Evolving a Multi-Layer Perceptron Controller for Landing Spacecraft

40275286

ABSTRACT

A multi-layer perceptron (MLP) is a type of artificial neural network, its goal in this context is to control the output of three thrusters on a spacecraft in order to land it safely on a landing pad with as little fuel left as possible. An evolutionary algorithm was developed to produce the MLP's weights, used in deciding the output of thrusters. In order to improve the fitness, the measure of success, of solutions the parameters for the algorithm were tuned through experimentation to create a MLP controller that was successful in landing spacecraft safely.

1. INTRODUCTION

This report covers the steps taken to create weights for a multi-layer perceptron artificial neural network using an evolutionary algorithm. The algorithm includes different operators for each stage of the evolutionary process which can be set by changing parameters. To find the optimal parameters for the algorithm tests were conducted on training sets and test sets of spacecraft, to determine how different parameters affect fitness. The next section covers the different operators implemented within each stage of the evolutionary algorithm and how they are expected to affect the fitness of solutions produced.

2. APPROACH

In an evolutionary algorithm, a population of solutions is created (initialisation). A loop is then run for a set number of iterations which selects parent solutions (selection), creates children with them (recombination and mutation) and then inserts them back into the population (replacement). There are different methods that can be implemented for each of these stages of the evolutionary process. The algorithm chosen is a steady-state evolutionary algorithm which means individuals can persist throughout cycles.

2.1 Initialisation

Initialisation is the process of creating an initial population. This population is unlikely to contain any usable solutions and so the individuals must be evolved. These are simple to implement operators which are only used once when the evolutionary algorithm runs.

2.1.1 Random

This method involves creating the initial population by randomly assigning each gene. This is good for population diversity as it is unlikely that identical solutions will be created. There is only one parameter for this initialisation, *population size*. The population created by this operator is unlikely to contain many solutions with high fitness.

2.1.2 Seeded

By evaluating randomly produced solutions, a population can be created using only solutions that have a better fitness. This means that the starting population's overall fitness will be better, as opposed to randomly generated populations; however the population diversity is decreased which can lead to premature conversion at a local optimum.

This method introduces a second parameter, *seeded population size*. This is the size of a randomly produced population which is then sorted by fitness and the final population created as a sub-list of length *population size* containing the solutions with the best fitness.

2.1.3 Opposition-Based

Opposition-based initialisation is similar to random initialisation except every time a new solution is created a copy is also created containing the genes of the original that have been switched to the opposite value, improving the chance to start with a fitter solution by checking the opposite solution simultaneously [1]. This results in two opposite solutions which are then evaluated and the solution with the best fitness is added to the population. This is repeated until the population reaches the size set by the *population size* parameter.

2.2 Selection

After a population is initialized the first set of parents must be selected. These parents are used to create children which are added back into the population, it's important that the parents are chosen in a way that increases the likelihood of producing fitter children. The methods used to select parents are as follows:

2.2.1 Tournament

For this method, a specified number (*t*) of individuals are selected at random and compared. The individual with the best fitness is then chosen to become a parent. This gives every solution, except the worst, a chance at becoming a parent with individuals with better fitness being more likely to chosen.

2.2.2 Elitism

For elitism, the individuals with the best fitness are chosen as parents always. This greatly increases selection pressure which will often result in premature convergence however as if the parent individuals do not produce offspring with a better fitness then the parents will be reselected on the next iteration.

2.2.3 Roulette

With the roulette method selection is proportionate to the fitness of an individual, the higher the fitness of an individual the higher the chances of selection [2]. Roulette selection does increase the likelihood of premature convergence when compared to tournament selection but will not exclusively select the best individuals as parents like elitism so selection pressure is not as high.

2.3 Recombination

Once parents have been selected they must reproduce to create offspring. To do this, a crossover (recombination) of genes must occur. For the implementation of these operators two children were produced from two parents. The following operators were added to the evolutionary algorithm to complete this task:

2.3.1 1-Point

For 1-Point crossover a cut-point is randomly chosen along the chromosome length. The genes before the split are assigned from one parent and the rest of the genes are assigned from the other producing a new solution.

2.3.2 2-Point

This method of recombination randomly creates two cut-points. Genes between the cut-points are assigned from one parent and genes outside of the cut-points are assigned from the other.

2.3.3 Uniform

Uniform crossover creates a child by considering each gene one at a time and randomly assigning one of the parent's genes to the child.

2.3.4 Arithmetic

Arithmetic crossover creates a new child using the average value for each gene from each parent. This results in a child that's the average of each parent solution.

2.4 Mutation

Once the children individuals have been created, before they are added to the population, they are mutated. Mutation randomly changes the value of genes on a chromosome based on the following parameters:

- *Mutation Rate* – The probability of a gene being mutated.
- *Mutation Change* – The value that a mutated gene is changed by.

Mutation adds diversity to the population and introduces new solutions for consideration that otherwise may not have been created.

2.5 Replacement

After children individuals are derived from the selected parents they are introduced back into the population for the next evolutionary cycle. Two operators were implemented for this:

2.5.1 Worst

Using worst replacement, the fitness of the population is evaluated and the individuals with worst performance are replaced by the children created in the previous stage.

This method ensures that the average fitness for the whole population is continually improving.

2.5.2 Tournament

Similar to the tournament method for selection, a set number of individuals (*t*) are randomly chosen from the population. The individual with the worst fitness is then replaced by a child. This is repeated for each child.

3. EXPERIMENTS AND ANALYSIS

The initial plan for experimentation with parameters, to improve the fitness of solutions created by the evolutionary algorithm, was to begin with the default parameters. Once an average fitness was calculated an iterative approach would be taken to alter parameters. Each parameter would be changed to a range of values and the results recorded. These would be assessed and the optimal parameters would be chosen and the process repeated again until a suitable fit solution was discovered.

Upon beginning experimentation a few problems emerged with this approach. Most of the initial tests showed that with the default parameters most solutions where converging at a local optimum giving a fitness of ~0.12, with changing a single parameter at a time not having a significant effect on fitness. Whilst this approach would have eventually improved solution fitness it would have been very time consuming.

To counter these problems, instead of using the default parameters, the parameters were set to use operators and values that were thought could lower the fitness based on details described in the approach section. These were then altered until the parameters consistently gave solutions with fitness lower than 0.12 meaning the point of convergence wouldn't be an issue. A range of parameter values were then tested using this initial parameter selection and a revised set of parameters were chosen based on results.

3.1 Initial Parameters

Each test for a set of parameters was run ten times, all the results shown in the following sections are an average of the ten runs for both the training and testing set. Data for each individual run can be found within the results folder within the project code.

The following parameters were used initially. Each test ran uses these parameters unless shown otherwise:

- Hidden Nodes: 5
- Min/Max Gene: -3/+3
- Population Size: 60
- Mutation Rate: 0.6
- Mutation Change: 0.5
- Initialisation: Seeded
- Seeded Population: 1000
- Selection: Roulette
- Crossover: 2-Point
- Replace: Worst
- Activation Function: tanh

The average fitness results for these parameters were:

Training Set	0.0388969
Test Set	0.0941109

The goal of the tests in the following section was to show how different parameter values affected fitness so that they can be changed for improvement.

3.2 Parameter Tests

This section covers the results of all tests undertaken for each parameter.

3.2.1 Hidden Nodes

Too many hidden layers can increase computation cost and increase run time. Therefore the range of hidden nodes tested was between one and ten. The results are shown below.

Hidden Nodes	Training Set	Test Set
1	0.0339809	0.0588714
2	0.0521649	0.1100600
3	0.0515192	0.1055780
4	0.0537746	0.1179940
5	0.0388969	0.0941109

6	0.0260654	0.0539394
7	0.0415033	0.0893673
8	0.0259056	0.0533625
9	0.0631966	0.1345760
10	0.0431464	0.0845777

From these results 1, 6 and 8 hidden nodes improved the average fitness. As 6 and 8 hidden nodes give similar average fitness, 6 would be chosen for the revised parameters in order to keep computation cost to a minimum.

3.2.2 Population Size

As shown by the results below, a population of size 20 performed the best. As the initialisation for all of these tests uses the seeded population operator this may be due to a smaller number of individuals in the initial population resulting in a better average fitness.

Population Size	Training Set	Test Set
10	0.0364481	0.0598345
20	0.0319116	0.0693845
40	0.0381054	0.0808265
60	0.0388969	0.0941109
80	0.0580490	0.1089780
100	0.0493316	0.0978134

Though a population of size 20 will be considered for the revised parameters, depending on how well other initialisation operators perform the value chosen may end up being higher.

3.2.3 Initialisation

The results for testing initialisation operators are shown below. Three different values for the seeded population were used, 500, 1000 and 1500. It was originally theorized that a higher value for the seeded population would result in a better fitness however this is not the case:

Initialisation	Training Set	Test Set
Random	0.0402826	0.0869763
Seeded(500)	0.0389114	0.0938961
Seeded(1000)	0.0388969	0.0941109
Seeded(1500)	0.0398866	0.1076090
Opposition-Based	0.0356066	0.0654588

The results show that opposition-based initialisation performed the best. This is probably due to the operator creating a diverse population with an average fitness higher than a population created randomly.

3.2.4 Selection

Selection tests were conducted using the tournament and roulette operators. A range of tournament participant (*t*) values were used, the results are shown below:

Selection	Training Set	Test Set
Tournament(t=2)	0.0461334	0.0976926
Tournament(t=4)	0.0432727	0.0947774
Tournament(t=6)	0.0407432	0.0820946
Tournament(t=8)	0.0579511	0.1079000
Tournament(t=10)	0.0450471	0.1042910
Roulette	0.0388969	0.0941109

Roulette selection performed better than tournament selection regardless of the value of *t*. Roulette selection allows for worst individuals to be picked from the population more than tournament selection which results in more diversity in chosen parents.

3.2.5 Crossover

Each crossover operator was tested, with the results displayed below:

Crossover	Training Set	Test Set
1-Point	0.0281285	0.0557576
2-Point	0.0254938	0.0442971
Uniform	0.0525703	0.1088720
Arithmetic	0.0314304	0.0612857

2-Point crossover performed much better than the other operators so will likely be used for the revised parameters also.

3.2.6 Mutation

The mutation rate, the probability of mutation occurring was altered and the results are shown below:

Mutation Rate	Training Set	Test Set
0.05	0.1025530	0.2198290
0.1	0.0911903	0.1984460
0.25	0.0566614	0.1215240
0.5	0.0474098	0.0925476
0.75	0.0437706	0.0856343
1	0.0490853	0.0828588

The fitness of solutions initially decreases as mutation rate increase but begins to increase again after 0.75; this is likely due to too many mutations creating solutions too far from the individual that was mutated resulting in a worse fitness.

The *mutation change* was unfortunately not tested due to time constraints.

3.2.7 Replacement

Replacement tests were conducted using the tournament and worst operators. A range of tournament participant (*t*) values were used, the results are shown below:

Replacement	Training Set	Test Set
Tournament(t=2)	0.0968987	0.1942740
Tournament(t=4)	0.0810496	0.1666450
Tournament(t=6)	0.0935997	0.1734590
Tournament(t=8)	0.1037550	0.2075380
Tournament(t=10)	0.0940024	0.1811760
Worst	0.0388969	0.0941109

Replacing individuals of the population with the worst fitness is shown to be the best strategy.

4. CONCLUSIONS

Following the experimentation, the best performing operators and parameter values were chosen to form a set of revised parameters. The revised parameters are as follows:

- Hidden Nodes: 6
- Min/Max Gene: -3/+3
- Population Size: 50
- Mutation Rate: 0.65
- Mutation Change: 0.5
- Initialisation: Opposition-Based
- Selection: Roulette
- Crossover: 2-Point
- Replace: Worst
- Activation Function: tanh

The average fitness results for these parameters were:

Training Set	0.0320884
Test Set	0.0526845

From parameter experimentation and tuning the training set average fitness was reduced by 16.5% and the test set average fitness was reduced by 44.0%.

This is unlikely to be the optimal parameters for the evolutionary algorithm but through testing of the weights with test fleets of spacecraft it's demonstrated that the weights produced by the algorithm enable the MLP controller to land spacecraft safely and consistently.

The *mutation rate* parameter had more of an effect on fitness than initially expected. The default mutation rate was very low (0.05) which is possibly the reason initial tests were all converging prematurely as there was not enough diversity being introduced. Raising the rate of mutation from this value showed an immediate improvement in fitness.

The initialisation operator used had the least significant effect on fitness, which is perhaps due to the random nature of producing each single individual regardless of how the population members are selected.

5. FUTURE WORK

To improve the algorithm in the future, another parameter could be added to choose whether the population will be generational or stead-state. The implemented algorithm was steady-state but if the algorithm was generational then each iteration of the evolutionary loop would use a new population produced by the previous loop. It's unknown if this would result in lower fitness values but this could be revealed through testing.

Different activation functions were not tested during this project, but altering the activation function could have resulted in better fitness results. The evolutionary algorithm implemented uses a tanh activation function but others that could have been tested are Sigmoid, ReLU and ELU.

With the knowledge that mutation rate had a substantial effect on fitness, if that had been the first parameter evaluated, then the initial plan for parameter testing may have been more appropriate. If the project was to be undertaken again this is the approach that'd be taken to parameter testing.

6. REFERENCES

[1] Rahnamayan, S., Tizhoosh, H. R., & Salama, M. M. A. (2007). A novel population initialization method for accelerating evolutionary algorithms. Computers and Mathematics with Applications. https://doi.org/10.1016/j.camwa.2006.07.013

[2] Saini, N. (2017). Review of Selection Methods in Genetic Algorithms. https://doi.org/10.18535/ijecs/v6i12.04