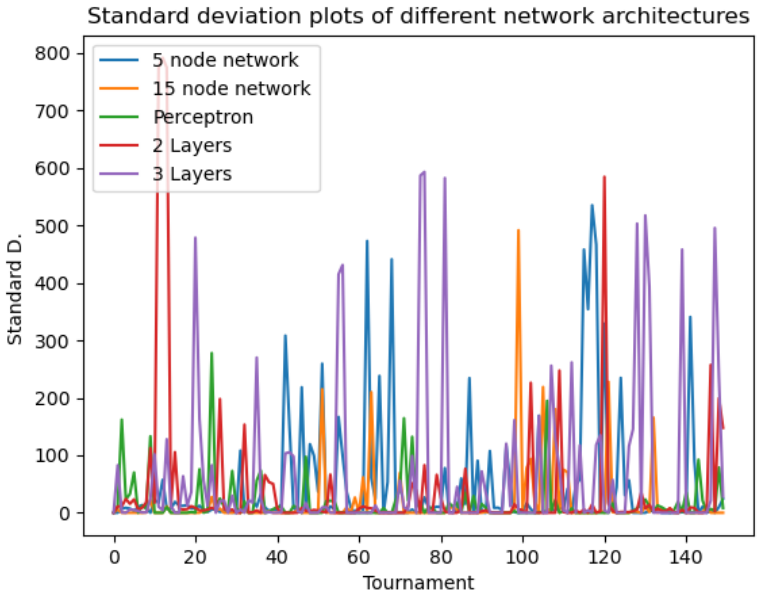

Figure 9: Standard deviation of 3 trials for different network architecture. This shows that the largest changes in standard deviation between the 3 trials belong to the 3 layer network. Each point of the trial was the best fitness of that tournament rather than the best fitness of the overall population at the generation.

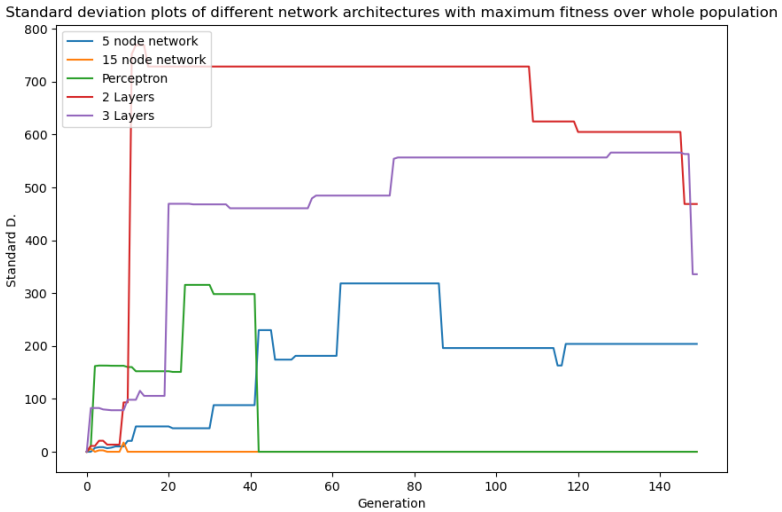

Figure 10: Standard deviation of 3 trials for different network architecture. This was gathered from the best fitness of the whole population rather than the best fitness of the tournament such as Figure 9. As seen it is much clearer than Figure 9 to see where change occurs.

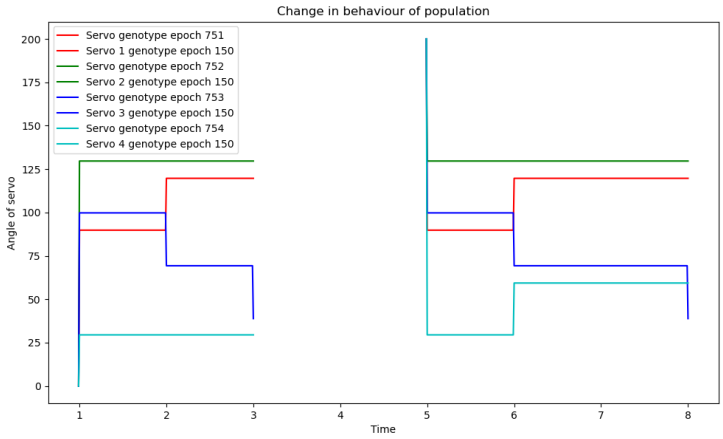

Figure 11: Behaviour change in best performing genotype. This was gathered by getting the fittest of the population at epoch 75, and then assessing the behaviour of this same gene at the end of the generations. Epoch behaviour is separated by the gap in the centre of the graph. This was gathered via the 1 layer 5 node network.

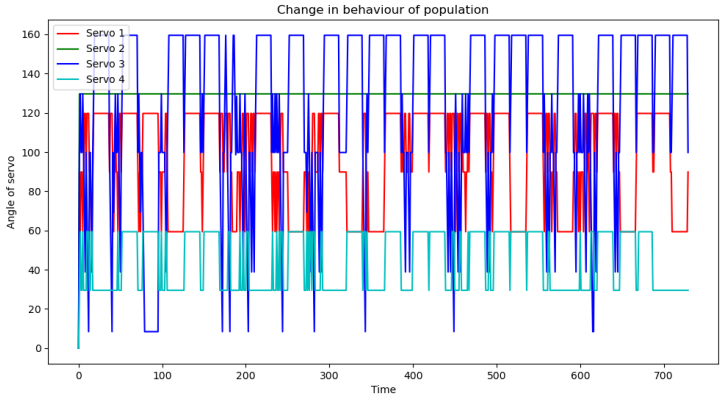

Figure 12: Behaviour change throughout the entire program. This was gathered by storing the current genotype selected from the algorithm. Although patterns can be seen, Figure 11 as a snippet of the same agent behaviour was much clearer and will be referred to in the discussion.

6 Discussion

In Figures 3, 4, 5, 6 and 7 we see 3 trials of each network architecture which shows the relations to the trials to help us find where the fluke trials exist. Figure 3 shows that all 3 trials ended in the higher end of the fitness units.

Figure 4 shows that there was no general area that the solutions would end up in, suggesting it was more luck based that it converged on a solution. The third trial was the best, but is not nearly as high as the best case for the 5 node model.

Figure 5 shows the perceptron which performed the worst out of all the architectures. We believe this was because the model was not able to link and work out relationships between motor positions and gyroscopic data.

Figure 6 shows the network of two layers and how there was a mixture of outcomes, however, two of them outperformed the best of the next best model seen in Figure 3. The lowest of these fitness values outperformed the best of the perceptron seen in Figure 5. Figure 7 shows great performance when using 3 layers. The layers do not perform as well as Figure 6 although end up in closer fitnesses than the 2 layer network trials. 2 layers perform much better on average suggesting that 3 layers is the start of decline into the vanishing gradient problem [4].

This is all summarized within Figure 8 where we can clearly see how well the multi-layer networks outperformed the perceptron. In addition, that 5 nodes was better than alternate patterns, and 2 layers was better than one.

Figures 9 and 10 show us the average distance between trials at specific generations. When taking the best fitness of the whole population, in Figure 10, it is clear that more layers means more variance in effectiveness of a solution. The data suggests the perceptron is limited in effectiveness of solution due to its small size, whereas multiple layers has a wider search space thus more room for varied solution convergence. Figure 10 shows dips in the plots nearing the end generation, which shows the solutions start to meet at higher fitnesses.

Figure 11 shows us that servo 1 and 2 did not change through the evolution, but servo 3 evolved to take longer to move and servo 4 completely changed. This movement shows that the genotype has mutated over time to be a similar series of actions to the past genotypes.

6.1 Observations

Through experiments, it was noticed that quite often one side of the biped would evolve a walking motion, and the other side would slide along [10] (See citation for video). This mirrored activity in walking is difficult to evolve using the microbial GA.

The robot would turn to the nearest wall and walk towards it. This was because it got a better fitness by doing this. Although this was not expected, this behaviour which evolved allowed the robot to cheat the system to improve fitness quicker.

The network of 2 hidden layers evolved more movement in joints at an earlier stage than the other architectures.

6.2 Future work

Future work would include attempt of modifying the algorithm to reflect a more symmetric pattern in the walking. In addition to this, evolving a distance predicting mechanism using depth perception. This would eliminate the need for an ultrasonic range finder. What would further test the reinforcement learning element is to develop behaviours such as chasing a ball through image recognition. The robot would then evolve the ability to move left and right as well as forward based on stimuli.

7 Conclusion

To conclude, this paper found that there was a better architecture to solving the problem of walking.

This paper has shown that using a biped chassis of 4 motors, a network with two hidden layers and 5 nodes within both hidden layers was the best performing out of our sample networks. The common evolved solution of walking included taking a step on one side and jumping the remainder. Advancing this model could use more sensors which would act more on the input layers allowing the robot to behave differently to specific inputs.

Having multiple layers was found to be effective as it gave more layers to evolve a solution, compared to a perceptron where there is less separation between inputs. More layers than 3 is expected to lead to a drop in fitness due to the vanishing gradient.

References

[1] Open AI. Evolution strategies as a scalable alternative to reinforcement learning. https://openai.com/blog/evolution-strategies/.

[2] Stephen Ornes. Playing hide-and-seek, machines invent new tools. https://www.quantamagazine.org/artificial-intelligence-discovers-tool-use-in-hide-and-seek-games-20191118/.

[3] Suryansh S. Genetic algorithms with neural networks. https://towardsdatascience.com/gas-and-nns-6a41f1e8146d.

[4] S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen, (1991).

[5] Shiva Verma. Teach your ai how to walk. https://towardsdatascience.com/teach-your-ai-how-to-walk-5ad55fce8bca.

[6] Tuomas Haarnoja. Learning to walk via deep reinforcement learning, 2019.

[7] Raspberry Pi Foundation. Raspberry pi boards overview. https://www.raspberrypi.org/products/.

[8] Components 101. Mg996r servo datasheet. https://components101.com/motors/mg996r-servo-motor-datasheet.

[9] R.L. Haupt. Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors. In *IEEE Antennas and Propagation Society International Symposium. Transmitting Waves of Progress to the Next Millennium. 2000 Digest. Held in conjunction with: USNC/URSI National Radio Science Meeting (C*, volume 2, pages 1034–1037 vol.2, 2000.

[10] YoutTube. Evolving robotic chassis demonstration. https://www.youtube.com/watch?v=0xNKY13SypU.

8 Appendix

Code

Robot code.py - 1 hidden layer
Robot code deep.py - 2 hidden layers
Robot code deep3.py - 3 hidden layers
Perceptron.py - Perceptron