Logo, company name

Description automatically generated

Genetic Algorithms

Jorn Schampheleer

Professionele Bachelor Elektronica-ICT Mentor: Johan Donné

Academiejaar 2020-2021

Logo, company name

Description automatically generated

Genetic Algorithms

Jorn Schampheleer

Professionele Bachelor Elektronica-ICT Mentor: Johan Donné

Academiejaar 2020-2021

Abstract

**Genetic Algorithms**

J. Schampheleer

This report discusses briefly how Genetic Algorithms work. It then compares some of the more prominently used libraries in python. In the end, a sample custom implementation of a Genetic Algorithm will be constructed.

Inhoud

[Figurenlijst 2](#_Toc53607363)

[Tabellenlijst 3](#_Toc53607364)

[1 Genetic Algorithm 4](#_Toc53607365)

[1.1 Wat is een Genetic Algorithm [1] 4](#_Toc53607366)

[1.2 Geschiedenis [4] 7](#_Toc53607367)

[1.3 Toepassingen [5] 7](#_Toc53607368)

[2 Libraries in python 8](#_Toc53607369)

[2.1 Genetic Algorithm Library [7] 9](#_Toc53607370)

[2.2 PyEvolve Library [8] 11](#_Toc53607371)

[2.3 DEAP Library [9] [10] 15](#_Toc53607372)

[2.4 NEAT Library [11] [12] 19](#_Toc53607373)

[2.5 Conclusie 19](#_Toc53607374)

[3 Eigen implementatie 20](#_Toc53607375)

[3.1 Probleemstelling 20](#_Toc53607376)

[3.2 Hypothese 20](#_Toc53607377)

[3.3 Strategie 20](#_Toc53607378)

[3.4 Implementatie [13] 20](#_Toc53607379)

[3.5 Evaluatie van het algoritme 23](#_Toc53607380)

[3.6 Besluit 24](#_Toc53607381)

[Bijlagen 26](#_Toc53607382)

[Bijlage 1 26](#_Toc53607383)

[Bijlage 2 27](#_Toc53607384)

[Bijlage 3 28](#_Toc53607385)

[Bibliografie 30](#_Toc53607386)

# Figurenlijst

[Figuur 1Proces van een GA. Bron: https://towardsdatascience.com/a-simple-genetic-algorithm-from-scratch-in-python-4e8c66ac3121 4](#_Toc53607387)

[Figuur 2 Illustratie bij DNA. Bron: https://blog.nebula.org/genome/ 5](#_Toc53607388)

[Figuur 3 Crossover. Bron: https://nl.wikipedia.org/wiki/Crossing-over 6](#_Toc53607389)

[Figuur 4 De hyperparamters van een Convolutional layer. Bron: https://www.tensorflow.org/api\_docs/python/tf/keras/layers/Conv2D 7](#_Toc53607390)

[Figuur 5 Stickman Gravity Run. Bron: https://apkpure.com/stickman-gravity-run-free/com.OvertyGames.StickmanGravityRun 8](#_Toc53607391)

[Figuur 6 De meest gebruikte programmeertalen voor machine learning. Bron: https://www.houseofbots.com/news-detail/11501-4-most-popular-10-programming-languages-being-used-for-machine-learning-according-to-github 9](#_Toc53607392)

[Figuur 7 De parameters van het geneticalgorithm in de geneticalgorithm library. Bron: https://pypi.org/project/geneticalgorithm 9](#_Toc53607393)

[Figuur 8 Voorbeeldcode van geneticalgorithm. Bron: https://pypi.org/project/geneticalgorithm 11](#_Toc53607394)

[Figuur 9 Voorbeeldcode van pyevolve. Bron: http://pyevolve.sourceforge.net/examples.html#example-1-simple-example 14](#_Toc53607395)

[Figuur 10 De createor die een subklasse FitnessMulti afleidt van base.Fitness. Bron: https://deap.readthedocs.io/en/master/tutorials/basic/part1.html 15](#_Toc53607396)

[Figuur 11 De creator die een subklasse Individual van de list klasse afleidt. Bron: https://deap.readthedocs.io/en/master/tutorials/basic/part1.html 15](#_Toc53607397)

[Figuur 12 De toolbox klasse van DEAP. Bron: https://deap.readthedocs.io/en/master/tutorials/basic/part1.html 16](#_Toc53607398)

[Figuur 13Fitness Function bij DEAP. Bron: https://deap.readthedocs.io/en/master/tutorials/basic/part2.html 16](#_Toc53607399)

[Figuur 14 Mutation bij DEAP. Bron: https://deap.readthedocs.io/en/master/tutorials/basic/part2.html 16](#_Toc53607400)

[Figuur 15 Inhoud van de tools klasse bij DEAP. Bron: https://deap.readthedocs.io/en/master/api/tools.html#module-deap.tools 17](#_Toc53607401)

[Figuur 16 Instellen van crossover, mutatie en selectie bij DEAP. Bron: https://deap.readthedocs.io/en/master/tutorials/basic/part2.html 17](#_Toc53607402)

[Figuur 17 Parallellisatie met scoop in DEAP. Bron: https://deap.readthedocs.io/en/master/tutorials/basic/part4.html 18](#_Toc53607403)

[Figuur 18 Parallellisatie met multiprocessing in DEAP. Bron: https://deap.readthedocs.io/en/master/tutorials/basic/part4.html 18](#_Toc53607404)

[Figuur 19 Voorbeeldcode DEAP. Bron: https://github.com/DEAP/deap 18](#_Toc53607405)

[Figuur 20 NEAT algoritme dat leert om flappy bird te spelen. Bron: https://github.com/SirBob01/NEAT-Python 19](#_Toc53607406)

[Figuur 21 De output van de eigen implementatie van een GA 23](#_Toc53607407)

[Figuur 22 De output van het GA gebruik makend van geneticalgorithm 23](#_Toc53607408)

[Figuur 23 Een tweede run van de eigen implementatie van een GA 24](#_Toc53607409)

[Figuur 24 Een tweede run van het GA gebruik makend van geneticalgorithm 24](#_Toc53607410)

# Tabellenlijst

[Tabel 1 Parameters van geneticalgorithm library 10](#_Toc53607411)

[Tabel 2 Features van pyevolve. Bron: http://pyevolve.sourceforge.net/intro.html#ga-features 13](#_Toc53607412)

# 1 Genetic Algorithm

## 1.1 Wat is een Genetic Algorithm [1]

Een Genetic Algorithm (GA) is een zoekalgoritme dat sterk gebaseerd is op de evolutietheorie van Charles Darwin [2]. Het algoritme reflecteert de natuurlijke selectie waarbij enkel de sterksten overleven en hun genen doorgeven aan volgende generaties. Het verschil is dat hier, in vergelijking tot andere algoritmes die steunen op complexe wiskunde, geopteerd wordt voor een algoritme gebaseerd op de werkelijkheid. Deze werkelijkheid is vaak minder complex.

GA’s volgen steeds volgend proces:

Diagram

Description automatically generated

Figuur 1Proces van een GA. Bron: <https://towardsdatascience.com/a-simple-genetic-algorithm-from-scratch-in-python-4e8c66ac3121>

### 1.1.1 Population

Om te kunnen spreken van natuurlijke selectie is eerst en vooral uiteraard een populatie nodig. Daarom zal de eerste stap van een GA dus altijd het initialiseren van de populatie zijn. Individuen van een populatie kunnen op verschillende manieren geïnitialiseerd worden. De meest voor de hand liggende opties zijn volledig random of volgens een bepaalde lijst van vooraf gedefinieerde waarden. Deze waarden maken dan het “chromosoom” van het individu op.

In Figuur 2 worden deze termen weergegeven.

A picture containing diagram

Description automatically generated

Figuur 2 Illustratie bij DNA. Bron: <https://blog.nebula.org/genome/>

In deze figuur is te zien hoe een chromosoom eigenlijk het DNA van een individu opmaakt en hoe deze chromosomen bestaan uit genen. De random waardes zijn dus te vergelijken met genen en alle random waardes maken het chromosoom en zullen dus het gedrag bepalen van dit individu.

### 1.1.2 Fitness Calculation

Zoals hiervoor vermeld zijn GA’s afhankelijk van natuurlijk selectie. Om te kunnen bepalen welk individu echter de “sterkste” is zal een criterium nodig zijn. Een voorbeeld hiervan is een glazenwasserpopulatie. In deze populatie zal enkel de glazenwasser die het properst wast overleven. De rest zal failliet gaan. Het valt nu zeer duidelijk aan te voelen dat bij elke populatie zo’n criterium moet bestaan. Dit criterium noemt men de “Fitness Function”. Deze zal een individu evalueren en geeft dan een score weer om aan te duiden hoe “fit” een individu is.

### 1.1.3 Terminate?

Optioneel kan gekozen worden om een terminate optie toe te voegen. Wanneer bijvoorbeeld de perfecte ruitenwasser gevonden is hoeft niet meer verder gezocht te worden. Niet elke situatie leent zich hiertoe, soms kan niet bepaald worden op voorhand wat de “perfecte” score is, of soms kan deze gewoon niet bereikt worden en zal de hoogst gevonden fitscore moeten dienstdoen.

### 1.1.4 Selection

Op basis van de voorgaande items, namelijk de populatie en de fitness calculation kan selectie uitgevoerd worden. De individuen met de hoogste fitness score worden dan gekozen om zich voort te planten. Voortplanten betekent het doorgeven van het DNA, en dus de willekeurig gekozen beginwaarden waaruit het chromosoom werd opgebouwd bij de initialisatie van de populatie. Op deze manier benadert deze stap opnieuw de werkelijkheid. Enkel de beste glazenwassers zullen geld op de tafel kunnen brengen en voor een gezin kunnen zorgen. Zij zullen dus de enige met kinderen zijn. Op deze manier valt nu reeds te bemerken dat een convergentie zal optreden. Enkel de beste genen zullen doorgegeven worden en dus in de generatie daarna zullen de beste genen van de beste genen doorgegeven worden etc. Belangrijk is hier dat bij een slecht gekozen fitness function de foute individuen overblijven en dus de populatie zullen dicteren na enkele generaties.

### 1.1.5 Crossover

Text, letter

Description automatically generated

Figuur 3 Crossover. Bron: <https://nl.wikipedia.org/wiki/Crossing-over>

Crossover in deze context heeft betrekking tot het voorplanten van de fittest. Zoals hierboven vermeld vindt na de selectie de voortplanting plaats van de meest fitte individuen. Ook hier werd gekeken naar het echte leven waar chromosomen over elkaar kunnen liggen tijdens de celdeling (meiose). De chromosomen zullen dan deels erfelijk materiaal uitwisselen. Bij GA’s vindt hetzelfde fenomeen plaats. Twee individuen combineren hun erfelijk materiaal. Op dit moment zijn er dus twee chromosomen aanwezig. Om hieruit kinderen te maken (kinderen hebben nooit het unieke DNA van 1 ouder) vindt crossover plaats waarbij elk kind een deel van het DNA van beide ouders heeft. Op dit moment is het kind dus een combinatie van twee DNA-combinaties van sterke individuen. Op zich is nu een goede basis aanwezig voor het algoritme. Er vindt natuurlijke selectie plaats en de sterkere individuen kunnen zich voortplanten. Het algoritme is echter nog niet compleet. Er bestaat een kans dat beide kinderen slechter zijn dan hun ouders. Om deze reden worden meestal de ouders bijgehouden in de populatie, tot ze niet meer de fitste zijn en sterven door de natuurlijke selectie. Een ander probleem manifesteert zich doordat de populatie geïnitialiseerd kan worden met allemaal willekeurige waarden. Dan zou het kunnen dat gestart wordt van een populatie van allemaal slechte glazenwassers. Het is mogelijk dan dat ze ofwel allemaal sterven, ofwel net fit genoeg zijn om te overleven en hun slechte genen door te geven. Het zou dus kunnen dat nooit tot een sterke glazenwasser gekomen wordt die het volledige potentieel van de menselijke genen benut. Om dit probleem op te lossen wordt nog een laatste stap in het algoritme geïntroduceerd.

### 1.1.6 Mutation

Een mutatie in een GA overschrijft een waarde in een gen van een individu. Op deze manier kan zelfs in een uiterst slechte populatie nog steeds de kans bestaan dat een individu een mutatie doet en op die manier een gen ontwikkelt dat beter is dan de anderen. Dit individu zal dan gaan voortplanten en verder muteren en op die manier kan nog steeds doorheen vele generaties een “sterkste” individu manifesteren. Het zorgt ook voor diversiteit in de populatie zodat niet alle individuen zich hetzelfde gaan gedragen. De sterkste staat vermeld tussen aanhalingstekens. Dit omdat het niet altijd mogelijk is om zeker te zijn dat een sterkste individu gevonden is dat ooit zou kunnen bestaan. Net zoals de homo erectus dacht dat deze de sterkste mens was ooit na het uitroeien van de homo habilis [3] kwam uiteindelijk toch de homo sapiens. Enkel wanneer uit de fitness function valt af te leiden wat de hoogst mogelijke fitness is kan verzekerd worden dat een individu weldegelijk de sterkste is.

### 1.1.7 Repeat

In conclusie doorloopt het algoritme dus volgende cyclus opnieuw en opnieuw. De individuen worden onderworpen aan een fitheidstest, de sterksten onder hen krijgen de kans om zich voort te planten en hun DNA te delen, tijdens het delen kunnen hun kinderen mutaties ontwikkelen. Deze kinderen vormen een nieuwe generatie die de cyclus opnieuw doorloopt. Een GA kan dus vele generaties nodig hebben om een optimaal individu te vinden. Een GA wordt wel een zoek heuristiek genoemd omdat ondanks de vele generaties er wel een methodische en systematische benadering tot het probleem is.

## 1.2 Geschiedenis [4]

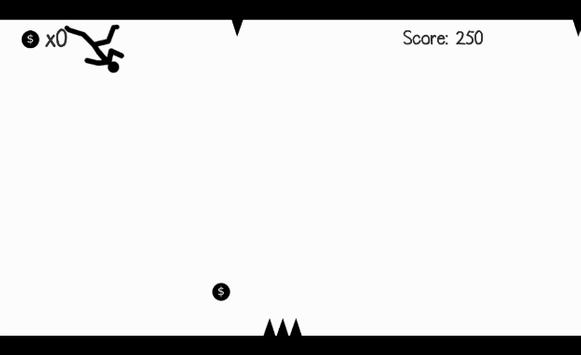
Al in 1950 Stelde Alan Turing voor om leeralgoritmes te onderzoeken gebaseerd op de natuur en evolutie. Vervolgens publiceerde Nils Aall Barricelli in 1963 een werk waarin hij de evolutie documenteerde van een AI die een spel leerde spelen. De meeste info rond GA’s bleef echter puur theoretisch tot eerste internationale conventie over GA’s in 1980. Sinds dan begonnen GA’s snel te evolueren hoewel de grootste doorbraken er pas kwamen na de groei van machine learning in 1990.

## 1.3 Toepassingen [5]

Heden zijn GA’s zowat overal te vinden. De velden waarin ze gebruikt worden gaan van natuurwetenschappen tot economie en van sociale wetenschappen tot ingenieurstechnieken. De belangrijkste implementatie van GA’s is het optimaliseren van hyperparameters bij neurale netwerken die zowat overal te vinden zijn tegenwoordig. Een voorbeeld hiervan is image recognition. Dit is zeer actueel. Image recognition maakt gebruik van CNN’s of Convolutional Neural Networks. Wanneer naar de documentatie van een Convolutional layer kijkt is direct op te merken dat deze veel hyperparamters heeft.



Figuur 4 De hyperparamters van een Convolutional layer. Bron: <https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D>

De belangrijkste zijn: filters, kernel\_size en strides. Met een GA kan een populatie gemaakt worden waar deze waarden allemaal willekeurig ingesteld worden. Met andere woorden een individu heeft een chromosoom met volgende genen: filters, kernel\_size en strides. Het model kan dan getraind worden en de accuracy op de validatie set kan gebruikt worden als fitness function. Op deze manier kan op een systematische en gestructureerde manier de beste implementatie van een Conv2D laag in een CNN gezocht worden. In dit geval is het zoeken naar goede parameters voor de hand liggend en makkelijk te vinden. Bij grotere neurale netwerken is dit echter niet meer het geval en kan het erg moeilijk zijn om de hyperparameters te optimaliseren, tenzij met een brute force. Het trainen van neurale netwerken duurt echter heel lang, zeker voor grotere datasets. Het is daarom aangeraden om op een gestructureerde manier te werk te gaan bij het zoeken naar deze onbekenden en een GA is hiervoor uitermate geschikt. Een andere toepassing, al is deze eerder recreatief, is het leren spelen van videogames. Wanneer de virtuele wereld kan afgeleid worden naar enkele basis inputs, kan een GA genen krijgen die specifiëren hoe er gereageerd moet worden op deze inputs. Een simpel voorbeeld hiervan is Stickman Gravity Run.

Figuur 5 Stickman Gravity Run. Bron: <https://apkpure.com/stickman-gravity-run-free/com.OvertyGames.StickmanGravityRun>

De bedoeling is dat de speler wanneer deze te dicht bij de driehoeken komt, de zwaartekracht omdraait en valt of stijgt naar de andere rand. In dit geval zouden de inputs voor het algoritme kunnen zijn: de x-afstand tot de dichtstbijzijnde driehoek en de y-afstand tot de dichtstbijzijnde driehoek. De populatie kan worden geïnitialiseerd met willekeurige waarden over wanneer (vanaf welke x-afstand en welke y-afstand) het individu tikt op het scherm. Uiteraard zal de score de fitness function zijn hier. Hier is duidelijk de theorie van de “overleving van de sterkste” zoals Darwin deze documenteerde te herkennen. Alleen de slimste en vaardigste algoritmen zullen lang overleven in het spel.

# 2 Libraries in python

Op vlak van machine learning is python nog steeds de meeste prominent aanwezige programmeertaal.

A picture containing diagram

Description automatically generated

Figuur 6 De meest gebruikte programmeertalen voor machine learning. Bron: <https://www.houseofbots.com/news-detail/11501-4-most-popular-10-programming-languages-being-used-for-machine-learning-according-to-github>

Figuur 6 toont de meest gebruikte programmeertalen voor machine learning volgens GitHub. De afbeelding dateert van 2019 maar python staat ook in 2020 nog steeds bovenaan [6]. Het is dus logisch dat in python veel libraries voor GA’s terug te vinden zijn.

## 2.1 Genetic Algorithm Library [7]

De geneticalgorithm library is een zeer lightweight en simpele library die enkel de basis omvat omtrent GA’s en bijna geen ruimte laat voor customisatie. De parameters van de geneticalgorithmlibrary worden geïllustreerd in Figuur 7.



Figuur 7 De parameters van het geneticalgorithm in de geneticalgorithm library. Bron: https://pypi.org/project/geneticalgorithm

Bovenstaande figuur toont dat de parameters voor het algoritme inderdaad zeer beperkt zijn. In Tabel 1 worden deze parameters kort toegelicht.

|  |  |
| --- | --- |
| max\_num\_iteration | Het maximum aantal generaties. Indien deze niet gespecifieerd wordt zal het algoritme dit zelf instellen op basis van het aantal genen, de mogelijke waarden van die genen en de grootte van de populatie. Dit kan dus snel zeer groot worden en het wordt sterk aangeraden in de documentatie om deze parameter te specifiëren. |
| population\_size | De grootte van de populatie. |
| mutation\_probability | Bepaalt de kans van elk gen van elk individu dat dit vervangen wordt door een willekeurige waarde. |
| elit\_ratio | Dit bepaalt hoeveel “elite” individuen worden bijgehouden. Een elite individu kan meerdere generaties oud zijn en toch bijgehouden worden om zijn goede genen. Teveel elite individuen bijhouden is een slecht idee omdat de elite individuen de populatie zullen domineren en het mogelijk is om vast te komen te zitten in een lokaal maximum. Het kan echter wel helpen om sneller tot de beste genen te komen door een paar elite individuen bij te houden. |
| crossover\_probability | Bepaalt de kans dat een individu zijn genen doorgeeft aan zijn kinderen. |
| parents\_portion | Bepaalt het percentage van ouders in de volgende generatie. |
| crossover\_type | Deze heeft 3 mogelijkheden:   * one\_point * two\_point * uniform   Hierover staat niet veel verdere uitleg al spreken de namen voor zich. De documentatie raad aan om uniform te gebruiken in de meeste gevallen. |
| max\_iteration\_without\_improv | Indien het algoritme te lang niet beter wordt zal het stoppen en de best gevonden oplossing teruggeven. Er moet hier wel voorzichtig mee omgesprongen worden. Een te kleine waarde zou ervoor kunnen zorgen dat niet voorbij een lokaal maximum geraakt kan worden. |

Tabel 1 Parameters van geneticalgorithm library

Om het algoritme dan aan te roepen wordt de code gebruikt die gegeven is in Figuur 8.



Figuur 8 Voorbeeldcode van geneticalgorithm. Bron: https://pypi.org/project/geneticalgorithm

Deze code maakt individuen aan met 3 genen die elk een waarde tussen 0 en 10 hebben. De populatie is 100 groot. Het model wordt aangemaakt met de ga() constructor. Er wordt een function parameter meegegeven die verwijst naar de fitness function. In dit geval zal de fitness function de som nemen van de 3 genen. Een eerste limitatie valt hier op te merken: er kan niet gespecifieerd worden of een zo hoog mogelijke score het doel is, of een zo laag mogelijke score. Vervolgens is er de dimension die het aantal genen weergeeft. De variable type geeft aan dat er reële getallen zitten in de genen, en de variable\_boundaries geven aan welke waarden deze kunnen aannemen (minimum en maximum). Tot slot kunnen ook de instellingen meegegeven worden te vinden in Tabel 1. Deze manier is zeer snel en makkelijk te implementeren en voldoende voor simpele problemen. Indien echter met klassen gewerkt zou worden voor complexere genen zal dit niet meer volstaan. Er is namelijk geen manier om te bepalen hoe mutatie van een klasse werkt. De mutatie kan dus niet zelf geïmplementeerd worden.

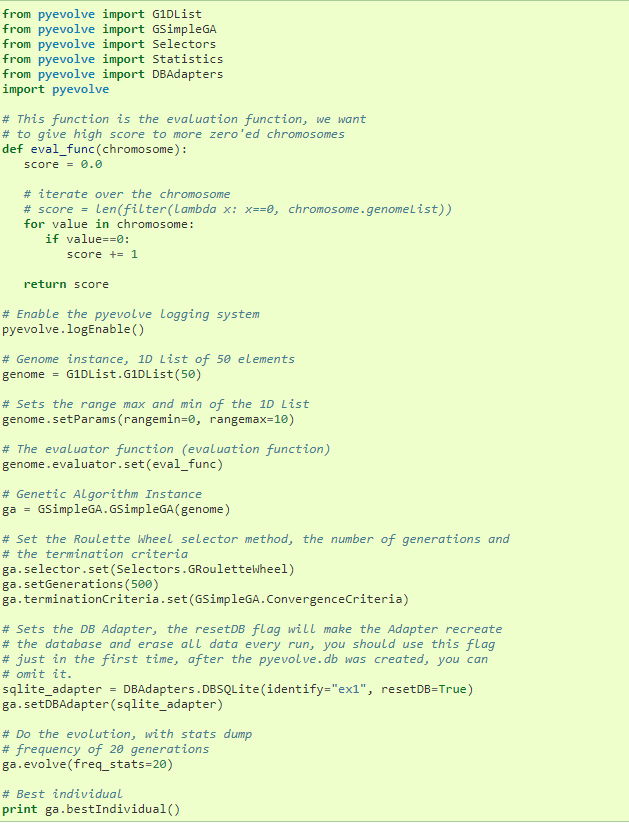
## 2.2 PyEvolve Library [8]

PyEvolve is een library die reeds meer mogelijkheden tot customisatie biedt. Reeds in de intro staan verschillende features gespecifieerd die de gebruikers meer vrijheden geven in tegenstelling tot geneticalgorithm. De belangrijke features worden voorgesteld in

|  |  |
| --- | --- |
| Chromosoom features | 1DList: stel de genen voor als een 1D-lijst |
| 2DList: stel de genen voor als een 2D-lijst |
| 1DBinaryString: stel de genen voor als een bitstring |
| Crossover features | 1D: Single point, two point, uniform  2D: Uniform, single vertical point, single horizontal point |
| Mutatie features | 1DBinaryString: Swap- en flipmutatie  1DList: Swap Mutator, Integer Range Mutator, Real Range Mutator, Integer Gaussian Mutator, Real Gaussian Mutator, Integer Binary Mutator, Allele Mutator  2DList: Swap Mutator, Integer Gaussian Mutator, Real Gaussian Mutator, Allele Mutator |
| Initisalisators | 1DBinaryString: Binary String Initializator  1DList en 2DList: Allele Initializator, Integer Initializator, Real Initializator |
| Scaling features | Scaling is nieuw in vergelijking met PyEvolve, scaling zal de output van de fitness function eerst nog kunnen scalen naar een andere waarde. De scalers zijn:  Linear Scaling, Sigma Truncation Scaling , Power Law Scaling en Raw Scaling |
| Selectie features | Roulette Wheel Selection: Zoals in een casine met het roulette wiel. Elk individu krijgt een kans op basis van zijn fitness function om gekozen te worden om te overleven.  Rank Selection: Een iets andere manier van kiezen. Bij Roulette Wheel Selection kan het zijn dat een individu 90% fitness heeft en dus telkens gekozen wordt. Er zal dan veel minder variatie zijn. Bij rank selection worden alle individuen op volgorde gezet op basis van hun fitness. De slechtste score komt 1 keer voor, de twee slechtste score komt 2 keer voor etc, op die manier zullen de betere individuen meer voorkomen, maar is er een eerlijkere verdeling van de kansen.  Uniform Selection: Deze zal opnieuw de mogelijkheden uittekenen volgens fitheid, maar zal werken met een discreet aantal pointers waaruit gekozen kan worden. Op deze manier hebben ook zwakkere individuen een kans om gekozen te worden.  Tournament Selection: Bij tournament selection zullen verschillende individuen gekozen worden uit de populatie om te “strijden” tegen elkaar. Degene met de hoogste fitness in zijn tournament wordt gekozen om te overleven. |

Tabel 2 Features van pyevolve. Bron: <http://pyevolve.sourceforge.net/intro.html#ga-features>

Onmiddelijk valt op dat deze library veel meer mogelijkheden heeft. De manier van selectie, alsook mutatie alsook crossover kan aangepast worden. Dit in tegenstelling tot geneticalgorithm waar enkel de manier van crossover kon gekozen worden. Pyevolve biedt ook ondersteuning voor bitstrings en 2D-lijsten, iets wat niet zo is bij geneticalgorithm. Er is ook mogelijkheid tot scaling bij pyevolve, om de resultaten van de fitness function mee te gaan scalen. Om beter te vergelijken staat tussen de examples van pyevolve hetzelfde voorbeeld als bij pyevolve. De code hiervoor is te zien in



Figuur 9 Voorbeeldcode van pyevolve. Bron: <http://pyevolve.sourceforge.net/examples.html#example-1-simple-example>

De code van pyevolve is opmerkelijk langer dan dezelfde code met geneticalgorithm. De fitness function blijft min of meer gelijk alleen wordt deze in de code in Figuur 9 iets anders gedefinieerd. Er wordt een chromosoom aangemaakt in een 1D-Lijst van 50 genen en deze moeten allemaal binnen het interval 0-10 blijven. Daarna wordt de fitness function gekoppeld en wordt gekozen voor een GSimpleGA (of met andere woorden een simpel GA). De selector kan nu ingesteld worden (iets wat niet mogelijk was bij pyevolve) en er wordt een maximum aantal generaties ingesteld. Er wordt ook een early stopping mechanisme ingebouwd met de terminationCriteria parameter. Dit is gelijkaardig aan de “max\_iterations\_without\_improvement” parameter bij geneticalgorithm. Hierna volgt nog een feature die niet aanwezig was in geneticalgorithm. Pyevolve kan automatisch data over de generaties wegschrijven naar een database. Hierna wordt de evolve methode aangeroepen om het proces te starten en kan de beste individu teruggevonden worden met ga.bestIndividual(). Het grootste verschil met geneticalgorithm is dat pyevolve veel meer customisatie toelaat en dus veel meer controle over het evolutieproces. Het nadeel hieraan is dat ook voor simpele taken meer code nodig is en meer moet nagedacht worden over de verschillende opties.

## 2.3 DEAP Library [9] [10]

Distributed Evolutionary Algorithms in Python (DEAP) is een library die hulpmethoden bevat voor allerlei soorten evolutionaire algoritmen zoals: GA’s, genetic programming, particle swarm optimization, etc. DEAP is het meest uitgebreide van de voorgaande libraries en laat veel ruimte voor customisatie. In plaats van veel logica zelf beschikbaar te stellen aan developers en daarbij vrijheid in te perken zoals geneticalgorithm, kiest DEAP ervoor om factory klassen ter beschikking te stellen zodat de developer zelf de logica nog kan schrijven. Om de volledige mogelijkheden van DEAP te illustreren wordt hieronder kort de “Basic Tutorials” besproken te vinden in Bron [10].

### 2.3.1 DEAP Mogelijkheden

#### Fitness

De creator kan klassen maken afgeleid van verschillende basisklassen. Een eerste basisklasse zo is “base.Fitness”. Deze klasse kan door de creator aangemaakt worden samen met enkele weights parameters. De weights geven aan of voor deze fitness test een zo hoog mogelijke score of een zo laag mogelijke score behaald moet worden. Dit is functionaliteit die in vorige libraries niet aanwezig was of niet voor de hand liggend. In DEAP is dit basisfunctionaliteit. Een voorbeeld van de code voor het afleiden van een subklasse van de abstracte basisklasse “Fitness” is te vinden in Figuur 10.



Figuur 10 De createor die een subklasse FitnessMulti afleidt van base.Fitness. Bron: <https://deap.readthedocs.io/en/master/tutorials/basic/part1.html>

De eerste parameter is de naam van de nieuwe klasse. Deze klassen bestaan enkel tijdens runtime.

#### Individual

Op een gelijkaardige manier aan Fitness kan ook een individu gemaakt worden door af te leiden van een standaardtype (list, int of een eigen klasse zelfs). Deze keer neemt de klasse als parameter de naam van de fitness klasse die gebruikt zal worden voor de evaluatie.

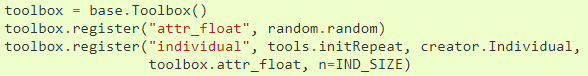


Figuur 11 De creator die een subklasse Individual van de list klasse afleidt. Bron: https://deap.readthedocs.io/en/master/tutorials/basic/part1.html

Doordat de weight van de FitnessMax klasse positief is zal dus gezocht worden naar het individu die de hoogste score haalt op de fitness function.

#### Toolbox

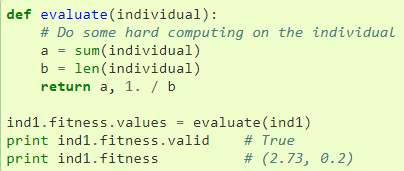
Om data op te slaan en door te geven wordt gebruik gemaakt van de toolbox klasse.



Figuur 12 De toolbox klasse van DEAP. Bron: <https://deap.readthedocs.io/en/master/tutorials/basic/part1.html>

Merk op hoe “attr\_float” gebruikt wordt om de individual te initialiseren (via tools.initRepeat) en op deze manier de lijst op te vullen die Individual op dat moment is. In attr\_float wordt gebruik gemaakt van random.random. Het is niet moeilijk in te zien hoe hier zelf eigen functies als argument kunnen meegegeven worden en de initialisatie van een individu dus volledig te personaliseren valt. Ook dit is een vrijheid die in voorgaande libraries niet in deze mate aanwezig is.

#### Evaluate



Figuur 13Fitness Function bij DEAP. Bron: <https://deap.readthedocs.io/en/master/tutorials/basic/part2.html>

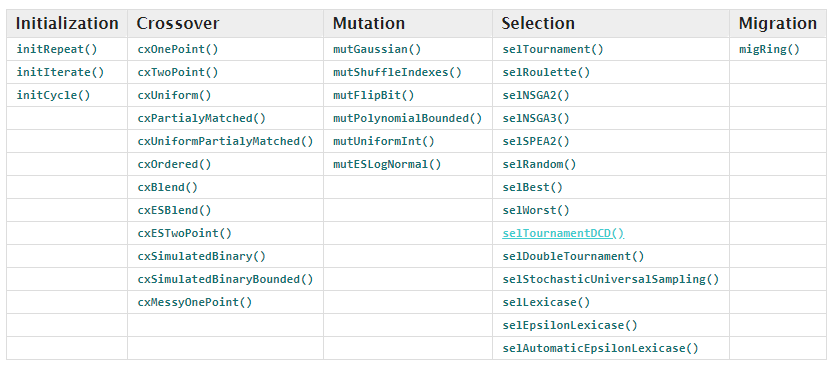
In Figuur 13 is te zien hoe een Fitness function kan geïmplementeerd worden. Belangrijk hier is in te zien hoe meerdere waarden via de fitness function teruggegeven worden. Deze zullen dan vergeleken worden met de weights van de klasse afgeleid van base.Fitness. Negatieve weights zullen zoeken naar de laagste waarde bij de score, positieve naar de hoogste. De vrijheid ligt hier opnieuw volledig bij de developer voor het implementeren van de fitness function. Dit is een fenomeen dat alle libraries delen en is ook logisch.

#### Mutation, selection en crossover



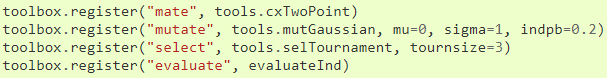
Figuur 14 Mutation bij DEAP. Bron: <https://deap.readthedocs.io/en/master/tutorials/basic/part2.html>

De methoden voor mutatie, selectie en crossover zitten vervat in de tools klasse van DEAP. De inhoud van deze klasse staat aangegeven in Figuur 15.



Figuur 15 Inhoud van de tools klasse bij DEAP. Bron: <https://deap.readthedocs.io/en/master/api/tools.html#module-deap.tools>

Ook al is hier duidelijk te zien hoe DEAP veel meer mogelijkheden heeft op alle vlakken betreffende deze parameters dan de andere libraries, zou het verbazend zijn moest hier de vrijheid ophouden. Het instellen van deze parameters gebeurt als volgt:



Figuur 16 Instellen van crossover, mutatie en selectie bij DEAP. Bron: <https://deap.readthedocs.io/en/master/tutorials/basic/part2.html>

In Figuur 16 is te zien hoe opnieuw de functies worden doorgegeven aan de toolbox. Opnieuw zou de developer hier dus eigen functies kunnen doorgeven aan het algoritme om zo volledige customisatie te benutten.

#### Computing statistics

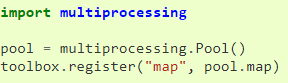
Bij pyevolve werd reeds vermeld dat deze library kon loggen naar een database. DEAP gaat hierin verder en laat toe zelf loggers te schrijven, alsook lambda functies die bepalen hoe gelogd wordt. De logs kunnen als pickle-objecten weggeschreven worden, naar het scherm geschreven worden, of rechtstreeks geplot worden via matplotlib.

#### Multiple processors

DEAP biedt ondersteuning voor multiprocessing. Hiervoor moet het mogelijk zijn om alle objecten te picklen (met andere woorden, alle objecten moeten compatibel zijn met de pickle library). DEAP zal dan alle individuen eerlijk verspreiden over de verschillende processen. DEAP laat net zoals overal de vrijheid aan de developer om te kiezen met welke library gewerkt wordt. In de documentatie staan voorbeelden voor Scalable Concurent Operations in Python (SCOOP) en multiprocessing. Deze mogelijkheden zijn zeer welkom bij intensievere GA’s met veel genen. Dit is dus zeker een belangrijk verschil met de voorgaande libraries.



Figuur 17 Parallellisatie met scoop in DEAP. Bron: <https://deap.readthedocs.io/en/master/tutorials/basic/part4.html>



Figuur 18 Parallellisatie met multiprocessing in DEAP. Bron: <https://deap.readthedocs.io/en/master/tutorials/basic/part4.html>

### 2.3.2 DEAP voorbeeldcode

Om dit hoofdstuk af te sluiten wordt opnieuw de vergelijking gemaakt in code voor het simpele “allemaal nullen probleem”.

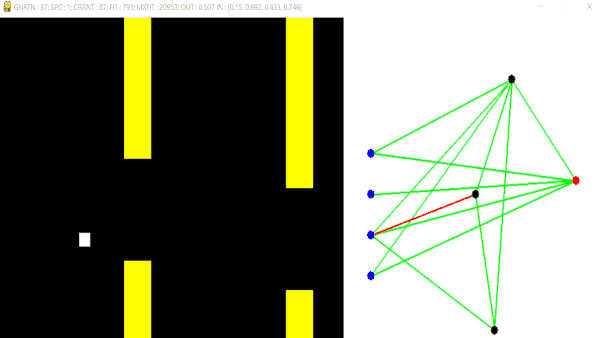


Figuur 19 Voorbeeldcode DEAP. Bron: <https://github.com/DEAP/deap>

Het probleem hier is niet volledig hetzelfde als dat van de vorige libraries. Er wordt hier gezocht naar een maximum in plaats van een minimum. Er wordt ook niet gewerkt met reële getallen maar met binaire opties namelijk 0 of 1. Het principe blijft echter hetzelfde en de code verschilt maar weinig. Een verschil met alle voorgaande algoritmen is dat DEAP vereist dat de for-loop zelf uitgevoerd wordt door de developer. Opnieuw is ook de hoeveelheid code die nodig is voor het uitvoeren toegenomen.

## 2.4 NEAT Library [11] [12]

NeuroEvolution of Augmenting Topologies (NEAT) is een algoritme ontwikkelt door Ken Stanley. Zoals de naam aangeeft is het eigenlijk geen puur GA, maar een NeuroEvolution algoritme. Wat NEAT speciaal maakt is dat het neurale netwerken gebruikt in genetische algoritmen. NEAT werkt net zoals neurale netwerken met inputs en outputs. Inputs stellen een “state” voor van de omgeving waarin het algoritme zich bevindt. Wanneer gekeken wordt naar Figuur 5 en het principe van Stickman Gravity Run kunnen inputs bijvoorbeeld zijn: de hoogte van de speler in de lucht, de richting van de zwaartekracht (naar boven of naar beneden), de x-coördinaat van de dichtstbijzijnde driehoek en de y-coördinaat van de dichtstbijzijnde driehoek. Als outputs is hier slechts 1 waarde nodig namelijk of het algoritme de zwaartekracht zou omkeren of niet. NEAT kan dan gebruik maken van neurons om een neuraal netwerk op te bouwen om zo complexe situaties op te lossen. Dit soort algoritmen benadert de werking van een brein door gebruik te maken van neurons. Het algoritme heeft daardoor een veel breder spectrum aan mogelijkheden. Een voorbeeld gegeven op de github van NEAT-Python [11] is een algoritme dat leert om een simpelere variant van Flappy Bird te spelen.



Figuur 20 NEAT algoritme dat leert om flappy bird te spelen. Bron: <https://github.com/SirBob01/NEAT-Python>

Aan de rechterkant van Figuur 20 staat het brein van de AI. De connecties tussen de neurons zijn duidelijk te herkennen. De vier inputs hier zijn: x-afstand tot de eerstvolgende pijp, y-coördinaat van de speler, y-coördinaat van het bovenste punt van het gat, y-coördinaat van het onderste punt van het gat. De output bepaalt het springen. Daartussen zitten 1 of meerdere hidden layers die NEAT zal trainen. Om NEAT te configureren wordt meestal beroep gedaan op een apart configuratie bestand omdat de opties dusdanig talrijk zijn. Een voorbeeld van zo een configuratie bestand kan gevonden worden in Bijlage 1.

## 2.5 Conclusie

Een beste library bestaat in dit geval niet. Elke library heeft een eigen toepassingsgebied. Geneticalgorithm is zeer nuttig voor kleine en simpele problemen omdat dit met weinig code te implementeren is. Wanneer meer configuratie nodig is kan geopteerd worden voor pyevolve. Indien ook pyevolve niet genoeg configuratie toelaat kan tot slot gebruikt gemaakt worden van DEAP. DEAP heeft de meeste mogelijkheden tot configuratie maar is ook het moeilijkst te implementeren en vereist de meeste code om geschreven te worden. NEAT valt in een andere categorie dan voorgaande algoritmen omdat NEAT niet zomaar gebruik maakt van GA’s maar genen opstelt gemaakt van neurons en connecties. Dit maakt het trainen en implementeren veel complexer maar kan wel leiden tot intelligentere oplossingen zeker wanneer het probleem een individu vereist dat kan redeneren.

# 3 Eigen implementatie

In dit hoofdstuk volgt een eigen implementatie van een GA voor het probleem van de “allemaal nullen” dat ook vermeld staat bij geneticalgorithm en pyevolve.

## 3.1 Probleemstelling

De bedoeling is om te bepalen voor welke verzameling van n getallen de som het dichtst bij 0 is gelegen.

## 3.2 Hypothese

De hypothese is dat wanneer alle n getallen 0 zijn de som het dichtst bij 0 zal liggen.

## 3.3 Strategie

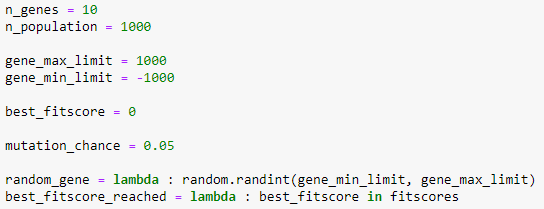
De strategie om dit probleem op te lossen met GA’s is een populatie van x individuen initialiseren met n genen waarvoor elk gen een willekeurige waarde krijgt (ofwel reëel, geheel of natuurlijk). De Fitness Function is de som van alle getallen en moet zo dicht mogelijk bij 0 liggen (hiervoor kan de pythagoreaanse afstand tot het punt (0, 0 , … 0) met in alle n dimensies 0 berekend worden). De afstand moet dan zo klein mogelijk worden, en dus ook de fitness. De Terminate? voorwaarde is voldaan wanneer de fitness 0 is. Er zal gebruik gemaakt worden van zowel crossover, mutatie als selectie.

## 3.4 Implementatie [13]

Aan het begin van de code staan de imports. De code zal de random library gebruiken om genen te genereren en om selectie te doen. Een volledig overzicht van de code is te vinden in Bijlage 2.

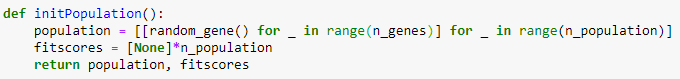


Na deze imports definieert het script enkele globale variabelen.

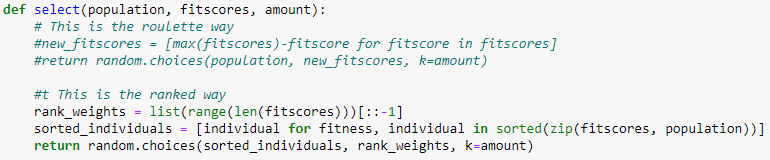


Deze variabelen worden even kort toegelicht:

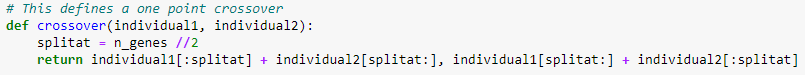
* n\_genes: Het aantal genen van elk individu
* n\_population: Het aantal individuen in een populatie
* gene\_max\_limit: De maximum waarde van een gen
* gene\_min\_limit: De minimum waarde van een gen
* best\_fitscore: De best behaalbare fitscore, zodat het algoritme weet wanneer te stoppen met zoeken
* mutation\_change: De kans op een mutatie van een gen
* random\_gene: een lambda functie die een random waarde voor een gen teruggeeft
* best\_fitscore\_reached: een lambda functie die een boolean teruggeeft of de best behaalbare fitscore bereikt is



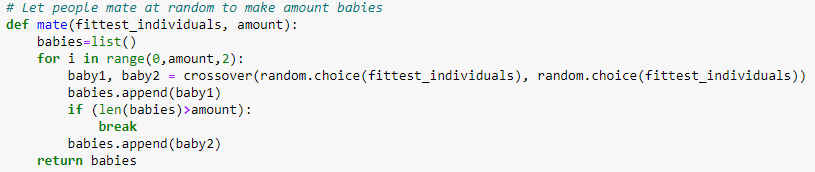
Na het definiëren van de variabelen volgt een functie om de populatie te initialiseren. Deze maakt alle individuen aan en voorziet hen van random genen. De fitscores lijst wordt eveneens aangemaakt en opgevuld met None waardes.



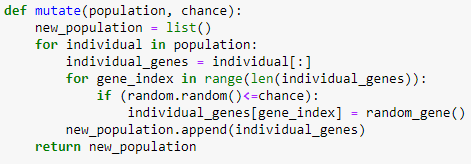
Na de initPopulation functie volgt de select functie. In commentaar staat eerst de roulette manier om aan selectie te doen. Deze manier staat uitgelegd in Tabel 2 bij selectie features. Bij de roulette manier kan de populatie echter snel gedomineerd worden door de sterkste individuen. Daarom staat daaronder de ranked manier die ook uitgelegd staat in Tabel 2. Het algoritme gaat te werk door een lijst te maken van natuurlijke getallen, van groot naar klein. Dit betekent dus: 999, 998, 997, 996, … voor een populatie van 1000 individuen. Daarna sorteert het algoritme de individuen op basis van hun fitheid, de fitste individuen komen bovenaan en de slechtste individuen onderaan. Tot slot wordt gebruik gemaakt van random.choices om k individuen te kiezen volgens een gewicht. Dit betekent dat het fitste individu 999 kansen heeft om geselecteerd te worden voor voortplanting, en het slechtste individu 0 kansen heeft.



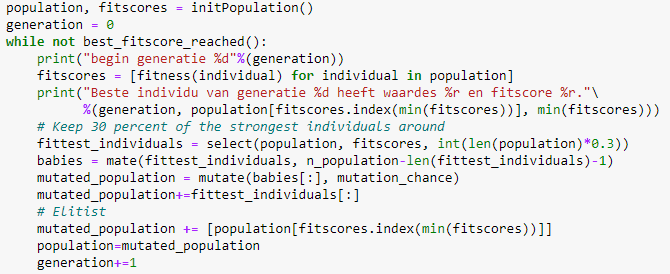
De volgende functie is de crossover functie. Deze neemt 2 individuen om kinderen te maken. De genen van de individuen worden middendoor geknipt en elk kind krijgt de helft van de moeder en de helft van de vader. Dit is een implementatie van one point crossover. Deze functie brengt dus twee baby’s voort.



De mate functie is een functie die de crossover functie zal aanroepen. De amount parameter van de mate functie specifieert hoeveel babies gemaakt moeten worden. De mate functie zal dan telkens uit de fitste individuen er 2 uitkiezen (al of niet dezelfde, ad random) en deze samen 2 baby’s laten maken. Dit wordt herhaald tot er amount baby’s zijn, waarna de lijst van alle nieuwe baby’s wordt teruggegeven.



De laatste functie is de mutate functie. Deze zal alle individuen overlopen, en voor elk gen gaan kijken of het gen aan de kans voldoet om te muteren. Indien dit het geval is wordt dit gen vervangen door een nieuw willekeurig gen.

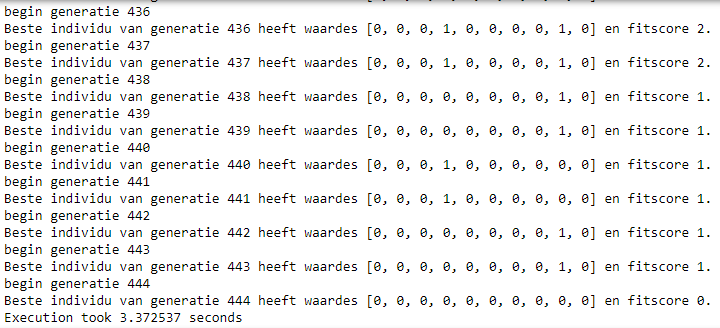


Tot slot resteert nog de main code van het script. Eerst worden de populatie en fitscores aangemaakt en ingesteld met de initPopulation functie. Er wordt ook een variabele generation aangemaakt om bij te houden welke de huidige generatie is. Er wordt dan een loop gestart die blijft lopen zolang de beste fitscore niet gevonden is.

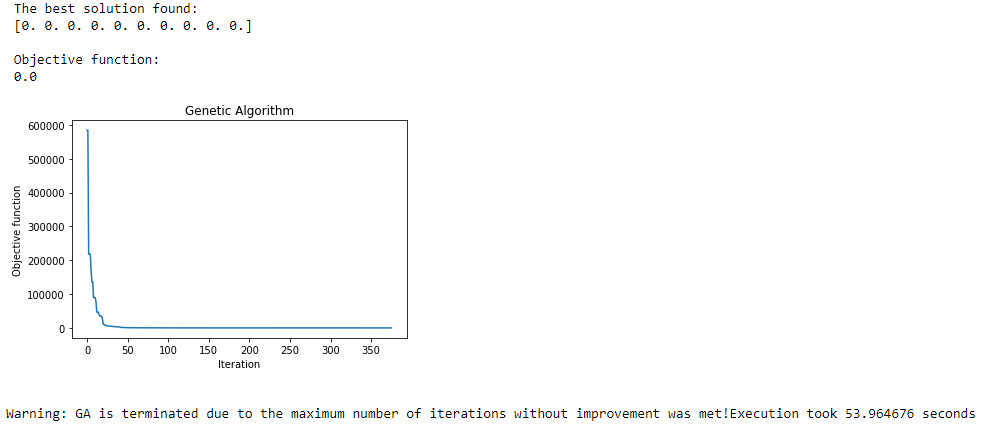
Na de eerste print worden voor alle individuen de fitheidsscores berekend. Na de tweede print wordt natuurlijke selectie toegepast. Er wordt meegeven als parameter dat 30% van de individuen mag overleven. Hierna mogen deze overlevende individuen baby’s maken. Het aantal baby’s dat gemaakt wordt is gegeven door de expressie “n\_population -len(fittest\_individuals) -1”. Dit is omdat de populatie op n\_population individuen constant moet blijven. De fittest individuals blijven echter leven en zullen deel uitmaken van de volgende generatie. Een paar lijnen verder wordt ook nog 1 elitist toegevoegd aan de populatie. Om deze reden moeten er dus baby’s gemaakt worden om de populatie op te vullen, minus het aantal overlevenden en de elitist. In de volgende regel muteren de baby’s met de mutate functie, en worden alle individuen van de volgende generatie in de mutated\_population gestopt. Deze bestaat dus uit 30% overlevende ouders, 1 elitist en de rest baby’s. Als laatste statements wordt de populatie overschreven met de mutated\_population variabele en wordt de generatie verhoogd. Vanaf dan wordt de loop weer hervat.

## 3.5 Evaluatie van het algoritme

Om het algoritme te evalueren wordt het vergeleken met dezelfde oplossing gebruik makend van geneticalgorithm. Er wordt gekeken naar de oplossingscorrectheid en naar de snelheid. De code van het script gebruik makend van geneticalgorithm staat vermeld in Bijlage 3. De volgende figuren vergelijken de output van beide scripts en de tijd die zij namen om uit te voeren:

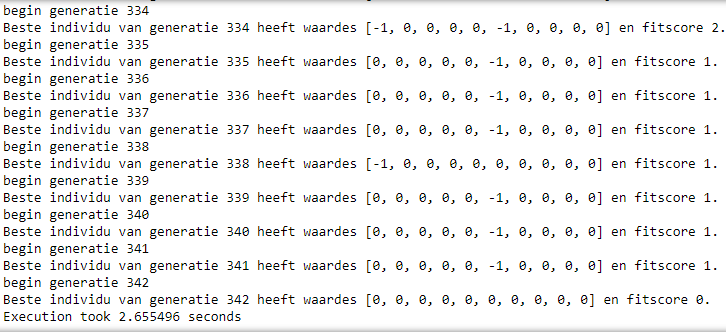


Figuur 21 De output van de eigen implementatie van een GA

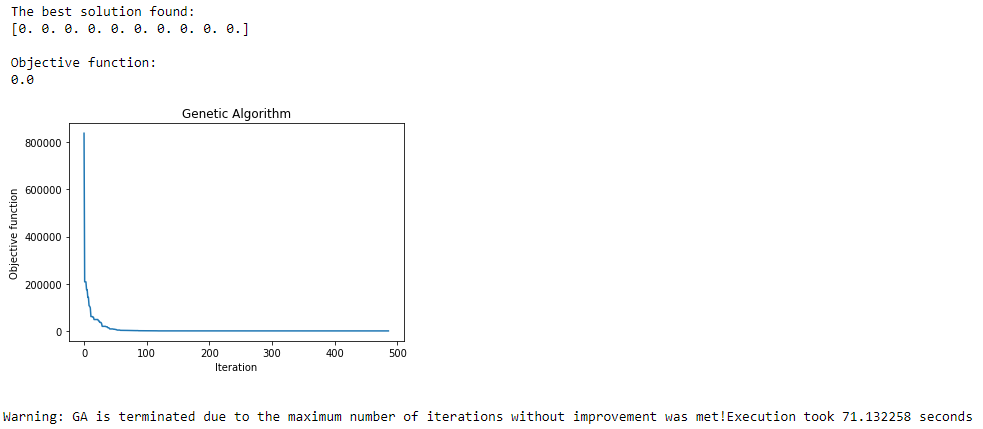


Figuur 22 De output van het GA gebruik makend van geneticalgorithm

Te zien in Figuur 21 en Figuur 22 valt aft e leiden dat mijn eigen implementatie van het algoritme veel sneller is. Er moet wel vermeld worden dat dit vermoedelijk is omdat geneticalgorithm geen parameter gebruikt om aan te geven wat de best behaalbare waarde is van de fitness function. Deze library zal dus nog 100 extra generaties moeten uitvoeren voordat gestopt wordt. Deze 100 generaties verklaren echter niet het tienvoudige tijdsverbruik van geneticalgorithm. Langs de andere kant moet wel vermeld worden dat de eigen implementatie veel meer generaties nodig heeft om te convergeren. De library geneticalgorithm zal slechts ~250 generaties nodig hebben om de juiste oplossing te vinden (het aantal geprobeerde generaties verminderd met de max\_iteration\_without\_improvement parameter die hier ingesteld is op 100). Daarbovenop zijn veel waarden afhankelijk van een willekeurige initialisatie. Wanneer beide scripts opnieuw uitgevoerd worden zijn volgende resultaten waar te nemen:



Figuur Een tweede run van de eigen implementatie van een GA



Figuur Een tweede run van het GA gebruik makend van geneticalgorithm

Nu valt op dat de eigen implementatie minder generaties nodig had om tot de oplossing te komen (342 generaties versus ~400 van geneticalgorithm). De tijd die geneticalgorithm nodig heeft blijft wel nog steeds extreem veel hoger (meer dan een factor 20 deze keer).

## 3.6 Besluit

Zowel de eigen implementatie als de implementatie gebruik makend van geneticalgorithm bekomen tot accurate resultaten. De eigen implementatie is echter veel sneller dan de implementatie met geneticalgorithm. Het is wel zo dat geneticalgorithm veel meer mogelijkheden tot customisatie biedt (zoals onder andere verschillende vormen van crossover, alsook de optie tot early stopping). Het opzet is geslaagd en er is een oplossing gevonden voor het probleem. De hypothese bleek correct en de nuloplossing is de juiste.

# Bijlagen

## Bijlage 1



Bijlage 1 NEAT configuratie bestand. Bron: <https://github.com/CodeReclaimers/neat-python/blob/master/examples/xor/config-feedforward>

## Bijlage 2



Bijlage 2 Code van het eigen GA

## Bijlage 3



Bijlage 3 De vergelijkingscode gebruik makend van geneticalgorithm

# Bibliografie

|  |  |
| --- | --- |
| [1] | V. Mallawaarachchi, „Towards Data Science,” 8 Juli 2017. [Online]. Available: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3. [Geopend 13 Oktober 2020]. |
| [2] | Wikipedia, „Wikipedia,” [Online]. Available: https://nl.wikipedia.org/wiki/Evolutietheorie. [Geopend 13 Oktober 2020]. |
| [3] | Wikipedia, „Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Archaic\_humans. [Geopend 13 Oktober 2020]. |
| [4] | Wikipedia, „Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Genetic\_algorithm. [Geopend 13 Oktober 2020]. |
| [5] | Wikipedia, „Wikipadia,” [Online]. Available: https://en.wikipedia.org/wiki/List\_of\_genetic\_algorithm\_applications. [Geopend 13 Oktober 2020]. |
| [6] | P. Banerjee, „Geeks For Geeks,” 26 Juni 2020. [Online]. Available: https://www.geeksforgeeks.org/top-5-programming-languages-and-their-libraries-for-machine-learning-in-2020/. [Geopend 14 Oktober 2020]. |
| [7] | R. (. Solgi, „PyPI,” 4 Mei 2020. [Online]. Available: https://pypi.org/project/geneticalgorithm. [Geopend 14 Oktober 2020]. |
| [8] | C. S. Perone, „SourceForge,” 2009. [Online]. Available: http://pyevolve.sourceforge.net/. [Geopend 14 Oktober 2020]. |
| [9] | „Github,” [Online]. Available: https://github.com/DEAP/deap. [Geopend 14 Oktober 2020]. |
| [10] | F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau en C. Gagné, „Read The Docs,” 12 Juli 2012. [Online]. Available: https://deap.readthedocs.io/en/master. [Geopend 14 Oktober 2020]. |
| [11] | K. Leonardo, „Github,” [Online]. Available: https://github.com/SirBob01/NEAT-Python. [Geopend 14 Oktober 2020]. |
| [12] | „Read The Docs,” CodeReclaimers, LLC, [Online]. Available: https://neat-python.readthedocs.io/en/latest/. [Geopend 14 Oktober 2020]. |
| [13] | J. Korstanje, „Towards Data Science,” 1 Juni 2020. [Online]. Available: https://towardsdatascience.com/a-simple-genetic-algorithm-from-scratch-in-python-4e8c66ac3121. [Geopend 13 Oktober 2020]. |