Requirements

Needed:

1. music21: to handle the music and sound part.
2. random: to generate the random mutations and variations.

Might be necessary in the future:

1. Numpy: to handle easily the arrays.
2. Scikit-learn: to use the algorithm to distinguish automatically cacophonies.

The Algorithm

Songs as lists

The idea behind this genetic algorithm is pretty simple and replicates the biologic evolution in nature.
The first step is to create a population of 20 individuals, that are the songs of 16 notes, and to attribute the fitness, that is an evaluation of their goodness. It may take a while, so a good bias is to select at first some songs that we know are good:

['A4', 'F4', 'E4', 'D4', 'A4', 'A4', 'D4', 'A4', ‘G4', 'F4', 'G4', 'G4', 'A4', 'F4', 'E4', 'D4']

This is the transcription using the american notation of a very famous european song. Its fitness in a scale in range 0-10 is surely 10. I need 20 individuals - the more the merrier - so 10 will be created randomly and 10 will be some real songs whose fitness is 10/10.

It’s difficult to handle strings in machine learning therefore I chose to convert each note of the Major Scale to an integer number for a total of 12 numbers. For instance the list above here would be:

[9, 5, 4, 2, 9, 9, 2, 9, 7, 5, 7, 7, 9, 5, 4, 2]

This has been done using the music21 library developed and released by the MIT.

Create the .midi files and associate the fitness

To listen to the randomly created music I used the built-in function of music21 to create a midi file. Substantially is the reverse process of the encoding of the notes. After listening to the 10 musics I assigned to each of them a value, which is the fitness. I manually compiled the list with those values.

Find the two parents

The goal is to find two songs whose fitness is high but don't have to be necessarily the best two. Thus for each parent are drawn three numbers, each of them represents a song (0 is the first one and so on). Of this three songs I chose the one with the highest fitness. In this way I got my two parents.

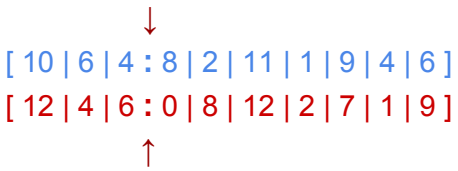
The crossover

In biology two parents mix their DNA to generate a son. Here happens the same. The DNA of cellule is the sequence of notes of the song. How to mix the notes?

Let's say these are the two chosen songs:

[10 | 6 | 4 | 8 | 2 | 11 | 1 | 9 | 4 | 6] ---> Parent 1
[12 | 4 | 6 | 0 | 8 | 12 | 2 | 7 | 1 | 9] ---> Parent 2

A random number is drawn: 3, for example. Let's go to position 3 in the lists and truncate them there.



And now the swapping. The final result is:

[12 | 4 | 6 | 0 | 8 | 12 | 2 | 7 | 1 | 9] ---> Child 1
[10 | 6 | 4 | 8 | 2 | 11 | 1 | 9 | 4 | 6] ---> Child 2

That's it. Easy, isn't it?

The Mutation

Well, normally a mutation is a changing of a value of one unit, positive or negative, that happens with a certain probability. But in this case we can use a trick. In fact we now that C (Do) sounds better with A (La) or E (Mi), so it makes more sense that the mutation changes the value so that the note beside it is two tones higher or lower. Let's make an example.

['A4', 'F4', 'E4', 'D4', 'A4', 'A4', 'D4', 'A4', 'G4', 'F4', 'G4', 'G4', 'A4', 'F4', 'E4', 'D4']

Pick a random number: 6. In position 6 (counting the 0) we have an A (La). The mutation would be to change the note beside the A4 to a C4 or to an F4. To decide which of the two options is drawn a bool. True means *increase* and False *decrease.* The probability of mutation is set as a parameter of the function. I'd say that 5% is fair enough.

Revaluation

Now that two children have been generated, they must be listened and evaluated in another fitness list named... let me think… fitness1!

Elitism

This word might be unknown. It describes *the dominance of a society or system by an elite.* In this case elite = the-song-with-the-highest-fitness. So the music in the children population with the lowest fitness is deleted and substituted by the one with the highest fitness in the parents' population. *Yes but in starting_population 10 arrays has the same fitness*. And here is used another time the random module. Choose one and that's it.

Repeat, repeat, repeat…

This process must be repeated as long as you don't get acceptable music, evaluating each time the fitness. Each set of children is said generation.

Wanna improve the model?

Of course you can. Even typing random keys the code would be better. Even though my Github is almost empty I have many projects and no much time to dedicate to this one. However I'm going to write a machine learning algorithm to evaluate the fitness automatically. And so an iterator. It would be great to train 100 generations in a couple of minutes!

Tommaso Sala, 2nd July 2019