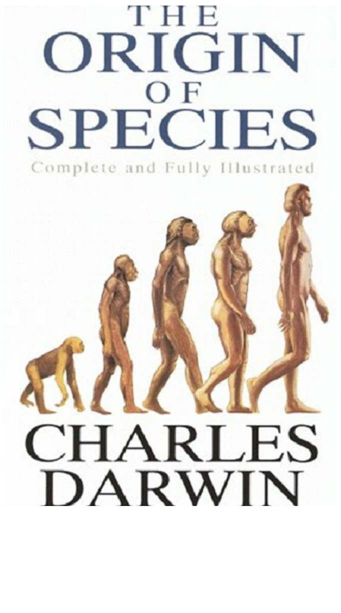
Introduction to A.I.

Genetic Algorithm

The Virtual Selection



What is Darwin’s Theory of Evolution ?

The theory of evolution by natural selection, first formulated in Charles Darwin's book "On the Origin of Species" in 1859, describes how organisms evolve over generations through the inheritance of physical or behavioral traits.

The main principles of Darwin's Theory necessary for evolution to happen are :

- Heredity - There must be a process in place by which children receive the property of their parent
- Variation - There must be a variety of traits present in population or a means with which to introduce a variation
- Selection - There must be a mechanism by which some members of the population can be parents and pass down their genetic information and some do not (survival for the fittest)

Understanding Genetic Algorithm

It is an algorithm that is inspired by **Darwin’s theory of Natural selection** to **solve optimization problems**.

Feel free to look at the Wikipedia page to make sure you understand the concept.

There are 5 phases in a genetic algorithm:

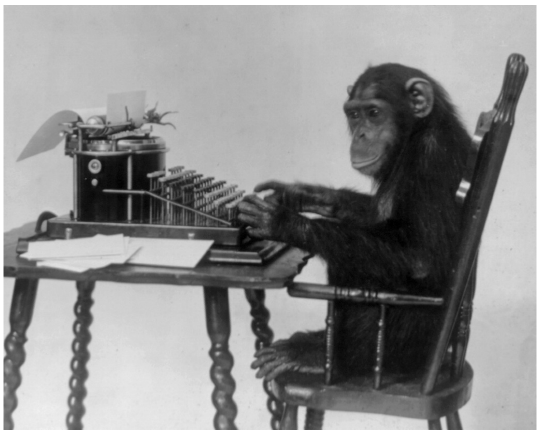
1. Creating an initial population
2. Defining a fitness function
3. Selecting the parents
4. Making a crossover
5. Mutation with a low chance

Let’s write our own Genetic Algorithm !

In this repository, you can find a **geneticAlgorithm.py** python file. We’ll need it later !

DO NOT LOOK AT THE SOLUTION FILE

Before coding, let me explain the goal of our next algorithm with the infinite Monkey Theorem !



The infinite monkey theorem states that if a monkey starts hitting keys at random on a keyboard for an infinite amount of time, he will almost surely type a given text, such as the complete works of William Shakespeare.
In fact, the monkey would almost surely type every possible finite text an infinite number of times. However, the probability of this event is so tiny that it will require more time than the estimated age of the universe, but chances of occurrence of this event is not zero.

Suppose the typewriter has 50 keys, and the word to be typed is “**monkey**”. If the keys are pressed randomly and independently, we can calc the chance of the first 6 letters spelling “**monkey**” :

(1/50) ^ 6 = 1 / 15 625 000 000

less than one in 15 billion, **BUT STILL NO ZERO CHANCE**

But, you can tell me, if I want to type the same, it will take me less than 6 seconds to do it. And you wouldn’t be wrong ! But why ? Because you know the word “**monkey**” and its spelling !

So, can we use Evolution Theory and improve my program significantly ? The answer is YES ! And this is thanks to the concept of Genetic algorithms.

1 - Creating an initial population

Let’s start at the beginning, i.e. with the generation of the initial population !

We want to generate a list of size **arrLength** of random strings with a length of **len**.

Complete the following functions:

```
def createStringWithRandomChars(len: int) -> str:

def initPopulation(arrLength: int) -> VectorStr:
```

Example of result of the **initPopulation(6)** function:

```
['sbzrtf', 'mgbral', 'hutmpw', 'anoyjj', 'ajndmu', 'zfnbxx']
```

2 - Defining a fitness function

The fitness function determines how likely an individual is fit to be selected for reproduction, and this based on its fitness score.
Let’s suppose our fitness function will assign a fitness score or a probability percentage to each element from the population, for each character matching our target word **monkey**.

Example:

- “bonkye” -> 5 (5 characters out of letters from “monkey” are present in this word)
- “abcdef” -> 1 (1 character similar)
- “mnopqr” -> 3
- etc.

You can also add score if the letter is in a good place.

Complete the following function:

```
def fitnessFunction(str: str) -> int:
```

After that, you need to create a dictionary that links a word with his own fitness score.

Complete the following function:

```
def createDictionnaryWithFitness(arrayStr: VectorStr) -> dict:
```

You should now have a generated population with for each word its own fitness score.

Example of result:

```
{'zabjzl': 0, 'zefolj': 2, 'myjdxj': 2, 'nixcrl': 1, 'sqqlya': 1, 'pkqaqr': 1}
```

3 - Selecting the parents

The idea behind this step is to select the fittest individuals and let them pass their genes to the next generation. Two elements of the population are selected based on their fitness scores. In our case, we select individuals with high fitness scores.

In our case, we selected these elements as these words have a high fitness score from the given population.

Per example, the highest of a population are:

“bonkye” -> 5
“momyek” -> 5

Complete the following function:

```
def getTwoStringWithHighestFitness(dictionnary: dict) -> Tuple[str, str]:
```

Example of result:

```
(“bonkye”, “momyek”)
```

4 - Making a crossover

After that, we're going to work on the most significant phase in a genetic algorithm :

the **CROSSOVER**

In this step, we reproduce a new population of n elements from the selected elements. In this step, we have to permute and combine as many possible words from the characters obtained from the two parent words that were selected in the previous step. In our example, the parent words are "**bonkye**" and "**momyek**".

For example, we can pick the last 3 words from the word "**bonkye**" and first 3 words from the word "**momyek**" and form a new word as "**bonyek**".

After applying all possible combinations from the word "**bonkye**" and "**momyek**", we get a new population set !

Complete the following function:

```python
def getAllPossibilitiesOfCrossingOverTwoStrings(str1: str, str2: str) -> VectorStr:
```

Example of a new population with "**bonkye**" and "**momyek**" as parents :

```
['momyek', 'bomyek', 'bomyek', 'bonyek', 'bonkek', 'bonkyk']
```

Advice: The larger the population, the more likely the program will converge to the solution.

5 - Making a mutation

There are chances that from the crossover phase, we might get a population which will not contribute to the evolution of a new diverse population and our algorithm will converge prematurely. So we need to alter the sequence of words from 1% (per example) of the newly created population to maintain this diversity.

We can choose any sort of alteration.

Complete the following mutations functions:

```
def changeRandomlyOneCharacter(str: str) -> str:

def swapTwoRandomCharacterInString(str: str) -> str:
```

These both functions can be used as type of mutations but you are free to create your own type of mutation.

Complete the following function:

```
def mutationOfTenPercent(str: str) -> str:
```

Use 'random' library to mutate the string in parameter with 10% of chance.

Finally, when does this process stop ?

Repeating the previous steps until the population has the ‘goal’ word is a good deal, no ?

So, let’s code this function !

```
def checkSolutionInVectorStr(arrayStr: VectorStr) -> bool:
```

Now, we’re ready to code our program to let the “natural” selection do the job !

```
def GeneticAlgorithm() -> VectorStr:
```

Hint : I’ll give you the pseudocode of the genetic algorithm ;)

```
● START
    ○ -> create the initial population
● Compute fitness
● REPEAT :
    ○ -> Selection
    ○ -> Crossover
    ○ -> Mutation
    ○ -> Compute fitness
● UNTIL population has converged
● STOP
```

If you have some trouble developing the algorithm, you can contact me or be helped by the solution (don’t abuse it)

Once the algorithm is finished, try to add some features like the number of generations the program generated to find the goal word or try to optimize your functions (look at the solution description to see what functions you need to optimize to improve your genetic algorithm).

Great algorithm but why should it be used in Artificial Intelligence ?

We can implement Genetic Algorithms to learn the best hyper-parameters for Neural Networks.

Genetic Algorithms can be used to solve various types of optimization problems.

I recommend the following A.I. projects that use the Genetic Algorithm :

Neural Network + Genetic Algorithm + Game = ❤

Reproducing Images using a Genetic Algorithm with Python

Sources and Documentation:

- https://medium.com/xrpractices/reinforcement-learning-vs-genetic-algorithm-ai-for-simulations-f1f484969c56
- https://medium.com/analytics-vidhya/understanding-genetic-algorithms-in-the-artificial-intelligence-spectrum-7021b7cc25e7
- https://en.wikipedia.org/wiki/Genetic_algorithm
- https://becominghuman.ai/my-new-genetic-algorithm-for-time-series-f7f0df31343d

Made by Alexandre Juan