Evolutionary Computation

Y1481702

January 17, 2018

Contents

1 Introduction

1.1 Background

Biomimicry, the imitation of nature for solving human problems has produced many examples of world class design such as Velcro(R), self-cleaning paints and Shinkansen high-speed trains[1]. Since the term biomimicry was first coined in 1997, this design philosophy has started to inspire many new developments. As well as the aforementioned mimics of physical systems in nature, natural processes have also been imitated, such as in the circular economy in product design and swarm robotics in computer science.

Evolutionary Computation (EC) is a machine learning technique which is directly inspired by biological evolution. This is clearly another example of an imitation of nature's processes. Life has been around on earth for over 3.5 billion years, this is a lot of research & development time to produce useful solutions which can be related to many of the current problems in a wide range of fields. Through evolution, nature has already produced solutions to many of the hardest machine learning problems of today.
Machine learning usually focuses on optimisation of a particular goal...? link this into the following–¿

1.2 DEAP

Distributed **E**volutionary **A**lgorithms in **P**ython (DEAP) is an open source framework for Python[2]. The framework aims to provide tools for quickly producing custom evolutionary algorithms. It provides a large amount of in-built functionality for implementing commonly used evolutionary algorithms and allows for parts of these to be mixed and matched as well as intermixed with custom sections.

1.3 EC for Snake

The task given is to create an evolutionary algorithm in Python using the DEAP frameowrk in order to play the classic video game, Snake. This is not a new task and has been the subject of previous work including an ...[3] and more recently a ...[4]. Despite this there is still room for further work, due to ongoing research into evolutionary computing and significant challenges because of the nature of the game.

There are a variety of available instances of Snake with an array of different rules. In this case, there is a square game board of 14x14 cells as shown in Figure. 1. The snake will continously move forwards in the direction it is facing, which can be changed by the user at any time. The aim is to collect food that appears in a random cell on the board when the previous one is eaten. Each time the snake eats a piece of food, it grows and takes up an extra cell. Once the snake fills every cell it can no longer grow. Hence, the highest possible score (185) is equal to the number of cells (196) less the initial length of the snake (11).

Implementing this task will involve facing all of the usual challenges in evolutionary computation such as maintaining diversity and reducing bloat. Snake also brings along new challenges. The random placement of the food on the board adds an element of stochasticity to the solution meaning algorithm will perform differently across multiple runs and care will need to be taken to ensure that it works in the general case. The solution will also need to balance the need for the snake to avoid

crashing into walls or itself while still picking up the food before time runs out. Optimising both of these objectives at the same time might be difficult.
Starter sensing functions are given that tell the snake if there is a wall, itself or food directly ahead

2 Methods

2.1 Representation

2.1.1 Available Options

The representation links the original problem to the search space over which the algorithm runs. As such, the type of representation chosen can significantly affect the size of the search space and by extension, the time needed to search it.

Genetic Algorithms (GA)
GAs are perhaps the most commonly used form of evolutionary algorithm. They are generally used for optimisation problems but have been applied to a variant of the snake game before[4]. In this case, the GA was used to optimise four parameters (smoothness, space, food and direction) which determined the snake's movement.

Evolutionary Strategy (ES)

Genetic Programming (GP)
The aim of GP is to evolve programs that can be used to solve a problem. Clearly this is immediately more relevant for our snake implementation, the aim would be to create a program that can be run to determine our next move given the current state. GP uses a tree representation for programs, where the internal nodes are functions and the leaf nodes of the tree are constants and variables.
GP has previously been explored by Ehlis[3] and produced some promising results. Different function sets have been explored as well as a technique called priming which... [what is this?].

Neuroevolution
Neuroevolution is a hybrid technique between evolutionary computing and artificial neural networks (ANN). Unfortunately as the brief requires that the DEAP Python library is used, neuroevolution is not an available option as DEAP does not support it.
Grammatical Evolution? Is this explicitly supported in DEAP?[2] DEAP's transparency allows virtually any algorithm to be implemented but it's easier to use the built in structures.

2.1.2 Choice and Justification

Genetic Programming will be used as it most closely fits the aims set out in the introduction above.
The sensing functions can be used for the
Typing? Strong/Weak?

2.1.3 Physical Environments

This application of GP fits into a specific category involving a physical enviroment, which in this case is simulated. The added difficulty comes from the fact that the execution of any terminal

element may change the environment. Once this change has been made, the subsequent execution may produce a different effect in the environment.
In this kind of application the internal EA mechanics remains the same, but fitness evalutation can be significantly more computationally expensive.[5, p. 110]
Additional functions may help the snake to find a better solution, however we need to be careful of overparameterising the problem. This is a common trade-off in many forms of machine learning?

2.1.4 Initialisation Procedure

2.2 Population

2.2.1 Size

In almost all EA applications the population size is constant[5, p. 20] The size of the population can also affect the algorithm's performance. If the population size is too small it may not be exploring enough of the search space to contain any good solutions. It will then require a large amount of mutation to find any good solutions creating a very unstable solution. If the population size is too large it will take a lot of time to evaluate and select potential parents.

2.2.2 Diversity

Within this evolutionary computation task, each individual of the population corresponds to a point in the search space of all possible programs. Diversity corresponds to having these points reasonably spread out throughout this space. A diverse population is, by definition, exploring more of this search space than one that is not.

2.3 Evaluation

Implementing a good fitness evaluation function is essential to evolving a solution to this problem. Without this, there will be no way to assess individuals in order to determine which will survive and reproduce. While there are two major objectives at play in the game, avoiding crashing the snake and getting to the food before the timer hits zero, these can be represented by the final size of the snake when the game ends. As such, we can simply consider this a single-objective problem where the objective is produce the maximum length of snake before either the snake crashes or the timer hits zero. This significantly simplifies the problem with minimal impact.

The snake's food spawns randomly around the board. This element of stochasticity makes evaluating the fitness of a particular solution more difficult. The fitness of a given algorithm will likely change between games. In order to ensure a fairer representation of fitness, each solution will need to be evaluated across multiple games.

2.4 Parent Selection

Selecting which individuals should have influence over the next generation is one of the most important factors for success in Evolutionary Computing. If only the best individuals are selected then diversity will be lost, the algorithm will get stuck in a local minima and be unable to find a good general solution. This is known as premature convergence. If too many of the less-promising solutions are selected then the algorithm may be unstable meaning good solutions will take too long to find and may be lost quickly. Striking a good balance between a diverse population and a stable algorithm is necessary for success in this task.

In biological evolution, old generations die to free up resources for newer generations. Disregarding solutions due to age ...
Parent Selection can be random, so (if this is the case) good solutions may well die out. Elitism can be used to ensure that the best solution isn't randomly removed from the population. If no better solution is ever found, this solution will want to be present in the final population once evolution is complete.

2.4.1 Maintaining Diversity

Diversity is difficult to quantity in an objective manner and, as such, no single measure for it exists. A range of different measures including the number of different fitness values, number of different phenotypes/genotypes or entropy may be used[5].

Roulette Wheel

Tournament

2.4.2 Bloat

Bloat, the increase of the size of a genetic program without significant increase in fitness, is a difficult problem to overcome. Bloat is linked to, but different from, overfitting, a common problem in many forms of machine learning[6].
Controlling bloat while maximising fitness turns the evolution into a multi-objective optimisation problem. The two most commonly used solutions to bloat are parsimony pressure and double tournaments.
Parsimony pressure subtracts a value based on the size of the program tree from the individual's fitness, this value does not necessarily need to be linear. [7]
Double tournaments can be used for parent selection as an extension of regular tournament selection described in Section 2.4.1. It has been shown that the order of a double tournament has no significant bearing on the end result. [cite, practical?]
[8]

2.5 Variation Operators

2.5.1 Mutation

Mutation is the operator which incorporates new ideas into the population. Without it, genes would only be shuffled around and no new solutions could be found. A trade-off is required here, too much mutation will result in a highly unstable algorithm. Far too much mutation would Care needs to be taken when implementing mutation so as not to create biases. This is particularly the case with real-valued mutation..

2.5.2 Recombination

2.5.3 Termination Condition

3 Results

3.1 Calibration

4 Conclusion

4.1 Main Findings

4.2 Further Work

The complexity and diversity of the snake game can lead into plenty of areas for further exploration. Recent developments in evolutionary computation such as co-evolution and neuroevolution could be explored in the context of this task. With co-evolution individuals would have to compete in order to get to the food items first, creating snake players that find the food faster.

Future work could also explore the adaptability of this program to perform well across the wide number of variants of the game that exist with different formats and rules. This could be further expanded to attempt to create individuals that generalise across multiple game variants and carry their learning across in the way a human player would.

5 References

[1] J. M. Benyus, *Biomimicry: Innovation Inspired by Nature.* HarperCollins, May. 1997.

[2] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, Jul. 2012.

[3] T. Ehlis. (2000, Aug) Application of genetic programming to the snake game. [Online]. Available: https://www.gamedev.net/articles/programming/artificial-intelligence/application-of-genetic-programming-to-the-snake-r1175/ [Accessed: Nov. 30, 2017]

[4] J.-F. Yeh, P.-H. Sun, S.-H. Huang, and T.-C. Chiang, "Snake game ai: Movement rating based functions and evolutionary algorithm-based optimization," in *2016 Conference on Technologies and Applications of Artificial Intelligence, Proceedings*, Nov. 2016.

[5] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing.* Springer, 2007.

[6] L. Vanneschi, M. Castelli, and S. Silva, "Measuring bloat, overfitting and functional complexity in genetic programming," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '10. New York, NY, USA: ACM, 2010, pp. 877–884. [Online]. Available: http://doi.acm.org/10.1145/1830483.1830643

[7] S. Luke and L. Panait, "Fighting bloat with nonparametric parsimony pressure," *Parallel Problem Solving from Nature*, pp. 411–421, 2002.

[8] E. D. de Jong, R. A. Watson, and J. B. Pollack, "Reducing bloat and promoting diversity using multi-objective methods," in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO'01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 11–18. [Online]. Available: http://dl.acm.org/citation.cfm?id=2955239.2955241

6 Appendix

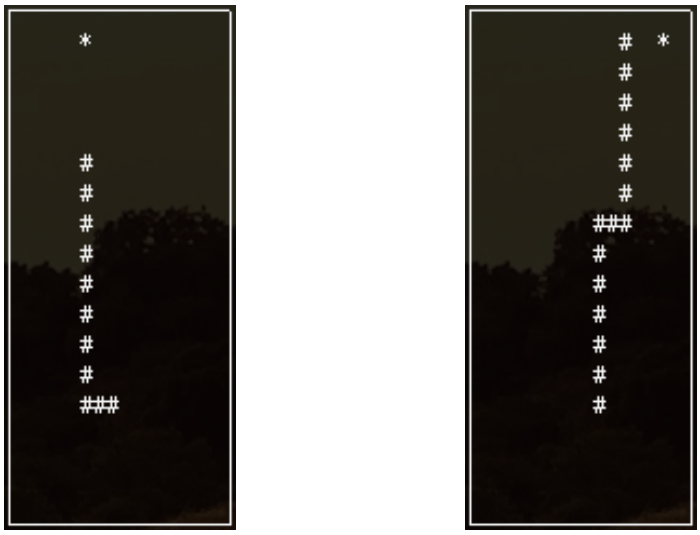

Figure 1: The 14x14 Game Board