Genetic Algorithms & Evolutionary Computation

Autonomous Systems

Dr. Matthia Sabatelli

September 29th, 2023

Today's Agenda

1. Motivation & General Principles
2. Genetic Algorithms
3. Genetic Programming
4. Practical Applications

Motivation & General Principles

Some History about Genetic Algorithms

- Genetic Algorithms (GAs) started to be developed in the early 1960s by **John Holland**, who is considered to be the father of Algorithmic Evolution.
- However, only in the early mid-1980s GAs started to find successful applications
- In the 1990s the ideas of GAs evolved and resulted in Genetic Programming
- While still powerful for certain classes of problems, GAs are not very popular nowadays

Motivation & General Principles

Genetic Algorithms provide an approach to **learning** that is based on **simulated evolution**

- They fall within the category of search techniques
- More specifically they are categorized as Global Search Heuristics, meaning they do not search for general $\rightarrow$ specific hypothesis
- Instead they repeatedly mutate and recombine parts of the best currently known solutions

Motivation & General Principles

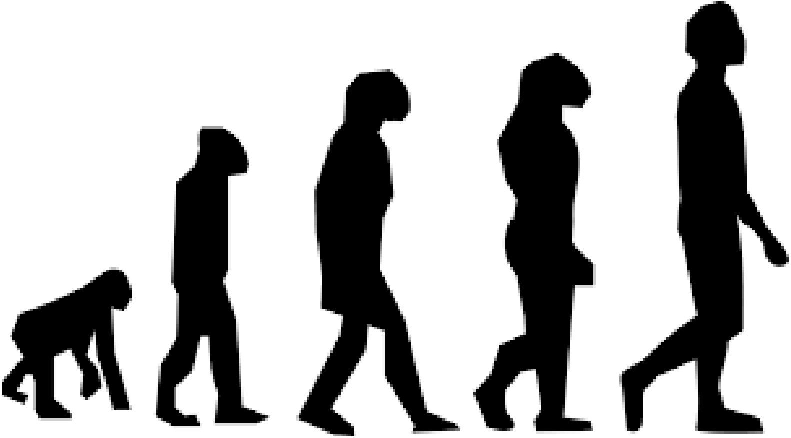
How do GAs **intuitively work**?

$\Rightarrow$ We typically start with a problem we wish to solve and make a guess of potential solutions that can solve it

1. These solutions get evaluated and ranked based on how well they perform on our initial problem
2. Once evaluated, the solutions can interact across each other and produce a new set of solutions
3. This process repeats itself until we find a solution that satisfies our initial problem or a stopping criteria is met

Motivation & General Principles

The aforementioned steps mimic biological evolution



Motivation & General Principles

The **popularity** of GAs in the 80s and 90s was motivated by the following factors

1. Evolution is known to be successful in nature, why not use its ideas within AI as well?
2. Back then they could search very complex hypothesis spaces
3. They can be easily parallelized, therefore they can take advantage of powerful computer hardware

Genetic Algorithms

The **components** of a Genetic Algorithm are:

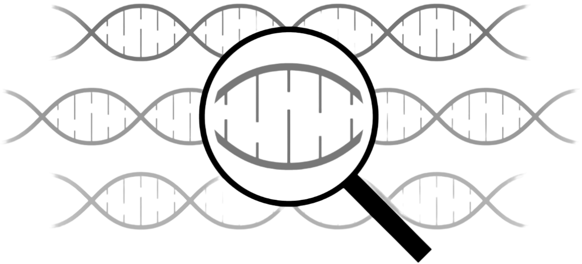
i) A learning problem we wish to solve

which allows us to define ii) a fitness function $f(x)$

Genetic Algorithms

The **components** of a Genetic Algorithm are:

iii) An initial pool of solutions e.g. programs that can play chess



Genetic Algorithms

The **components** of a Genetic Algorithm are:

iv) An evolution strategy

That tells us how we should modify our initial pool of solutions based on how bad/well they perform on our aforementioned evaluation function $f(x)$

Genetic Algorithms

A **Practical Example**: the MAXONE Problem

$\Rightarrow$ We have a string of 10 binary digits e.g. 1011010110 and want to **maximize** the number of ones in a string. Our objective, therefore, becomes writing a program that automatically finds:

1111111111

Genetic Algorithms

A **Practical Example**: the MAXONE Problem

Let's start with a **randomly generated pool** of solutions s which we call G_1 where G stays for Generation:

$$G_1 = \begin{cases} s_0 = 0101100001 \\ s_1 = 1100010001 \\ s_2 = 1111110000 \\ s_3 = 1010101111 \\ s_4 = 0000000010 \end{cases}$$

Genetic Algorithms

A **Practical Example**: the MAXONE Problem

Let's now **evaluate** based on $f(x)$[1] how good/bad these solutions are

$$G_1 = \begin{cases} s_0 = 0101100001 \\ s_1 = 1100010001 \\ s_2 = 1111110000 \\ s_3 = 1010101111 \\ s_4 = 0000000010 \end{cases}$$

[1] We simply count how many ones appear in the string

Genetic Algorithms

A **Practical Example**: the MAXONE Problem

Let's now **evaluate** based on $f(x)$[2] how good/bad these solutions are

$$G_1 = \begin{cases} s_0 = 0010100001 \rightarrow \mathbf{3} \\ s_1 = 1100010001 \rightarrow \mathbf{4} \\ s_2 = 1111110000 \rightarrow \mathbf{6} \\ s_3 = 1110101111 \rightarrow \mathbf{8} \\ s_4 = 0000000010 \rightarrow \mathbf{1} \end{cases}$$

[2] We simply count how many ones appear in the string

Genetic Algorithms

A **Practical Example**: the MAXONE Problem

Let's now create a **new generation** G_2 where we start by only keeping the fittest solutions

$$G_2 = \begin{cases} s_0 = 1110101111 \rightarrow \mathbf{8} \\ s_1 = 1111110000 \rightarrow \mathbf{6} \\ s_2 = 1100010001 \rightarrow \mathbf{4} \\ s_3 = \cancel{0010100001} \rightarrow \mathbf{3} \\ s_4 = \cancel{0000000010} \rightarrow \mathbf{1} \end{cases}$$

Genetic Algorithms

A **Practical Example**: the MAXONE Problem

What to do with the spots which were taken by the worst solutions?

$$G_2 = \begin{cases} s_0 = 110101111 \rightarrow \mathbf{8} \\ s_1 = 1111110000 \rightarrow \mathbf{6} \\ s_2 = 1100010001 \rightarrow \mathbf{4} \\ s_3 = ?????????? \rightarrow \mathbf{?} \\ s_4 = ?????????? \rightarrow \mathbf{?} \end{cases}$$

Genetic Algorithms

A **Practical Example**: the MAXONE Problem

What to do with the spots which were taken by the worst solutions?

$\Rightarrow$ We can create two new solutions which take advantage of the two fittest solutions in G_1. We call them c_1 and c_2 were c stays for child

$$c_1 = 1110101111 + 1111110000 = 11101\ 10000$$
$$c_2 = 1110101111 + 1111110000 = 01111\ 111111$$

Genetic Algorithms

A **Practical Example**: the MAXONE Problem

What to do with the spots which were taken by the worst solutions?

⇒ We can create two new solutions which take advantage of the two fittest solutions in G_1. We call them c_1 and c_2 were c stays for child

$$c_1 = 11101\cancel{01111} + \cancel{11111}10000 = 11101\ 10000$$
$$c_2 = \cancel{11101}01111 + 11111\cancel{10000} = 01111\ 111111$$

Genetic Algorithms

A **Practical Example**: the MAXONE Problem

Let's evaluate the two new solutions ...

$$G_2 = \begin{cases} s_0 = 1110101111 \rightarrow \mathbf{8} \\ s_1 = 1111110000 \rightarrow \mathbf{6} \\ s_2 = 1100010001 \rightarrow \mathbf{4} \\ s_3 = 1110110000 \rightarrow \mathbf{?} \\ s_4 = 0111111111 \rightarrow \mathbf{?} \end{cases}$$

Genetic Algorithms

A **Practical Example**: the MAXONE Problem

We have found a new best solution!

$$G_2 = \begin{cases} s_0 = 1110101111 \rightarrow \mathbf{8} \\ s_1 = 1111110000 \rightarrow \mathbf{6} \\ s_2 = 1100010001 \rightarrow \mathbf{4} \\ s_3 = 1110110000 \rightarrow \mathbf{5} \\ s_4 = 0111111111 \rightarrow \mathbf{9} \end{cases}$$

Genetic Algorithms

A **Practical Example**: the MAXONE Problem

We have found a new best solution!

$$G_2 = \begin{cases} s_0 = 1110101111 \rightarrow \mathbf{8} \\ s_1 = 1111110000 \rightarrow \mathbf{6} \\ s_2 = 1100010001 \rightarrow \mathbf{4} \\ s_3 = 1110110000 \rightarrow \mathbf{5} \\ s_4 = \boxed{0111111111 \rightarrow \mathbf{9}} \end{cases}$$

Genetic Algorithms

$\Rightarrow$ If we keep performing this iterative process, therefore creating many generations G, we will eventually find a solution that satisfies the MAXONE problem.

Genetic Algorithms

$\Rightarrow$ If we keep performing this iterative process, therefore creating many generations G, we will eventually find a solution that satisfies the MAXONE problem.

Elitism

We ensure that the solution quality obtained by the GA will not decrease from one generation to the next, and that the best solutions of a generation will carry over to the next.

Genetic Algorithms

For many real world problems this can take prohibitively long ☹

Genetic Algorithms

In our MAXONE problem, we have simply removed the solutions within G_1 by ranking all the different solutions and discarding the last two ones. This was rather a simple heuristic.

$\Rightarrow$ Typically, in Genetic Algorithms a probabilistic approach decides whether to keep/discard a solution.

$$\Pr(s_i) = \frac{f(s_i)}{\sum_{j=1}^{p} f(s_j)}$$

Genetic Algorithms

$\Rightarrow$ Next to the process that incrementally comes up with better and better *solutions* and that, therefore, resembles natural selection, there's one more important component in Genetic Algorithms that mimics biological evolution

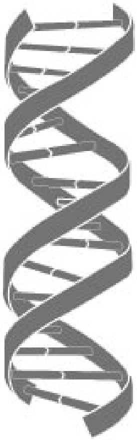
Genetic Algorithms

$\Rightarrow$ The operations that recombine and mutate selected members of one specific generation G. We call these operations **Genetic Operators**

There are two operators which are the most common ones:

- *Crossover*
- *Mutation*

Genetic Algorithms



Genetic Algorithms

$\Rightarrow$ Every *Crossover* operator produces two new children c_1 and c_2 from two strings by copying selected bits from each parent p

Single-Point Crossover

$$p_1 = 11101001000$$
$$p_2 = 00001010101$$

Genetic Algorithms

$\Rightarrow$ Every *Crossover* operator produces two new children c_1 and c_2 from two strings by copying selected bits from each parent p

Single-Point Crossover

$$p_1 = 11101001000$$
$$p_2 = 00001010101$$

Genetic Algorithms

⇒ Every *Crossover* operator produces two new children c_1 and c_2 from two strings by copying selected bits from each parent p

Single-Point Crossover

$$p_1 = 11101001000$$
$$p_2 = 00001010101$$

$$c_1 = 11101010101$$

Genetic Algorithms

$\Rightarrow$ Every *Crossover* operator produces two new children c_1 and c_2 from two strings by copying selected bits from each parent p

Single-Point Crossover

$$p_1 = 11101001000$$
$$p_2 = 00001010101$$

$$c_2 = 00001001000$$

Two-Point Crossover

$$p_1 = 11101001000$$
$$p_2 = 00001010101$$

Genetic Algorithms

Two-Point Crossover

$$p_1 = 1110100 01000$$

$$p_2 = 00001010101$$

Genetic Algorithms

Two-Point Crossover

$$p_1 = 11101001000$$
$$p_2 = 00001010101$$

$$c_1 = 00101000101$$

Genetic Algorithms

Two-Point Crossover

$$p_1 = 11101001000$$
$$p_2 = 00001010101$$

$$c_2 = 11001011000$$

Genetic Algorithms

As you can imagine there are plenty of crossover combinations one can choose from. One could even design its own crossover operator but the *single-point* and *double-point* operators are typically good enough.

A special case of crossover which only requires a single parent is:

Point Mutation

$$p_1 = 11101001000$$

Genetic Algorithms

As you can imagine there are plenty of crossover combinations one can choose from. One could even design its own crossover operator but the *single-point* and *double-point* operators are typically good enough.

A special case of crossover which only requires a single parent is:

Point Mutation

$$p_1 = 11101\mathbf{0}1000$$

Genetic Algorithms

As you can imagine there are plenty of crossover combinations one can choose from. One could even design its own crossover operator but the *single-point* and *double-point* operators are typically good enough.

A special case of crossover which only requires a single parent is:

Point Mutation

$$p_1 = 11101\mathbf{1}1000$$

Genetic Algorithms

As you can imagine there are plenty of crossover combinations one can choose from. One could even design its own crossover operator but the *single-point* and *double-point* operators are typically good enough.

A special case of crossover which only requires a single parent is:

Point Mutation

$$\mathbf{c}_1 = 1110111000$$

Genetic Programming

Genetic Programming is a form of evolutionary computation in which the individuals in the evolving population are full computer programs rather than binary strings!

- It is an extension of Genetic Algorithms
- Has demonstrated to produce intriguing results:
    1. Design of electronic filter circuits
    2. Classification of segments of protein molecules

$\Rightarrow$ How does it work?

Genetic Programming

We need to find a way of representing a computer program.
This is done via a **tree**:

- Each function call is represented by a node in the tree
- The arguments of the function are given by the descendant nodes

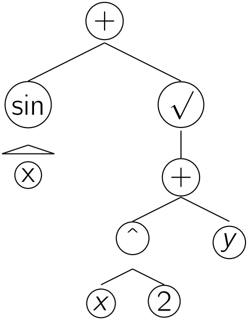
We would like to find a tree representation for the function

$$\boxed{\sin(\mathbf{x}) + \sqrt{\mathbf{x}^2 + \mathbf{y}}}$$

Genetic Programming

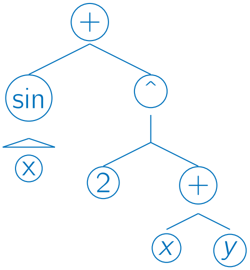
⇒ Which should look like:

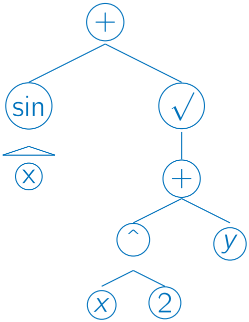
$$\boxed{\sin(\mathbf{x}) + \sqrt{\mathbf{x}^2 + \mathbf{y}}}$$



Genetic Programming

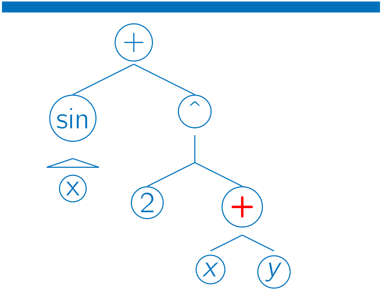
Let's consider the following two trees as candidate solutions:

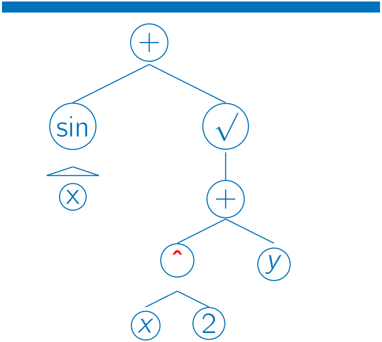




Genetic Programming

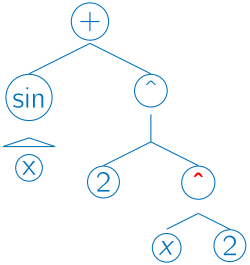
Let's consider the following two trees as candidate solutions:

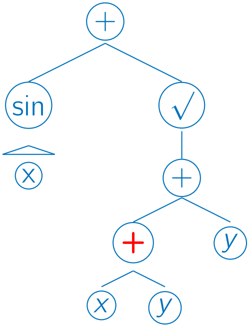




Genetic Programming

⇒ After performing a *crossover* operation they will result in the following two new solutions:



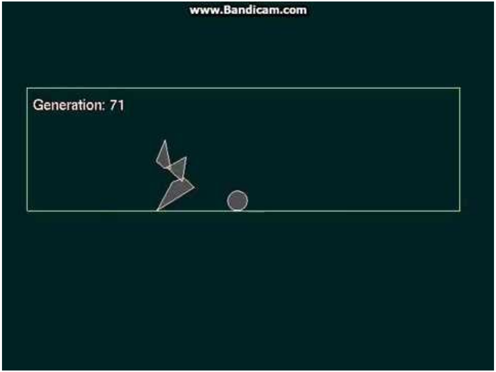


Genetic Programming

In Genetic Programming:

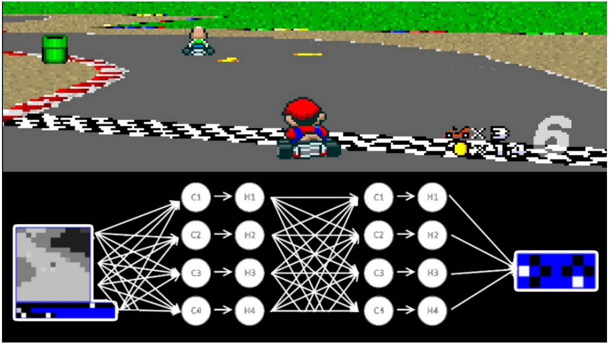
1. We use the same genetic operators that define Genetic Algorithms
2. However, we need to define the set of primitive functions e.g. $+$, $\sqrt{}$, sin... which is not trivial
3. A solution is represented by an entire program tree, which can make their evaluation expensive

Practical Applications



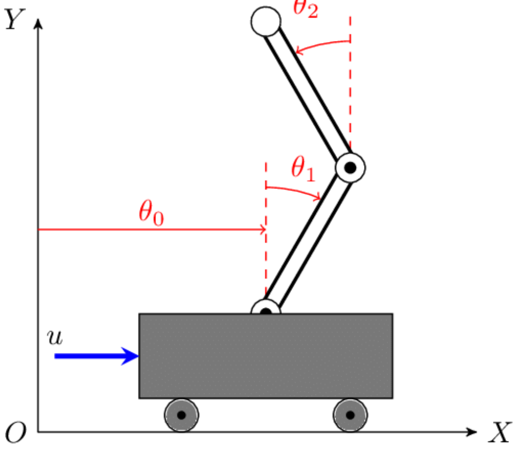
https://www.youtube.com/watch?v=Gl3EjiVlz_4

Practical Applications



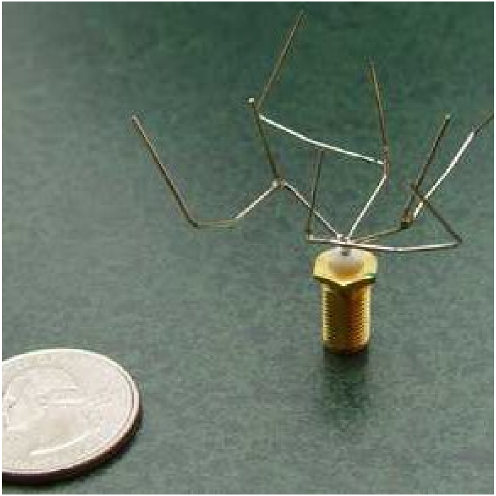
https://www.youtube.com/watch?v=qv6UVOQ0F44

Practical Applications



https://www.youtube.com/watch?v=czhhzKrBJfM

Practical Applications



Pros & Cons of Genetic Algorithms

Among the main benefits we have that:

- The underlying concept is easy to understand ✓
- Can be used for multi-objective optimization ✓
- Support distributed learning ✓
- Same cooking recipe can be used across large variety of tasks ✓
- Work well when the problem is not differentiable ✓

However there are some significant limitations as well ...

Pros & Cons of Genetic Algorithms

⇒ Among such limitations we have:

- No convergence guarantees ✗
- Computing the evaluation function $f(x)$ can be expensive ✗
- Lots of implementation parameters need to be defined ✗
- Termination Criteria? ✗

Final References

- Mitchell, Tom M., and Tom M. Mitchell. Machine learning. Vol. 1. No. 9. New York: McGraw-hill, 1997. Chapter 9.
- Holland, John H. "Genetic algorithms." Scientific american 267.1 (1992): 66-73.
- Koza, John R., and Riccardo Poli. "Genetic programming." Search methodologies. Springer, Boston, MA, 2005. 127-164.
- B, Thomas, and Hans-Paul Schwefel. "An overview of evolutionary algorithms for parameter optimization." Evolutionary computation 1.1 (1993): 1-23.

The GECCO Conference