The Computer Programmer: Using Genetic Algorithms to Write Code

Aaron Kaufer, Daniel Tan

Motivating Question

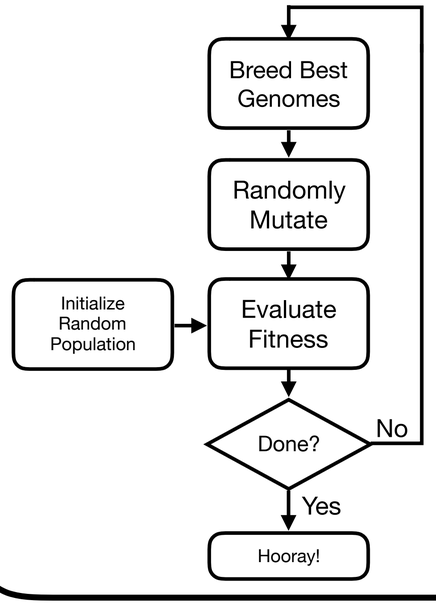
Is it possible for a computer to learn how to code?

Task Definition

- Write an algorithm which takes in a programming language and a description of a program and outputs the source code for that program.
- Two types of programs:
  - Procedural Programs: accomplish a task:
    - `print("hello world")`
  - Functional Programs: model a function
    - `def f(x): return sin(x)+1`

Approach: Genetic Algorithm

- Key Idea: Treat source code as "genomes" and evolve programs according to principles of natural selection.



1) Initialize population of random genomes/ source codes.
2) Assign each genome a fitness value that measures how well it performs.
3) If some genome is perfect, we're done. Otherwise, move to next generation:
4) Breed the best genomes (mix their source codes together).
5) Randomly mutate every genome.
6) Return to step 2.

Procedural Programs

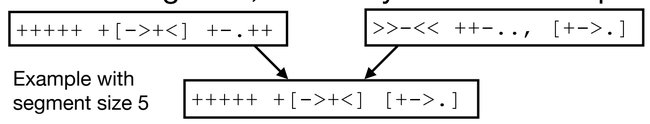
Language: Brainfuck

- Esoteric language consisting of only: `+-<>,.[]`
- Turing complete + small alphabet = natural choice for genetic algorithm.
- Hello World:

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++.>+.++++++
+..+++.>++.<<+++++++++++++++.>.+++.------.--------.>+.>.
```

Breeding Algorithm: Segment Breeding

- Divide each parent's genes into segments.
- For each segment, randomly decide which parent.



Fitness Function:

- Compare output to desired output character by character.

```
for i in 0 to len(output)-1:
  fitness += 256 - abs(output[i] - target[i])
```

Results

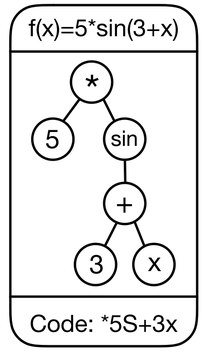
String	Time (s)	Gen
"hello"	5.95	1,616
"hello world"	54.29	12,983
"Hello World!"	146.31	31,822

- Can gets short (< 5 chars) string in under a minute.
- Struggles with longer and more diverse strings.
- Completely unable to get basic input-output programs like addition
- Main challenge: BF programs change too discretely, genetic algorithms work best with somewhat continuous structure

Functional Programs

Language: TreeFunction

- Motivating Idea: Represent function as a tree, encode it as a pre-order traversal of the tree.



Breeding Algorithms:

1. Segment Breeding on the code, using segment size 1. (Often warps tree structure)
2. Tree Breeding: Randomly swap entire subtrees of parents (Preserves tree structure)

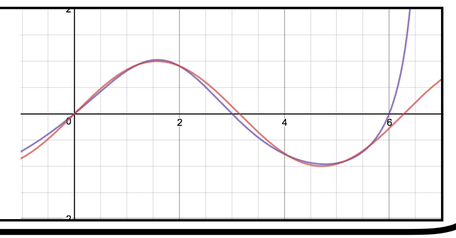
Fitness Function

1. Evaluate the function at a finite fixed set of inputs.
2. Compute relative error of each output value (compared to correct/desired output value).
3. Give high fitness to genomes with low total error.

Results

- Consistently gets simple functions (e.g. 5*sin(x+3), x^3+5) in under 10 seconds.
- Good at low-degree polynomial interpolation.
- Can guess constants (for example, give it f(x)=1.64492 and it will guess $f(x)=\pi^2/6$)
- Can do constrained optimization well (e.g. approximate sin(x) using only rational functions):

Red: sin(x)

Purple:

$$\frac{-5x^3+45x^2-90x}{2x^3-21x^2+66x-108}$$