Investigating the optimal gene mutation mechanism for the purpose of neural network weight evolution in the Evoman game-playing framework

Evolutionary Computing, Task 1: specialist agent, team 35

Namitha Tresa Joppan
Vrije Universiteit
Amsterdam
Student Number: 2685670
namitha.joppan@student.uva.nl

Maud Bremer
Vrije Universiteit
Amsterdam
Student Number: 2607583
maud.bremer@student.uva.nl

Fiona Gallagher
Vrije Universiteit
Amsterdam
Student Number: 2686848
fiona.gallagher@student.uva.nl

Jasper de Koning
Vrije Universiteit
Amsterdam
Student Number: 2544454
dekoningjasper@gmail.com

1 INTRODUCTION

Evolutionary computation (EC) tries to find global optima using a mechanism inspired by biological evolution. Similar to natural selection, the algorithm prefers individuals with higher fitness. These individuals are allowed to recombine and mutate after which a new generation is selected. By repeating these steps, the population of individuals will evolve into a solution with higher fitness. In order for the algorithm to keep variance in the population and not terminate at a local optimum instead of a global optimum, mutation operators can be applied. These will alter the genotypes of individuals in the population randomly. Evolutionary algorithms can consist of various selection, mutation and recombination operators. Due to the fact that each of the components of such an algorithm can be adapted, different evolutionary algorithms can be constructed to fit the problem.

Evolutionary algorithms can also be applied to games where the optimal solution will be able to win the game with as much resources left, or win the game as quickly as possible. The field of computational intelligence attempts this by using machine learning techniques to create fully autonomous agents [4]. One of the approaches is neuroevolution that is commonly used to play agent games [2, 7]. Here, the networks consists of interconnected nodes that will yield an outcome based on its input and the weights of the node and will form the genotype of the individual [6]. An example of a video game to train these algorithms on is Evoman [1]. The platformer game consists of a player who tries to defeat the enemy by dodging and shooting the enemy, and either wins (by killing the enemy) or dies. The player can encounter eight different enemies, each having a different weapon and tactic, which are all described da Silva Miras de Araujo et al. [1]. By applying an evolutionary algorithm on this game, the player will learn to make the correct choices and change its behaviour based on sensors and inputs from its environment. [5]

Two neuroevolution algorithms will applied on the Evoman game. Three different enemies are selected: Airman which will be called Enemy 2, Metalman which will be called Enemy 5 and finally Crashman which will be called Enemy 6. These three enemies and their tactics are explained in subsection 2.4. The two algorithms differ in the mutation operator, where one will mutate based on a Gaussian distribution and the other on uniform distribution. Since mutation operations are important in creating more diversity in the population and in that way extending the search space, it is important to understand the differences between mutation operators. Using a Gaussian distribution the gene is mutated by adding or subtracting from the current value, while the uniform distribution will replace the gene with any number between -1 and 1. The resulting best fitness values will be compared between the two evolutionary algorithms. The hypothesis is that these evolutionary algorithms would perform similarly.

2 METHOD

2.1 Experimental setup

The goal of this experiment is to develop an evolutionary algorithm where the Evoman game is won by the player, or in this case an AI. In this algorithm the player fights against an enemy for several generations while evolving his strategy. There are a few possible moves: walk left, walk right, shoot, jump and stop jumping. The stop jumping is in order to determine the height of the jump. If stop jumping is not used, the player will jump to the maximum height. The amount of times an enemy or player can take damage, is measured in terms of energy. A shot from either the enemy or player depletes the energy of the opponent. Additionally, when the player makes contact with the enemy, the player will lose some energy as well. Both enemy and player start with an energy level of 100. The player wins if the enemy he is fighting against has zero energy left. In the evolutionary algorithm he has however two ways of losing: either the time runs out, or his own energy depletes. After the fight is over, the fitness is calculated using Equation 1. In this equation e and p are the energy levels of the player and enemy, respectively. And t is the total number of time steps necessary to end the fight. It can be observed that the time a fight takes gives a penalty to the fitness. Other components that influence the outcome of the fitness are the energy of the player and the enemy. The more energy the player has left, the higher the fitness. This is opposite for the energy of the enemy.

$$f = 0.9 \cdot (100 - e) + 0.1 \cdot p - \log t \qquad (1)$$

After an iteration, a certain amount of parents are selected and tournament selection is used to determine which of these parents can create offspring incorporated in the upcoming generation.

2.2 Parameter settings

The evolutionary algorithms in this paper are implemented using the DEAP framework and differ in the mutation mechanism [3]. The EAs use a neural network in which the weights are evolved. Multiple mutation algorithms namely, Uniform mutation, Gaussian mutation, Flipbit mutation, and ShuffleIndexes mutation were examined. This was done in combinations with varying mutation probabilities, crossover probabilities, numbers of neurons and numbers of parents. The Blend crossover was used in all scenarios with α set to 0.5 and the parent selection and survival selection were fixed to be Tournament selection with a tournament size of 3. A tournament size of 3 was chosen to have more genotypes to compete against each other, therefore by increasing the tournament size, the selection pressure also increases. The different algorithms ran over a population of 50 individuals for 50 generations in every single run and each generation 20 parents were selected to create new offspring. The genes in the genotype were initialized randomly with values between -1 and 1. The difficulty of the game was set at 2 and there was a time limit of 1000 timesteps. The best fitness values were observed for Uniform and Gaussian mutation methods and further parameter tuning was performed by varying crossover and mutation probability and the number of neurons. A higher fitness value was observed for both mutation algorithms in combinations of higher mutation probability and lower crossover probability, and for lower mutation probability and higher crossover probability. It was theorized that this was due to the fact that high crossover probability in combination with a high mutation probability results in too much randomness, and evolution is just as easily discarded as it is produced.

Finally, Uniform (ranging from -1 to 1) and Gaussian ($\mu = 0$, $\sigma = 1$) mutation algorithms were applied with an independent mutation probability of any gene (weight) being mutated set at 0.2.

The overall mutation probability for any individual to be targeted was 0.8 while the crossover probability was 0.3. This was for a total of 20 neurons with Tournament selection for both parent and survival selection, and Blend crossover for 50 individuals over 50 generations. Both algorithms were evaluated for a total of 10 different initial populations by setting a unique random seed for each one and the best results were chosen.

2.3 Budget

While running the evolutionary algorithm, most of the computing time was spent on playing the game to calculate the fitness function, rather than applying mutation, recombination and selection on the population, which were computationally less heavy. The number of generations and the population size also resulted in increasing computation time. Taking double the number of generations or population leads to an approximate doubling of computation time if the percentage of parents that created offspring stays the same. Addition of more hidden layers was also computationally expensive, but this was not as explicit as in the increase of generations or population. The simulations for Enemy 6 were the longest compared to other enemies. This could be due to the fact that not every iteration was won against this particular enemy, which will be discussed later. Sometimes the timer ran out resulting in a longer fight. As mentioned, this resulted in more computing power and time and thus a higher budget. All in all, one simulation with the parameters discussed in subsection 2.2 took between one or two hours depending on the computer used.

2.4 Enemies

Enemies were chosen based on similar behaviour so that the final evolutionary algorithm would perform well against all of them. Enemy 5 was chosen initially with the capability to jump up and down while shooting. Additionally, the floor contains a conveyor belt that moves left and right. Enemies 2 and 6 exhibit similar behavior of jumping up and down, with unique characteristics of their own such as, blowing wind by enemy 2 and dropping bombs and jumping towards the player by enemy 6. Another difference between enemy 6 and the other two is that when playing against enemy 6, the game has a time limit of 2200 time steps instead of 1000.

3 RESULTS AND DISCUSSION

The results for the two different EAs are as follows. For each algorithm and each of the three enemies, 10 independent experiments with the parameter settings outlined in section 2.2 were conducted. This resulted in a total of 60 experiments each of 50 generations.

Figure 1 demonstrates the behaviour of our algorithms for Enemy 2. The dashed lines indicate the maximum fitness value corresponding to the best individual in each generation averaged over the 10 experiments. The solid lines indicate the mean fitness value in each generation similarly averaged over the 10 experiments. The evolutionary algorithms yield comparable fitness values over all generations. The average fitness starts quite low, but quickly converges to a maximum after a few generations. Something interesting to note is that the average maximum fitness in the first generation is immediately close to the final converged fitness. This indicates that our initial population is diverse enough to contain an individual with a suitable genotype capable of defeating the enemy.

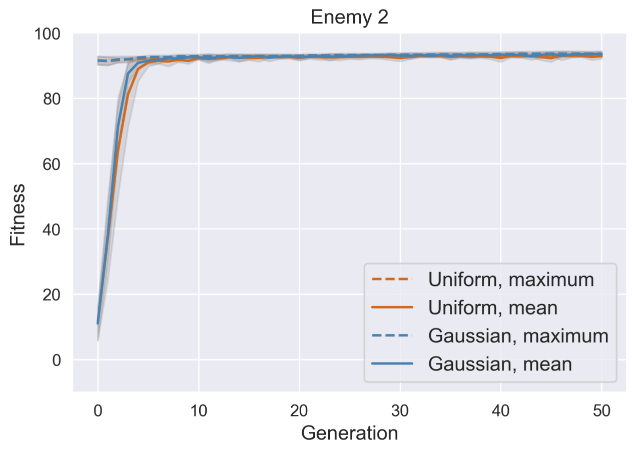

Figure 1: Algorithm comparison of fitness across generations (average over 10 independent runs) for enemy 2

The plot for Enemy 5 in Figure 2 is similar to that of Enemy 2. We note that these enemies have close behaviour as outlined in Section 2.4, which offers an explanation for the results.

Turning to the results for Enemy 6 as seen in Figure 3, the overall fitness is significantly lower. As previously described, Enemy 6 is more challenging for the player due to the fact that it jumps towards the player. Additionally, the player is unable to touch the enemy without losing energy. The initial parameter tuning was undertaken with Enemy 5 which may contribute to the disparity in fitness results. It also seems that the overall fitness does not converge within 50 generations - we see some continuing instability unlike the previous two enemies. Table 1 indicates the fitness of the final converged individual in the population after 50 generations along with the standard deviation over the 10 experiments.

In order to compare the performance of our algorithms, we selected the best performing individual solution from each of the 10 independent experiments (regardless of generation). The best candidate was tested against each enemy 5 times to account for stochasticity in the Evoman game-playing framework. The results are seen in Figure 4 where measure of individual gain is used:

$$\textbf{individual gain} = \textbf{player energy} - \textbf{enemy energy} \qquad (2)$$

It is apparent that both the EAs with Gaussian and Uniform mutation behave well on Enemy 2 and 5, and poorly on Enemy 6. Indeed, both the Gaussian and Uniform mutation show games where the player lost, even after the selection of the best solution.

	Enemy 2	**Enemy 5**	**Enemy 6**
Gaussian	93.5 ± 0.9	93.9 ± 0.2	75.0 ± 13.0
Uniform	92.9 ± 0.9	93.6 ± 0.8	71.0 ± 12.0

Table 1: Fitness statistics for final converged solution after 50 generations (1-σ uncertainty, 3 s.f.)

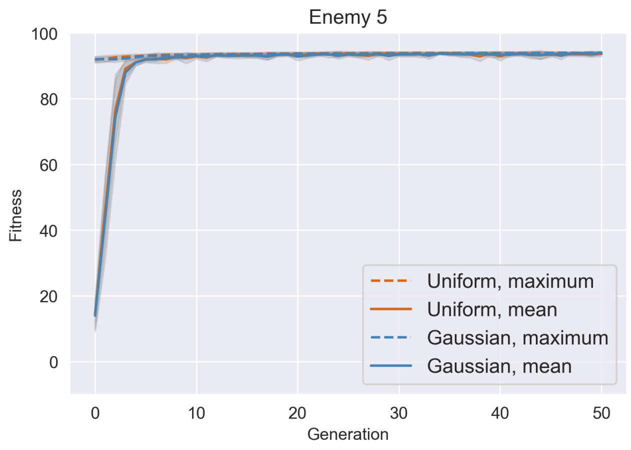

Figure 2: Algorithm comparison of fitness across generations (average over 10 independent runs) for enemy 5

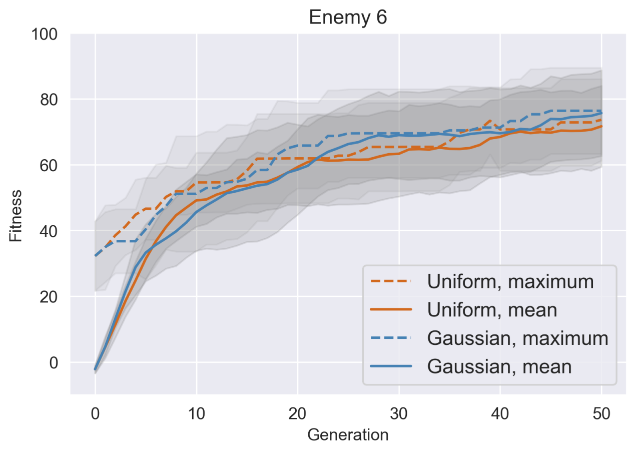

Figure 3: Algorithm comparison of fitness across generations (average over 10 independent runs) for enemy 6

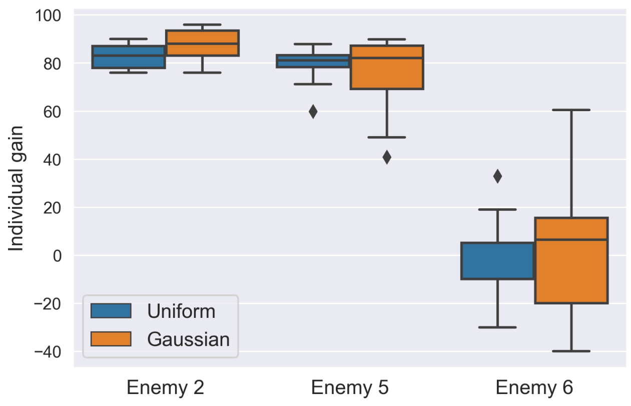

Figure 4: Best solution mean performance for 10 runs for each configuration

In the baseline paper [1], two algorithms (LinkedOpt (LO) and Genetic Algorithm (GA)) are comparable to our own where the weights of neural network are evolved. We selected 20 neurons while they selected either 10 or 50. The final result for Enemy 6 against the GA show the algorithm losing to the enemy. We obtain similar results as seen in Figure 4 with a negative individual gain indicating a loss. However, both our Gaussian and Uniform mutation EAs do result in some instances where Enemy 6 is beaten. Enemies 2 and 5 are beaten by GA, and we see the same for our two EAs. A clearer comparison between algorithms requires the fitness results.

3.1 Statistical comparison of the two EAs

In order to more rigorously compare our two evolutionary algorithms, a 2 sample t-test was performed on the samples of the best performing individual solution from each of the 10 independent experiments (Figure 4). The null hypothesis is stated as follows: there exists a zero difference in the average of means for both evolutionary algorithms and for all enemies. The significance level is kept at 0.05. The underlying assumption was that the distribution of individual gains was normal for each independent experiment. This is difficult to verify, because the sample size is low ($n = 10$ for each configuration). However, the assumption was made due to the fact that the population distribution for all repetitions of the game is normal, regardless of stochasticity in Evoman gameplay.

This t-test results in p-values 0.637, 0.486 and 0.523 for Enemy 2, 5 and 6 respectively. The p-values being greater than 0.05 indicate that the null hypothesis is accepted and it can be concluded that there is no difference in the average of means of both algorithms. Hence, there is no significant statistical difference between the average of means of both algorithms. Thus, the two evolutionary algorithms yield similar results for all three enemies.

4 CONCLUSION

In this report, the question of which mutation mechanism in an evolutionary algorithm results in the best solution for guiding an AI against a static enemy in the Evoman game was investigated. The parameter tuning stage involved an exploration of parameter space wherein the final parameters yielding the best overall fitness were selected. The resulting best parameters included a high mutation probability and low crossover probability. This parameter configuration yields an ideal balance of randomness and solution survival allowing for an optimal traversal of the adaptive landscape. The final performance of our algorithms were similar on each of the three enemies studied. A t-test further demonstrated the similarity of our algorithms. It can be concluded that the best found fitness of both Gaussian and Uniform mutation mechanisms are the same for the purpose of evolving weights in a neural network to play the Evoman game.

REFERENCES

[1] K. da Silva Miras de Araujo and F. O. de Franca. 2016. Evolving a generalized strategy for an action-platformer video game framework. In *2016 IEEE Congress on Evolutionary Computation (CEC)*. 1303–1310.

[2] Dario Floreano, Peter Dürr, and Claudio Mattiussi. 2008. Neuroevolution: from architectures to learning. *Evolutionary Intelligence* 1, 1 (01 Mar 2008), 47–62. https://doi.org/10.1007/s12065-007-0002-4

[3] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* 13 (jul 2012), 2171–2175.

[4] Simon M. Lucas. 2008. Computational intelligence and games: Challenges and opportunities. *International Journal of Automation and Computing* 5, 1 (01 Jan 2008), 45–57. https://doi.org/10.1007/s11633-008-0045-8

[5] Risto Miikkulainen, Bobby D. Bryant, Ryan Cornelius, Igor V. Karpov, Kenneth O. Stanley, and Chern Han Yong. 2006. Computational Intelligence in Games. In *Computational Intelligence: Principles and Practice*, Gary Y. Yen and David B. Fogel (Eds.). IEEE Computational Intelligence Society, Piscataway, NJ. http://nn.cs.utexas.edu/?miikkulainen:ci06

[6] S. Risi and J. Togelius. 2017. Neuroevolution in Games: State of the Art and Open Challenges. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 1 (2017), 25–41.

[7] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. 2005. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation* 9, 6 (2005), 653–668.