Genetic Algorithm Documentation

This project is a package that implements a novel genetic algorithm. It finds a set of 16 numbers between 0 and 1 that maximize the output of an arbitrary fitness function that takes 16 numbers between 0 and 1 as input. It does this by making successive “generations” of “individuals” who “reproduce” in a way that ensures the individuals in each generation are “descended” from individuals in the previous generation with the highest output from the fitness function.

Components

`individual.py`: Defines the Individual class, which represents an “individual”, a set of 16 numbers between 0 and 1 with methods to assess its fitness with a fitness function and combine with another Individual instance to return a “child” with a combination of their genes.

`generation.py`: Defines the Generation class, which represents a “generation” of individuals. Each Generation instance has a list of Individual instances, and methods to assess those individuals and combine them into a new Generation with children descended from the fittest individuals.

`population.py`: Defines the Population class, which represents a “population” consisting of successive generations. Each Population instance has a list of Generation instances, and a `main()` method that adds new Generation instances descended from the existing ones until an Individual with a fitness exceeding a target value is found. Apparently, in scholarly writing on the topic, the term ‘population’ is used to refer to the members of a generation, and not a collection of successive generations. We didn’t know this when we named the class and are clarifying that they refer to different things to avoid confusion.

`test_project.py`: Contains unit tests for the project.

`demonstration.py`: Uses the package to find coefficients for a polynomial of degree 15 to approximate a polynomial with random coefficients, and will show an animation of a histogram of fitnesses for each generation after it’s done. Because our project is a package and not a standalone script, this file is included as an example of an application of the package to show it in action. The code within this file is not technically part of the project, but it’s included in the repository so you can see it in action without making your own application.

How to run from command line

This project is a package, and therefore not really suited to being run directly. However, to see an example of the program in action, the following will work:

```
    python demonstration.py
```

How to use

This project is meant to be imported and used to approximate solutions to problems it's suited for. To import the package using pip, use the following command in the terminal:

```
pip install -i https://test.pypi.org/simple/ genetic-algo
```

Within your own code, import `Population` from `genetic_algo` and define a Population instance. The Population constructor takes a target value and a fitness function as arguments. The target value is the fitness value that has to be surpassed by an individual to stop `main()` and have it return the fittest Individual. The fitness function is a function that takes a list of 16 floats as an argument and returns a value where higher output is better.

The `main()` method of the population object will return the first Individual to have a fitness higher than the target value supplied in the Population instance's constructor. The `genes` attribute of this Individual is the generated array.

While the `main()` method is running, it'll print information about the current generation to the console. This printout is in the following format:

`Gen {`# of the generation`} ({`# of individuals`} individuals): {`best fitness born this generation`} [{`best fitness all time`}] (avg:{`average fitness of this generation`})`

This allows you to follow how the model is doing while it's running, and get a sense for whether it's working or not.

Works Cited

"Packaging Python Projects." *Packaging Python Projects - Python Packaging User Guide*, Python Packaging Authority, packaging.python.org/tutorials/packaging-projects/.

Tutorial used to upload the project as a pip package.

SethBling, director. *MarI/O - Machine Learning for Video Games*. *SethBling*, YouTube, 13 June 2015, youtu.be/qv6UVOQ0F44.

This video provided the idea for the project, and provided the foundation for most elements of the implementation. The implementation shown is much more complicated, but we used many of the ideas discussed in a more rudimentary implementation.

Stanley, Kenneth O., and Risto Miikkulainen. "Evolving Neural Networks through Augmenting Topologies." *Evolutionary Computation*, vol. 10, no. 2, 2002, pp. 99–127., doi:10.1162/106365602320169811.

Great paper that was the foundation for the MarI/O YouTube video. Was re-read for a refresher on the topic.

Vose, Michael D. *The Simple Genetic Algorithm: Foundations and Theory*. MIT Press, 1999.

Interesting read that provided new ways of looking at how the simple genetic algorithm works.