Neural Network and Genetic Algorithm

In this project the goal is to simulate three mathematical functions ([Sine function](https://en.wikipedia.org/wiki/Sine_wave), [Rosenbrock function](https://en.wikipedia.org/wiki/Rosenbrock_function) and [Rastrigin function](https://en.wikipedia.org/wiki/Rastrigin_function#:~:text=In%20mathematical%20optimization%2C%20the%20Rastrigin,has%20been%20generalized%20by%20Rudolph.)) with artificial neural networks, to demonstrate their learning capabilities and the power of the Genetic Algorithm.

This project uses a [TLU Perceptron](https://medium.com/@isa.inuwa/introduction-to-tlu-and-perceptron-e0d9c351e6a6) as a neuron, and all the corresponding layouts associated with the TLU Perceptorn.

The basic process is:

- generation of a set number of neural networks.

- train those neural networks with set examples.

- pick two best neural networks to serve as parents for the next generation.

- create a new population with parents and a potential mutation factor, and also include a set number of best networks from the previous generation.

We can repeat these steps a set number of times, while ensuring that we are always getting the same or better results in simulating the functions. That is due to the fact that we can bring a set number of best neural nets from the previous generation into a new generation. We will call those neural nets: “Elitists”.

The user has the ability to chose between three neural net layouts:

-5s (5 neurons in a hidden layer)

-20s (20 neurons in a hidden layer)

-5s5s (two hidden layers, both with 5 neurons)

Lets look at how to call the main function with all the parameters.

Here is an example:

>>> python main.py --train sine\_train.txt --test sine\_test.txt –-nn 5s --popsize 10 --elitism 1 --p 0.1 --K 0.1 --iter 10000

The parameters:

--train (the training file that contains some inputs and outputs for a specific function we want to emulate.)

--test (the testing file that contains some inputs and outputs for a specific function we want to emulate.)

--nn (specifies what neural network layout we will be using)

--popsize (the size of a population for every iteration)

--elitism (the number of “elitists” we want to transfer to the next generation)

--K (scale of mutation, standard deviation by Gauss)

--iter (number of iterations)

--p (chance of mutation for every weight in a neural network)

Output:

On every 10 iterations/generations of neural nets:

[Train error @<number of iteration>]: <error on a current iteration from the best neural net>

In the end:

[Test error]: <the error on the best neural net in the last population>

\*Error – The error is the difference between the output and the expected output.

For example: If we are simulating a sinus function and we give a neural net 3.14

And it outputs 0.34, the error is 0.34.

The expected output was 0,so

abs|0-0.34|=0.34