Application of Differential Evolution to create an autonomous playing agent for EvoMan

Evolutionary Computing task 1 - group 9

Meifang Li
2719570
Vrije Universiteit
Amsterdam, the Netherlands

Maaike Scholten
2686118
Vrije Universiteit
Amsterdam, the Netherlands

Wenbo Sun
2718218
Vrije Universiteit
Amsterdam, the Netherlands

ABSTRACT

A game playing agent is an ideal simulation of human decision making in complicated scenarios. [3] proposed neural network game playing agents and utilized the standard Genetic Algorithm (GA) to optimize the weights. In this paper, we introduce the Differential Evolution (DE) algorithms to optimizing the game playing agent and compare the performance of DE and GA on the EvoMan [2] framework. We show that DE with best parent selection outperformed the GA in an evident statistical significance. The result conclude that the DE illustrates impressive adaptation on game playing agent optimization.

1 INTRODUCTION

Video games are ideal platforms to develop and test machine learning models, because games provide vivid and visualized scenarios to create agents simulating human-like decision making. Many researchers have utilised machine learning techniques to construct automatic gamers. The most recent spotlight focuses on neural networks. [3] has offered a neural network game player tested with the EvoMan [2] framework, resulting in an impressive performance. Optimizing a neural network player is challenged by game playing scenarios, because the result of a boss fight in EvoMan depends on an action sequence, which means a gradient-based optimizer is not feasible anymore since the gradient updating relies on particular input but neglects context.

To optimize the game-playing agent, the authors implemented a standard Genetic Algorithm (GA) to simultaneously evolve weights of the neural network. As a vital part of an evolutionary algorithm, tournament selection was chosen as parents selection strategy. Intuitively, best-picking and recombination by averaging gradually narrows down the population distribution. The problem with this is the fact that escaping from local optima becomes harder, because the averaging shrinks population distribution. An extra individual mutation may expand the search space. However, another drawback emerges because the search strategy becomes a trail-and-error trick, resulting in a slower convergence speed. Last but not least, the fitness setting is a tricky technique in standard GAs. When a fitness function generates a non-differentiable or non-linear space, the GA will seldomly find the global optimum.

To overcome these flaws, Rainer Storn and Kenneth Price [5] introduced the Differential Evolution (DE) algorithm. DE is a direct search method designed to solve multi-dimensional real value optimisation problems. Figure 1 demonstrates a direct searching example. The largest difference between GA and DE is that the DE is a stochastic direction seeking procedure that expands search space rather than crowding towards a narrow zone. This mechanism gives offspring the chance to escape from local optima and the ability to handle non-differential space. Moreover, with the population evolving, overall fitness is high and relative vector length is getting smaller. Once a proper direction emerges, it helps the whole population arrive at optima faster, indicating DE has a better convergence property.

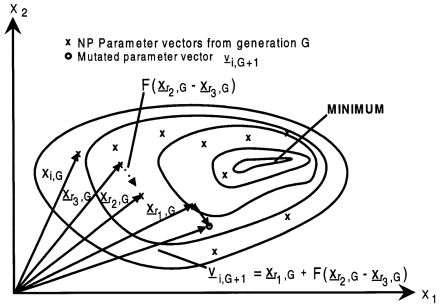

Figure 1: An example of 2-dimensional optimization showing the direct searching process.

Due to the advantages of DE as compared to the GA, two DE algorithms are designed and compared to the GA baseline performed by [3]. The goal of this process is to determine whether DE algorithms provide better results than the GA in terms of playing agent performance against three different enemies in the EvoMan game. Due to the advantages of DE over GA as described above, a better agent performance is expected.

2 METHODS

The GA baseline [3], to which the created algorithms will be compared, defines the crucial crossover operator, which is given by a weighted average of two parents where x denotes the individual genotype, as follows:

$$x_{g+1} = \alpha \cdot x_{g,1} + (1 - \alpha) \cdot x_{g,2} \tag{1}$$

The parameter α is generated uniformly. The parents selection operator uses tournament selection to pick out two relatively high fitness parents to execute crossover.

To overcome the flaws associated with this parents selection method, as described in the Introduction, a DE method is applied.

2.1 Differential Evolution Algorithm

The initial DE population is usually uniformly chosen. By selecting three parents, DE generates a new vector by adding a weighted

difference between two individual vectors to the base vector. A formal expression of candidate vector generation is given by:

$$v_{g+1} = x_{g,1} + F \cdot (x_{g,2} - x_{g,3}) \tag{2}$$

2.1.1 Parents selection. As a symbolic parameter, parents selection has many variants [1]. One option is randomly choosing parents, denoted in this paper as EA1, where

$$v_{i,g+1} = x_{r1,g} + F \cdot (x_{r2,g} - x_{r3,g}) \tag{3}$$

This method could maintain relatively high diversity.

Alternatively, a quicker success can be chased by selecting the best individual as the base vector denoted by EA2, where

$$v_{i,g+1} = x_{best,g} + F \cdot (x_{r1,g} - x_{r2,g}) \tag{4}$$

This method attempts to get close to optima faster.

2.1.2 Recombination. After obtaining candidate vectors, a probabilistic crossover will occur to generating potential offspring. Potential offspring are represented by:

$$u_{i,g+1} = (u_{i1,g+1}, u_{i2,g+1}, \cdots, u_{in,g+1}) \tag{5}$$

The crossover operator is given by:

$$u_{ij,g+1} = \begin{cases} v_{ij,g+1}, & random(i) \leq CR \\ x_{ij,g}, & random(i) \geq CR \end{cases} \quad j = 1, 2, \cdots, n \tag{6}$$

Parameter CR is a predefined constant $\in [0, 1]$, controlling the crossover occurrence probability.

2.1.3 Evaluation. Evaluation of individual performance involves the EvoMan scenario, integrating player's energy, enemy's energy and time cost. We set the fitness function as the following expression:

$$f = 0.9 * (100 - p) + 0.1 * e - \log t \tag{7}$$

where p and e represent the remaining energy of the player and the enemy after combat. t is the total time cost of combat. Each fight is limited to 1000 seconds.

2.1.4 Offspring selection. The target of this algorithm is optimizing the player's fitness. If the player could defeat the enemy in a short time while saving energy as much as possible, the player gains a high fitness. Therefore, to decide whether the individual survives into the next generation, the potential offspring $u_{i,g+1}$ is compared to the corresponding individual in last generation $x_{i,g}$. If $u_{i,g+1}$ has a higher fitness value, it will be selected as a formal offspring $x_{i,g+1}$, otherwise the old one will survive.

2.2 Experiment setting

To examine the performance of DE in the neural network weights optimization task, we set an identical neural network player with the same topology to challenge the EvoMan video game. The standard GA [3] was taken as a baseline to compare them in convergence speed, average population fitness and average best fitness. Additionally, to examine the adaption of DE in various scenarios, each of the two algorithms (DE and GA) trained three independent neural networks to combat with three enemies. Since the evolutionary algorithm is basically stochastic, the initial uniformly generated population coupled with random search brings uncertainty to the final performance. Thus, the training process was repeated ten times for each algorithm for each enemy to create an average to measure overall rather than accidental performance.

We have also explored how the parents selection mechanism affects the search for the global optimum. Intuitively, selecting the best individuals from the last generation as parents is a considerable method, because this ensures the offspring gets closer to an optimum. In contrast, picking parents randomly can maintain a high level of diversity. To measure how these two mechanisms perform in this scenario, we have implemented two DE algorithms that are totally equivalent except for parent selection. EA1 uses random parent selection, whereas the second version (EA2) selects the best individual as the base vector.

	EA1	EA2	GA
population	100	100	100
generations	30	30	30
parents	rand(3)	best(1)+rand(2)	tournament(2)
crossover	differential	differential	weighted average
crossover rate	80%	80%	100%
mutations	-	-	norm-distribution
mutation rate	-	-	20%
survivor	fitness	fitness	fitness

Table 1: Parameter settings for EA1, EA2 and the baseline GA.

EAs often have many parameters, including numerical and symbolic algorithms. The parameter settings used in the following experiments are listed in table 1.

To implement the evolutionary algorithms, the DEAP [4] framework was used, which provides out-of-the-box EA building blocks.

To check whether algorithms outperformed each other, Student's t-tests were performed on the individual gain measured for each enemy in each algorithm.

3 RESULTS AND DISCUSSION

3.1 Algorithms compared by enemy

3.1.1 Fitness comparison. Figure 2 compares fitness evaluated on three enemies independently, indicating that EA2 outperformed EA1 in both best and average fitness.

Interestingly, the fitness of EA2 surpassed EA1 at the first several generations, especially for enemy 2. Both EAs generated identical strategies against enemy 2, but EA2 achieved the solution much faster, indicating EA2 has better convergence property. Additionally, we evaluated standard deviation errors across 10 runs on each generation and the result also illustrates that the width of standard deviation error bonds declines faster on EA2, also indicating the overall performance of EA2 exceeds EA1. Evidently, the selecting the best individual as one of three parents indeed has advantages on convergence property and solution space exploration as compared to random parents selection.

3.1.2 Comparison of individual gain. We also have taken further analysis on the result of combat by individual gain, which is defined as: $individual\_gain = player\_energy - enemy\_energy$. We took the best solution in each of the 10 independent runs and evaluated gain 5 times for each enemy. The results, as shown in 3, revealed that the stability of the agent produced by EA2 is much higher than EA1.

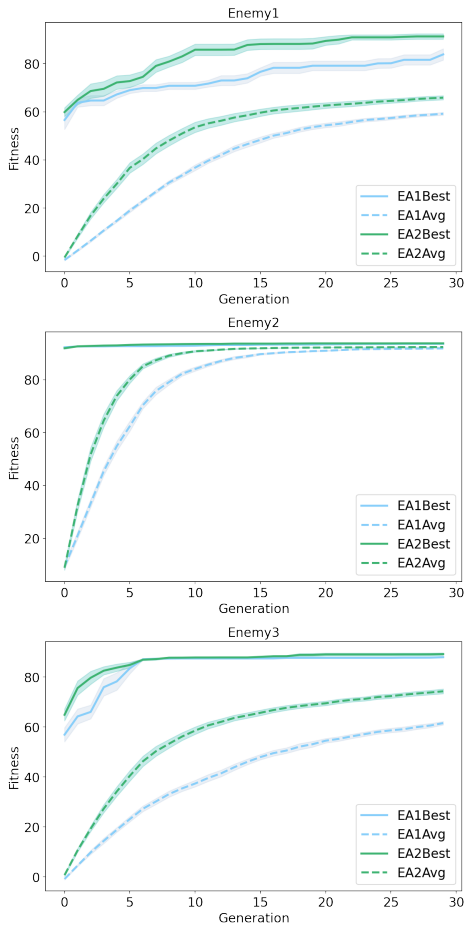

Figure 2: The average/standard deviation error for the mean and the maximum of fitness across generations shown per enemy.

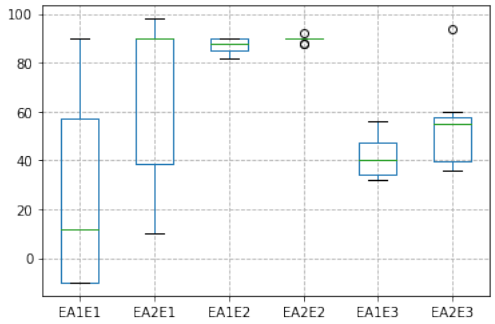

Figure 3: The average individual gain of EA1 and EA2 per enemy (denoted as E1, E2 and E3).

For both EAs, enemy 1 results in fluctuating performance, which implies this enemy is hard to defeat. Evidently, the complicated battlefield settings of enemy 1 correspond to a more complex search space, which means optimization is more difficult. In addition, a noticeable outlier of EA2 against enemy 3 indicates EA2 has potential to achieve better solutions.

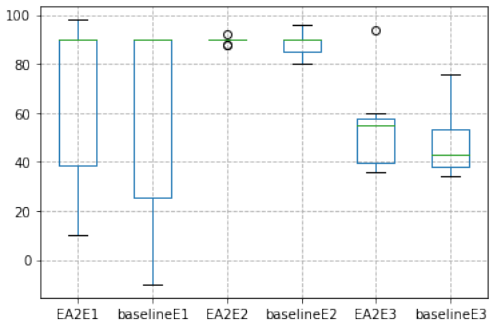

Figure 4: The average individual gain of EA2 as compared to the baseline GA per enemy (denoted as E1, E2 and E3).

In conclusion, EA2 utilizes information of the best individual in each generation to generate a potential evolution direction, leading to advantages in both convergence speed and the best solution.

3.2 Algorithms compared to baseline

In order to compare both algorithms to the baseline [3], the best solution of the baseline was tested 5 times for each of the 10 independent runs. The results are presented as individual gain in figure 4. This figure shows the medium gain in 10 runs for EA2 is higher than the baseline's in three enemies. Furthermore, table 2 indicates EA2 indeed performed significantly better than the baseline in enemy 2 and enemy 3, but not for enemy 1. This result also indicates that stochastic direct search is a practical approach in real-value optimization problems.

	GA	EA2	p-value
Enemy 1	60.2	68.4	0.15
Enemy 2	88.4	89.8	0.03
Enemy 3	46.4	53.4	0.02

Table 2: p-values associated with the difference in performance between EA2 and the baseline GA. Average individual gain for each algorithm is shown per enemy.

4 CONCLUSIONS

A series of experiments have demonstrated that EA2, implementing selection of the best individual as one of the three parents in parents selection, outperformed the initial DE with random parent selection. Meanwhile, EA2 also outperformed the baseline GA, as hypothesized due to the advantages of DEs over GAs. This makes EA2 the best performing algorithm out of the three algorithms examined in this paper.

REFERENCES

[1] T. J. Choi, J. Togelius, and Y. Cheong. 2020. Advanced Cauchy Mutation for Differential Evolution in Numerical Optimization. *IEEE Access* 8 (2020), 8720–8734.

[2] Karine da Silva Miras de Araújo and Fabrício Olivetti de França. 2016. An electronic-game framework for evaluating coevolutionary algorithms. *arXiv preprint arXiv:1604.00644* (2016).

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

[4] 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.

[5] Rainer Storn and Kenneth Price. 1997. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization* (1997), 341–359.