Simple Genetic Algorithm From Scratch in Python

by **Jason Brownlee** on October 12, 2021 in **Optimization** 111

The **genetic algorithm** is a stochastic global optimization algorithm.

It may be one of the most popular and widely known biologically inspired algorithms, along with artificial neural networks.

The algorithm is a type of evolutionary algorithm and performs an optimization procedure inspired by the biological theory of evolution by means of natural selection with a binary representation and simple operators based on genetic recombination and genetic mutations.

In this tutorial, you will discover the genetic algorithm optimization algorithm.

After completing this tutorial, you will know:

- Genetic algorithm is a stochastic optimization algorithm inspired by evolution.
- How to implement the genetic algorithm from scratch in Python.
- How to apply the genetic algorithm to a continuous objective function.

Kick-start your project with my new book Optimization for Machine Learning, including *step-by-step tutorials* and the *Python source code* files for all examples.

Let's get started.

Simple Genetic Algorithm From Scratch in Python
Photo by Magharebia, some rights reserved.

Tutorial Overview

This tutorial is divided into four parts; they are:

1. Genetic Algorithm
2. Genetic Algorithm From Scratch
3. Genetic Algorithm for OneMax
4. Genetic Algorithm for Continuous Function Optimization

Genetic Algorithm

The Genetic Algorithm is a stochastic global search optimization algorithm.

It is inspired by the biological theory of evolution by means of natural selection. Specifically, the new synthesis that combines an understanding of genetics with the theory.

> *Genetic algorithms (algorithm 9.4) borrow inspiration from biological evolution, where fitter individuals are more likely to pass on their genes to the next generation.*

— Page 148, Algorithms for Optimization, 2019.

The algorithm uses analogs of a genetic representation (bitstrings), fitness (function evaluations), genetic recombination (crossover of bitstrings), and mutation (flipping bits).

The algorithm works by first creating a population of a fixed size of random bitstrings. The main loop of the algorithm is repeated for a fixed number of iterations or until no further improvement is seen in the best solution over a given number of iterations.

One iteration of the algorithm is like an evolutionary generation.

First, the population of bitstrings (candidate solutions) are evaluated using the objective function. The objective function evaluation for each candidate solution is taken as the fitness of the solution, which may be minimized or maximized.

Then, parents are selected based on their fitness. A given candidate solution may be used as parent zero or more times. A simple and effective approach to selection involves drawing *k* candidates from the population randomly and selecting the member from the group with the best fitness. This is called tournament selection where *k* is a hyperparameter and set to a value such as 3. This simple approach simulates a more costly fitness-proportionate selection scheme.

> *In tournament selection, each parent is the fittest out of k randomly chosen chromosomes of the population*

— Page 151, Algorithms for Optimization, 2019.

Parents are used as the basis for generating the next generation of candidate points and one parent for each position in the population is required.

Parents are then taken in pairs and used to create two children. Recombination is performed using a crossover operator. This involves selecting a random split point on the bit string, then creating a child with the bits up to the split point from the first parent and from the split point to the end of the string from the second parent. This process is then inverted for the second child.

For example the two parents:

- parent1 = 00000
- parent2 = 11111

May result in two cross-over children:

- child1 = 00011
- child2 = 11100

This is called one point crossover, and there are many other variations of the operator.

Crossover is applied probabilistically for each pair of parents, meaning that in some cases, copies of the parents are taken as the children instead of the recombination operator. Crossover is controlled by a hyperparameter set to a large value, such as 80 percent or 90 percent.

> *Crossover is the Genetic Algorithm's distinguishing feature. It involves mixing and matching parts of two parents to form children. How you do that mixing and matching depends on the representation of the individuals.*

— Page 36, Essentials of Metaheuristics, 2011.

Mutation involves flipping bits in created children candidate solutions. Typically, the mutation rate is set to *1/L*, where *L* is the length of the bitstring.

> *Each bit in a binary-valued chromosome typically has a small probability of being flipped. For a chromosome with m bits, this mutation rate is typically set to 1/m, yielding an average of one mutation per child chromosome.*

— Page 155, Algorithms for Optimization, 2019.

For example, if a problem used a bitstring with 20 bits, then a good default mutation rate would be (1/20) = 0.05 or a probability of 5 percent.

This defines the simple genetic algorithm procedure. It is a large field of study, and there are many extensions to the algorithm.

Now that we are familiar with the simple genetic algorithm procedure, let's look at how we might implement it from scratch.

Genetic Algorithm From Scratch

In this section, we will develop an implementation of the genetic algorithm.

The first step is to create a population of random bitstrings. We could use boolean values *True* and *False*, string values '0' and '1', or integer values 0 and 1. In this case, we will use integer values.

We can generate an array of integer values in a range using the randint() function, and we can specify the range as values starting at 0 and less than 2, e.g. 0 or 1. We will also represent a candidate solution as a list instead of a NumPy array to keep things simple.

An initial population of random bitstring can be created as follows, where "*n_pop*" is a hyperparameter that controls the population size and "*n_bits*" is a hyperparameter that defines the number of bits in a single candidate solution:

```
...
# initial population of random bitstring
pop = [randint(0, 2, n_bits).tolist() for _ in range(n_pop)]
```

Next, we can enumerate over a fixed number of algorithm iterations, in this case, controlled by a hyperparameter named "*n_iter*".

```
...
# enumerate generations
 for gen in range(n_iter):
 ...
```

The first step in the algorithm iteration is to evaluate all candidate solutions.

We will use a function named *objective()* as a generic objective function and call it to get a fitness score, which we will minimize.

```
...
# evaluate all candidates in the population
scores = [objective(c) for c in pop]
```

We can then select parents that will be used to create children.

The tournament selection procedure can be implemented as a function that takes the population and returns one selected parent. The *k* value is fixed at 3 with a default argument, but you can experiment with different values if you like.

```
# tournament selection
def selection(pop, scores, k=3):
 # first random selection
 selection_ix = randint(len(pop))
 for ix in randint(0, len(pop), k-1):
 # check if better (e.g. perform a tournament)
 if scores[ix] < scores[selection_ix]:
 selection_ix = ix
 return pop[selection_ix]
```

We can then call this function one time for each position in the population to create a list of parents.

```
...
# select parents
selected = [selection(pop, scores) for _ in range(n_pop)]
```

We can then create the next generation.

This first requires a function to perform crossover. This function will take two parents and the crossover rate. The crossover rate is a hyperparameter that determines whether crossover is performed or not, and if not, the parents are copied into the next generation. It is a probability and typically has a large value close to 1.0.

The *crossover()* function below implements crossover using a draw of a random number in the range [0,1] to determine if crossover is performed, then selecting a valid split point if crossover is to be performed.

```
# crossover two parents to create two children
def crossover(p1, p2, r_cross):
 # children are copies of parents by default
 c1, c2 = p1.copy(), p2.copy()
 # check for recombination
 if rand() < r_cross:
 # select crossover point that is not on the end of the string
 pt = randint(1, len(p1)-2)
 # perform crossover
 c1 = p1[:pt] + p2[pt:]
 c2 = p2[:pt] + p1[pt:]
 return [c1, c2]
```

We also need a function to perform mutation.

This procedure simply flips bits with a low probability controlled by the "*r_mut*" hyperparameter.

```
# mutation operator
def mutation(bitstring, r_mut):
 for i in range(len(bitstring)):
 # check for a mutation
```

```
 if rand() < r_mut:
 # flip the bit
 bitstring[i] = 1 - bitstring[i]
```

We can then loop over the list of parents and create a list of children to be used as the next generation, calling the crossover and mutation functions as needed.

```
...
# create the next generation
children = list()
for i in range(0, n_pop, 2):
 # get selected parents in pairs
 p1, p2 = selected[i], selected[i+1]
 # crossover and mutation
 for c in crossover(p1, p2, r_cross):
 # mutation
 mutation(c, r_mut)
 # store for next generation
 children.append(c)
```

We can tie all of this together into a function named *genetic_algorithm()* that takes the name of the objective function and the hyperparameters of the search, and returns the best solution found during the search.

```
# genetic algorithm
def genetic_algorithm(objective, n_bits, n_iter, n_pop, r_cross, r_mut):
 # initial population of random bitstring
 pop = [randint(0, 2, n_bits).tolist() for _ in range(n_pop)]
 # keep track of best solution
 best, best_eval = 0, objective(pop[0])
 # enumerate generations
 for gen in range(n_iter):
 # evaluate all candidates in the population
 scores = [objective(c) for c in pop]
 # check for new best solution
 for i in range(n_pop):
 if scores[i] < best_eval:
 best, best_eval = pop[i], scores[i]
 print(">%d, new best f(%s) = %.3f" % (gen,  pop[i], scores[i]))
 # select parents
 selected = [selection(pop, scores) for _ in range(n_pop)]
 # create the next generation
 children = list()
 for i in range(0, n_pop, 2):
 # get selected parents in pairs
 p1, p2 = selected[i], selected[i+1]
 # crossover and mutation
 for c in crossover(p1, p2, r_cross):
 # mutation
 mutation(c, r_mut)
 # store for next generation
 children.append(c)
 # replace population
 pop = children
 return [best, best_eval]
```

Now that we have developed an implementation of the genetic algorithm, let's explore how we might apply it to an objective function.

Genetic Algorithm for OneMax

In this section, we will apply the genetic algorithm to a binary string-based optimization problem.

The problem is called OneMax and evaluates a binary string based on the number of 1s in the string. For example, a bitstring with a length of 20 bits will have a score of 20 for a string of all 1s.

Given we have implemented the genetic algorithm to minimize the objective function, we can add a negative sign to this evaluation so that large positive values become large negative values.

The *onemax()* function below implements this and takes a bitstring of integer values as input and returns the negative sum of the values.

```
# objective function
def onemax(x):
 return -sum(x)
```

Next, we can configure the search.

The search will run for 100 iterations and we will use 20 bits in our candidate solutions, meaning the optimal fitness will be -20.0.

The population size will be 100, and we will use a crossover rate of 90 percent and a mutation rate of 5 percent. This configuration was chosen after a little trial and error.

```
...
# define the total iterations
n_iter = 100
# bits
n_bits = 20
# define the population size
n_pop = 100
# crossover rate
r_cross = 0.9
# mutation rate
r_mut = 1.0 / float(n_bits)
```

The search can then be called and the best result reported.

```
...
# perform the genetic algorithm search
```

```
best, score = genetic_algorithm(onemax, n_bits, n_iter, n_pop, r_cross, r_mut)
print('Done!')
print('f(%s) = %f' % (best, score))
```

Tying this together, the complete example of applying the genetic algorithm to the OneMax objective function is listed below.

```
# genetic algorithm search of the one max optimization problem
from numpy.random import randint
from numpy.random import rand

# objective function
def onemax(x):
 return -sum(x)

# tournament selection
def selection(pop, scores, k=3):
 # first random selection
 selection_ix = randint(len(pop))
 for ix in randint(0, len(pop), k-1):
 # check if better (e.g. perform a tournament)
 if scores[ix] < scores[selection_ix]:
 selection_ix = ix
 return pop[selection_ix]

# crossover two parents to create two children
def crossover(p1, p2, r_cross):
 # children are copies of parents by default
 c1, c2 = p1.copy(), p2.copy()
 # check for recombination
 if rand() < r_cross:
 # select crossover point that is not on the end of the string
 pt = randint(1, len(p1)-2)
 # perform crossover
 c1 = p1[:pt] + p2[pt:]
 c2 = p2[:pt] + p1[pt:]
 return [c1, c2]

# mutation operator
def mutation(bitstring, r_mut):
 for i in range(len(bitstring)):
 # check for a mutation
 if rand() < r_mut:
 # flip the bit
 bitstring[i] = 1 - bitstring[i]

# genetic algorithm
def genetic_algorithm(objective, n_bits, n_iter, n_pop, r_cross, r_mut):
 # initial population of random bitstring
 pop = [randint(0, 2, n_bits).tolist() for _ in range(n_pop)]
 # keep track of best solution
 best, best_eval = 0, objective(pop[0])
 # enumerate generations
 for gen in range(n_iter):
 # evaluate all candidates in the population
 scores = [objective(c) for c in pop]
 # check for new best solution
 for i in range(n_pop):
 if scores[i] < best_eval:
 best, best_eval = pop[i], scores[i]
 print(">%d, new best f(%s) = %.3f" % (gen,  pop[i], scores[i]))
 # select parents
 selected = [selection(pop, scores) for _ in range(n_pop)]
 # create the next generation
```

```
 children = list()
 for i in range(0, n_pop, 2):
 # get selected parents in pairs
 p1, p2 = selected[i], selected[i+1]
 # crossover and mutation
 for c in crossover(p1, p2, r_cross):
 # mutation
 mutation(c, r_mut)
 # store for next generation
 children.append(c)
 # replace population
 pop = children
 return [best, best_eval]

# define the total iterations
n_iter = 100
# bits
n_bits = 20
# define the population size
n_pop = 100
# crossover rate
r_cross = 0.9
# mutation rate
r_mut = 1.0 / float(n_bits)
# perform the genetic algorithm search
best, score = genetic_algorithm(onemax, n_bits, n_iter, n_pop, r_cross, r_mut)
print('Done!')
print('f(%s) = %f' % (best, score))
```

Running the example will report the best result as it is found along the way, then the final best solution at the end of the search, which we would expect to be the optimal solution.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

In this case, we can see that the search found the optimal solution after about eight generations.

```
>0, new best f([1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1]) = -14.000
>0, new best f([1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0]) = -15.000
>1, new best f([1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1]) = -16.000
>2, new best f([0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1]) = -17.000
>2, new best f([1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -19.000
>8, new best f([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -20.000
Done!
f([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -20.000000
```

Genetic Algorithm for Continuous Function Optimization

Optimizing the OneMax function is not very interesting; we are more likely to want to optimize a continuous function.

For example, we can define the x^2 minimization function that takes input variables and has an optima at f(0, 0) = 0.0.

```
# objective function
def objective(x):
 return x[0]**2.0 + x[1]**2.0
```

We can minimize this function with a genetic algorithm.

First, we must define the bounds of each input variable.

```
...
# define range for input
bounds = [[-5.0, 5.0], [-5.0, 5.0]]
```

We will take the "*n_bits*" hyperparameter as a number of bits per input variable to the objective function and set it to 16 bits.

```
...
# bits per variable
n_bits = 16
```

This means our actual bit string will have (16 * 2) = 32 bits, given the two input variables.

We must update our mutation rate accordingly.

```
...
# mutation rate
r_mut = 1.0 / (float(n_bits) * len(bounds))
```

Next, we need to ensure that the initial population creates random bitstrings that are large enough.

```
...
# initial population of random bitstring
pop = [randint(0, 2, n_bits*len(bounds)).tolist() for _ in range(n_pop)]
```

Finally, we need to decode the bitstrings to numbers prior to evaluating each with the objective function.

We can achieve this by first decoding each substring to an integer, then scaling the integer to the desired range. This will give a vector of values in the range that can then be provided to the objective function for evaluation.

The *decode()* function below implements this, taking the bounds of the function, the number of bits per variable, and a bitstring as input and returns a list of decoded real values.

```
# decode bitstring to numbers
def decode(bounds, n_bits, bitstring):
 decoded = list()
 largest = 2**n_bits
 for i in range(len(bounds)):
 # extract the substring
 start, end = i * n_bits, (i * n_bits)+n_bits
 substring = bitstring[start:end]
 # convert bitstring to a string of chars
 chars = ''.join([str(s) for s in substring])
 # convert string to integer
 integer = int(chars, 2)
 # scale integer to desired range
 value = bounds[i][0] + (integer/largest) * (bounds[i][1] - bounds[i][0])
 # store
 decoded.append(value)
 return decoded
```

We can then call this at the beginning of the algorithm loop to decode the population, then evaluate the decoded version of the population.

```
...
# decode population
decoded = [decode(bounds, n_bits, p) for p in pop]
# evaluate all candidates in the population
scores = [objective(d) for d in decoded]
```

Tying this together, the complete example of the genetic algorithm for continuous function optimization is listed below.

```
# genetic algorithm search for continuous function optimization
from numpy.random import randint
from numpy.random import rand

# objective function
def objective(x):
 return x[0]**2.0 + x[1]**2.0

# decode bitstring to numbers
def decode(bounds, n_bits, bitstring):
 decoded = list()
 largest = 2**n_bits
 for i in range(len(bounds)):
 # extract the substring
 start, end = i * n_bits, (i * n_bits)+n_bits
 substring = bitstring[start:end]
 # convert bitstring to a string of chars
 chars = ''.join([str(s) for s in substring])
 # convert string to integer
 integer = int(chars, 2)
 # scale integer to desired range
 value = bounds[i][0] + (integer/largest) * (bounds[i][1] - bounds[i][0])
 # store
 decoded.append(value)
 return decoded

# tournament selection
def selection(pop, scores, k=3):
 # first random selection
 selection_ix = randint(len(pop))
 for ix in randint(0, len(pop), k-1):
 # check if better (e.g. perform a tournament)
 if scores[ix] < scores[selection_ix]:
```

```
 selection_ix = ix
 return pop[selection_ix]

# crossover two parents to create two children
def crossover(p1, p2, r_cross):
 # children are copies of parents by default
 c1, c2 = p1.copy(), p2.copy()
 # check for recombination
 if rand() < r_cross:
 # select crossover point that is not on the end of the string
 pt = randint(1, len(p1)-2)
 # perform crossover
 c1 = p1[:pt] + p2[pt:]
 c2 = p2[:pt] + p1[pt:]
 return [c1, c2]

# mutation operator
def mutation(bitstring, r_mut):
 for i in range(len(bitstring)):
 # check for a mutation
 if rand() < r_mut:
 # flip the bit
 bitstring[i] = 1 - bitstring[i]

# genetic algorithm
def genetic_algorithm(objective, bounds, n_bits, n_iter, n_pop, r_cross, r_mut):
 # initial population of random bitstring
 pop = [randint(0, 2, n_bits*len(bounds)).tolist() for _ in range(n_pop)]
 # keep track of best solution
 best, best_eval = 0, objective(decode(bounds, n_bits, pop[0]))
 # enumerate generations
 for gen in range(n_iter):
 # decode population
 decoded = [decode(bounds, n_bits, p) for p in pop]
 # evaluate all candidates in the population
 scores = [objective(d) for d in decoded]
 # check for new best solution
 for i in range(n_pop):
 if scores[i] < best_eval:
 best, best_eval = pop[i], scores[i]
 print(">%d, new best f(%s) = %f" % (gen,  decoded[i], scores[i]))
 # select parents
 selected = [selection(pop, scores) for _ in range(n_pop)]
 # create the next generation
 children = list()
 for i in range(0, n_pop, 2):
 # get selected parents in pairs
 p1, p2 = selected[i], selected[i+1]
 # crossover and mutation
 for c in crossover(p1, p2, r_cross):
 # mutation
 mutation(c, r_mut)
 # store for next generation
 children.append(c)
 # replace population
 pop = children
 return [best, best_eval]

# define range for input
bounds = [[-5.0, 5.0], [-5.0, 5.0]]
# define the total iterations
n_iter = 100
# bits per variable
n_bits = 16
# define the population size
```

```
# define the total iterations
n_iter = 100
# bits
n_bits = 20
# define the population size
n_pop = 100
# crossover rate
r_cross = 0.9
# mutation rate
r_mut = 1.0 / (float(n_bits) * len(bounds))
# perform the genetic algorithm search
best, score = genetic_algorithm(objective, bounds, n_bits, n_iter, n_pop, r_cross, r_mut)
print('Done!')
decoded = decode(bounds, n_bits, best)
print('f(%s) = %f' % (decoded, score))
```

Running the example reports the best decoded results along the way and the best decoded solution at the end of the run.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

In this case, we can see that the algorithm discovers an input very close to f(0.0, 0.0) = 0.0.

```
>0, new best f([-0.785064697265625, -0.807647705078125]) = 1.268621
>0, new best f([0.385894775390625, 0.342864990234375]) = 0.266471
>1, new best f([-0.342559814453125, -0.1068115234375]) = 0.128756
>2, new best f([-0.038909912109375, 0.30242919921875]) = 0.092977
>2, new best f([0.145721435546875, 0.1849365234375]) = 0.055436
>3, new best f([0.14404296875, -0.029754638671875]) = 0.021634
>5, new best f([0.066680908203125, 0.096435546875]) = 0.013746
>5, new best f([-0.036468505859375, -0.10711669921875]) = 0.012804
>6, new best f([-0.038909912109375, -0.099639892578125]) = 0.011442
>7, new best f([-0.033111572265625, 0.09674072265625]) = 0.010455
>7, new best f([-0.036468505859375, 0.05584716796875]) = 0.004449
>10, new best f([0.058746337890625, 0.008087158203125]) = 0.003517
>10, new best f([-0.031585693359375, 0.008087158203125]) = 0.001063
>12, new best f([0.022125244140625, 0.008087158203125]) = 0.000555
>13, new best f([0.022125244140625, 0.00701904296875]) = 0.000539
>13, new best f([-0.013885498046875, 0.008087158203125]) = 0.000258
>16, new best f([-0.011444091796875, 0.00518798828125]) = 0.000158
>17, new best f([-0.0115966796875, 0.00091552734375]) = 0.000135
>17, new best f([-0.004730224609375, 0.00335693359375]) = 0.000034
>20, new best f([-0.004425048828125, 0.00274658203125]) = 0.000027
>21, new best f([-0.002288818359375, 0.00091552734375]) = 0.000006
>22, new best f([-0.001983642578125, 0.00091552734375]) = 0.000005
>22, new best f([-0.001983642578125, 0.0006103515625]) = 0.000004
>24, new best f([-0.001373291015625, 0.001068115234375]) = 0.000003
>25, new best f([-0.001373291015625, 0.00091552734375]) = 0.000003
>26, new best f([-0.001373291015625, 0.0006103515625]) = 0.000002
>27, new best f([-0.001068115234375, 0.0006103515625]) = 0.000002
>29, new best f([-0.000152587890625, 0.00091552734375]) = 0.000001
>33, new best f([-0.0006103515625, 0.0]) = 0.000000
>34, new best f([-0.000152587890625, 0.00030517578125]) = 0.000000
>43, new best f([-0.00030517578125, 0.0]) = 0.000000
>60, new best f([-0.000152587890625, 0.000152587890625]) = 0.000000
>65, new best f([-0.000152587890625, 0.0]) = 0.000000
Done!
f([-0.000152587890625, 0.0]) = 0.000000
```

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

Books

- Genetic Algorithms in Search, Optimization, and Machine Learning, 1989.
- An Introduction to Genetic Algorithms, 1998.
- Algorithms for Optimization, 2019.
- Essentials of Metaheuristics, 2011.
- Computational Intelligence: An Introduction, 2007.

API

- numpy.random.randint API.

Articles

- Genetic algorithm, Wikipedia.
- Genetic algorithms, Scholarpedia.

Summary

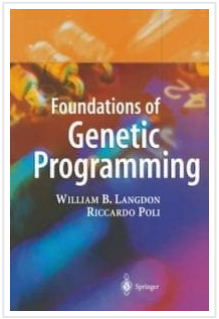
In this tutorial, you discovered the genetic algorithm optimization.

Specifically, you learned:

- Genetic algorithm is a stochastic optimization algorithm inspired by evolution.
- How to implement the genetic algorithm from scratch in Python.
- How to apply the genetic algorithm to a continuous objective function.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

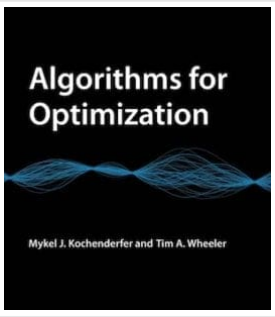
More On This Topic



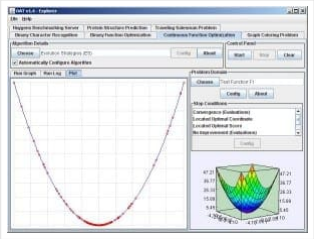
Books on Genetic Programming

Differential Evolution Global Optimization With Python

Understand Machine Learning Algorithms By…



3 Books on Optimization for Machine Learning



How I Got Started In Machine Learning

Why Optimization Is Important in Machine Learning

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.

View all posts by Jason Brownlee →

‹ Differential Evolution Global Optimization With Python

How to Update Neural Network Models With More Data ›

111 Responses to *Simple Genetic Algorithm From Scratch in Python*

Satish Chhatpar March 4, 2021 at 2:33 am #

REPLY

I did not understand above algorithm. Its complex

Jason Brownlee March 4, 2021 at 5:51 am #

REPLY

Sorry to hear that.

Which part was confusing?

Naadiya March 30, 2021 at 7:24 pm #

REPLY

I got all the idea of how it works but i have a querry can you please tell me which problem you are trying to solve?

Jason Brownlee March 31, 2021 at 6:00 am #

REPLY

We are solving two different optimization problems as described in the tutorial.

Mohammed Faizan May 27, 2021 at 12:35 am #

REPLY

Sir can i use these code for my final year project because my team is working on genetic algorithm.

Jason Brownlee May 27, 2021 at 5:39 am #

REPLY

This is a common request that I answer here:
https://machinelearningmastery.com/faq/single-faq/can-i-use-your-code-in-my-own-project

Ahmad Qadeib Alban August 14, 2022 at 4:44 am #

REPLY

I believe you have to pass number through the code step by step then you can get it. that works for me 🙂

Wilfredo Yeguez March 4, 2021 at 9:06 am #

REPLY

Thanks Jason! You gave me a good push.

Jason Brownlee March 4, 2021 at 1:08 pm #

REPLY

You're welcome.

Ankita March 5, 2021 at 4:23 am #

REPLY

Thankyou so much. Very helpful content for me as i am doing Ph.D in Genetic Algorithm. Could you please help me more. I need some help in further implementation. Mail me as soon as possible.

REPLY

Jason Brownlee March 5, 2021 at 5:35 am #

I don't have the capacity to help you with your research project, sorry.

REPLY

Satya September 14, 2021 at 5:42 pm #

Hii, I'm doing master's Thesis in the combination of Genetic algorithm with dynamic programming of solving travelling salesman problem.
I have doubts regarding implemenation part. THe reason why i'm asking for you is doing you Ph.D
I hope you will help me
thank you

REPLY

Peter March 5, 2021 at 7:41 am #

Awesome article, quite large though excellent example to learn from. Thank you

REPLY

Jason Brownlee March 5, 2021 at 8:16 am #

Thanks!

REPLY

Indira X January 1, 2022 at 7:50 pm #

Hi Jason,
Thanks for sharing your knowledge and code. I used it and it worked very well.
I have one question: How do you know it's a global optimum solutution?

REPLY

James Carmichael January 2, 2022 at 9:04 am #

Hi Indira,

What code in particular did you use? Keep in mind that for some of the simple examples presented, we already knew the global optimum solution.

Regards,

Indira X January 3, 2022 at 2:42 pm #

I used the code in the last example of this post but I changed it to multi-objective.
Obj1=abs(x[0]*a0 + x[1]*a1 + x[2]*a2 – a_target)**2
Obj2=abs(x[0] + x[1] + x[2])
I have also some constraints for x.
As I said, the code works fine and the solutions are really reasonable. However, I want to know if I can prove somehow this is the global optima, or even a local optima.

John Lee March 5, 2021 at 12:58 pm #

REPLY

Awesome lesson. Thanks!

Jason Brownlee March 5, 2021 at 1:39 pm #

REPLY

You're welcome!

huibin fu March 5, 2021 at 6:31 pm #

REPLY

can i copy the code to my Python? because I want to practice it

Jason Brownlee March 6, 2021 at 5:14 am #

REPLY

Yes.

Mojtaba March 5, 2021 at 9:46 pm #

REPLY

Hi dear Jason.
Thanks for this helpful tutorial.
May you give a tutorial on feature selection using genetic algorithms?

Jason Brownlee March 6, 2021 at 5:17 am #

REPLY

You're welcome!

Thanks for the suggestion!

Paul Winter March 13, 2021 at 8:03 am #

REPLY

Hi Jason, thanks for the great tutorial. I enjoyed reading and typing the code is step by step to really follow along and understand it.

I modified my genetic_algorithm to also have a decode and bounds input parameter to be able to reuse for both examples. I added a decode for oneup that just teturns the input value, and changed your decode so that the bitstring can be decoded into multiple params of same no of bits.

Hopefully thats the right direction for reuse.

I did spot a bug in decode. largest sholuld be (2**n_bits) -1

Jason Brownlee March 14, 2021 at 5:21 am #

REPLY

You're welcome!

Well done!

Noted.

Junaid Zaheer March 18, 2021 at 4:18 am #

REPLY

Hi dear very much difficult to understand such an important topic like genetic algorithm..Any how lots of thank yous to have some light on it..

Jason Brownlee March 18, 2021 at 5:24 am #

REPLY

You're welcome.

Arnav Das March 24, 2021 at 6:03 am #

REPLY

super cool article jason sir, and really really appreciate for putting everything in code, will help us all a lot in experimenting here and there.

was just wondering something about these algorithms, would be it fair to say as loss functions are to gradient descent do objective functions also serve the same purpose for genetic algorithms ?

And compared to genetic algorithms aren't gradient descent algorithm more objective based, I mean they are solely guided to find the best spot to stop while genetic algorithms more or less rely more on mutations and crossover to reach the same.

Jason Brownlee March 25, 2021 at 4:34 am #

REPLY

Thanks!

Yes, sure. Loss function is an objective function for the gradient descent optimizaiton algorithm.

No, they are just different algorithms. GD uses more information (e.g. derivatives) whereas GAs do not.

JG March 31, 2021 at 6:03 am #

REPLY

Hi Jason,

A great code and introduction to Genetic Algorithms (GA), as a beautiful alternative to Artificial Neural Networks (ANN). Congrats for this post!.

I am pleasantly surprised about how GA get quick convergence to the minimum quadratic function !.

In my opinion the main differences between ANN vs GA are:

with ANN we "map" output vs input with a dataset and a neural model that learn the weights vs GA which solving a "min/max" optimum problem, via "Artificial Gene Selection". That is, coding "genes" problems in bits > initiating a population > selecting parents via an objective function that evaluated better adaptation to the target > performing Crossover genes > mutation genes > replacing parent population for children population every generation.
So the key issue is coding the problems variables in bits, to be able to apply crossover and bits mutation methods, plus selecting parents via the better objetive performance.

– I intuit some "probabilistic" convergence pillars supporting this "Artificial Selection" (or GA) for optimum issues solving vs some SGD and backpropagation methodology (minimum error) as pillars supporting ANN.

I experiment with other objetive functions such cubic functions, etc. and in all of them the code performing pretty well founded the minimum value very quickly.

My only concern in terms of "artificial selection" methodology is, of course at least one individual member of the population, get very quickly the minimum searched, but the rest of population (even changing mutation and crossover rate) remain outside this optimum "gene" value, even if I play with different population number, number of generations, mutation and crossover rates, etc…

so finally we are not able to evolve completely the old population into a "new specie" population, at leat with this chunk of algorithm, but nature can evolve naturally producing new species from old ones :-))

Thank you for inspiring all of this beautiful issues!
Regards

Jason Brownlee March 31, 2021 at 6:09 am #

REPLY

Thanks!

Yes, I like to think of it as two techniques for solving very different problem types: “function approximation” vs “function optimization”.

Be careful, tuning GAs can be addictive 🙂

Yessense April 2, 2021 at 10:16 pm #

REPLY

There is an error in 63th string:
>> best, best_eval = 0, objective(pop[0])
Should be:

best, best_eval = 0, objective(decode(bounds, n_bits, pop[0])

Jason Brownlee April 3, 2021 at 5:32 am #

REPLY

Agreed!

Thanks, fixed.

Chris May 24, 2021 at 9:15 pm #

REPLY

…and on line 63 a missing parenthesis at the end.

Libo April 18, 2021 at 2:37 pm #

REPLY

Hi Jason, Very nice tutorial like all your other tutorials. I have a question, for each new generation, isn’t that we should keep all parents from last generation, plus the current children generation, sorted them according to the scores, keep the top max or min scores? The reason is because that some of the parents are better than child, therefore we want to keep the top performers? Thanks

Jason Brownlee April 19, 2021 at 5:48 am #

REPLY

Thanks!

There are many modifications of the algorithm you can make, including keeping the best parents from the previous generation. This is called elitism.

Mark April 20, 2021 at 7:07 pm #

REPLY

Hi Jason, can you please make a similar tutorial about Genetic programming, or you can just tell me where the algorithm will have to change to be a genetic programming algorithm not GA

Jason Brownlee April 21, 2021 at 5:54 am #

REPLY

Thanks for the suggestion.

Mohamed April 27, 2021 at 9:57 am #

REPLY

Hi, in line 52 at onmax objective function:
should it be like:

if score[i] > best_eval:
best, best_eval = pop[i], scores[i]

Jason Brownlee April 28, 2021 at 5:57 am #

REPLY

We have inverted the one max objective function to make it minimizing.

Oliver May 1, 2021 at 10:25 pm #

REPLY

Thanks jason, I've been a long time reader here and I think I'm using your textbook on GA for a class project?

Jason Brownlee May 2, 2021 at 5:33 am #

REPLY

Sure, this will help:
https://machinelearningmastery.com/faq/single-faq/can-i-use-your-code-in-my-own-project

john May 4, 2021 at 12:37 pm #

REPLY

hello,, i have syntax error here why?

for gen in range(n_iter):

^

SyntaxError: invalid syntax

REPLY

Jason Brownlee May 5, 2021 at 6:06 am #

Sorry to hear that, perhaps some of these tips will help:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

REPLY

Hamada May 7, 2021 at 8:55 am #

What is the basis for selecting the values of cross_over and mutation rates ?

REPLY

Jason Brownlee May 8, 2021 at 6:28 am #

Trial and error, or using values that have historically worked well on other problems.

REPLY

john May 8, 2021 at 11:10 pm #

what is the rastrigins function in python?
how i cam implement it in python

REPLY

Jason Brownlee May 9, 2021 at 5:56 am #

Sorry, I don’t think I have an example.

REPLY

agelos May 9, 2021 at 2:31 am #

File “”, line 65
for gen in range(n_iter):
^
SyntaxError: invalid syntax

in code for continuous function simply copied pasted this error comes up jason

Jason Brownlee May 9, 2021 at 5:57 am #

REPLY

This will help you copy the code without error:
https://machinelearningmastery.com/faq/single-faq/how-do-i-copy-code-from-a-tutorial

Guilherme May 16, 2021 at 5:39 am #

REPLY

Very nice article!

I'm starting to read about Neural Networks and stumbled upon this page while searching for Genetic Algorithms on Google. It helped me understand some basic concepts.

Thank you!

Jason Brownlee May 17, 2021 at 5:34 am #

REPLY

You're welcome!

Muruganandan S May 17, 2021 at 3:31 am #

REPLY

Dear Jason

Your article is very nice. But, I am not able to go line by line understanding as I am new to the GA. But I got some useful inputs to my work related to stock price predictions. However, I have lots of doubts regrading the implementation of GA in price predictions. Can you help me in this area.

Jason Brownlee May 17, 2021 at 5:39 am #

REPLY

Thanks!

If you are interested in time series forecasting, perhaps start here:
https://machinelearningmastery.com/start-here/#timeseries

RAHEEL SHAIKH May 25, 2021 at 3:09 pm #

REPLY

There is a bug in the code.

best, best_eval = 0, objective(decode(bounds, n_bits, pop[0])

should be

best, best_eval = 0, objective(decode(bounds, n_bits, pop[0]))
i.e. with last bracket. That is why many people having syntax error.

Thanks

REPLY

Jason Brownlee May 26, 2021 at 5:49 am #

Thanks, fixed.

REPLY

Yara May 29, 2021 at 10:23 am #

Hi! First of all, thanks for the tutorial. I'm currently working on an adaptation for a function that depends on 4 variables and having trouble with the decoding function. Is the following right?

```
def decode(bounds, n_bits, bitstring):
decoded = list()
largest = 4**n_bits-1
for i in range(len(bounds)):
# extract the substring
start, end = i * n_bits, (i * n_bits)+n_bits
substring = bitstring[start:end]
# convert bitstring to a string of chars
chars = ''.join([str(s) for s in substring])
# convert string to integer
integer = int(chars, 4)
# scale integer to desired range
value = bounds[i][0] + (integer/largest) * (bounds[i][1] – bounds[i][0])
# store
decoded.append(value)
return decoded
```

REPLY

Jason Brownlee May 30, 2021 at 5:47 am #

You're welcome.

Sorry, I don't have the capacity to review/debug your extensions. I hope you can understand.

REPLY

Guixin Liu June 12, 2021 at 8:03 am #

This is very clear and instructive. I used to study Matlab codes for GA but feel it very difficult. Now I realized it's not that the algorithm is hard itself, but that the codes I read before was not well written.

Thanks!

Jason Brownlee June 13, 2021 at 5:46 am #

REPLY

You're very welcome!

Jianhua June 12, 2021 at 3:43 pm #

REPLY

Hi Jason! Thank you for making this tutorial. I was wondering if it is possible to plot the convergence for your genetic algorithm? If so, how would you implement it?

Jason Brownlee June 13, 2021 at 5:47 am #

REPLY

Yes, you could save the best fitness in a list each iteration, then plot the list at the end of the run.

Mariona July 6, 2021 at 4:31 am #

REPLY

Hi Jason,
Thank you for sharing this 🙂 I am trying to apply this for a problem with both integer & continuous variables. Any tips on how to do this? I was thinking, in the decode function, only some of the values should be decoded to continuous, the rest should stay as binary or integer.

Jason Brownlee July 6, 2021 at 5:50 am #

REPLY

Perhaps first decide all to bits to integers, then covert some integers to floats in the required range.

ali July 14, 2021 at 8:04 am #

REPLY

thanks for this title
i have a question , I have some data from a function Can I predict what the actual function is ? use GP

Jason Brownlee July 15, 2021 at 5:22 am #

REPLY

You can approximate a function that matches the data. This is the goal of applied machine learning (function approximation).

Indi September 2, 2021 at 3:12 pm #

REPLY

Thanks a lot Jason!

Just a couple of notes if someone wants to use python 2.x:
1) For the OneMax example, replace c1, c2 = p1.copy(), p2.copy() by c1, c2 = p1[:], p2[:]
2) For the continuous function, write at the beginning of the code the following
from __future__ import division

Great job Jason!

Jason Brownlee September 3, 2021 at 5:28 am #

REPLY

Thanks for sharing!

Gabrielle November 3, 2021 at 3:08 am #

REPLY

You're an angel!
Thank you very much <3

tahir November 28, 2021 at 1:13 am #

REPLY

hello, I need help with my homework. I need to get the best 20 children in a population of 20 individuals with onemax. How should I change the onemax code you provided? i am new to this stuff.

Adrian Tam November 29, 2021 at 8:50 am #

REPLY

Get 20 out of 20: Isn't that means to pick everyone?

Kokot December 2, 2021 at 3:42 am #

REPLY

Hello, I have a question about maximizing a function:
Do we have to change only

if scores[i] best_eval:
?

Adrian Tam December 8, 2021 at 5:46 am #

REPLY

Yes. In that case, you remember the largest value you ever saw

Tolulope Babatunde December 9, 2021 at 1:51 am #

REPLY

Hello,

Thank you for this. I am trying to use genetic algorithm to solve a weighted set covering problem. The data looks like this;

Universe = set([1.,2.,3.,4.,5.,6.,7.,8.])
Subsets = [set([1.,2.]),
set([3.,4.]),
set([5.,6.]),
set([7.,8.]),
set([2.,4.,6.,8.])]

weights = [1.,1.,1.,1.,1.]

How do I define the objective function ?

Adrian Tam December 10, 2021 at 4:15 am #

REPLY

Maybe the amount of overlap?

Yousaf Ali December 28, 2021 at 4:28 pm #

REPLY

Objective() need any library to import ?

scores = [Objective(c) for c in pop]
NameError: name ‘Objective’ is not defined

James Carmichael December 29, 2021 at 11:42 am #

REPLY

Hi Yousaf…Please provide a full code listing so that we may determine what may be required.

Regards,

Indira X January 6, 2022 at 10:28 pm #

REPLY

Hi,
I hope somebody can help me, please.

I used the code in the last example of this post but I changed it to multi-objective.
Obj1=abs(x[0]*a0 + x[1]*a1 + x[2]*a2 – a_target)**2
Obj2=abs(x[0] + x[1] + x[2])
I have also some constraints for x.
The code works fine and the solutions are really reasonable. However, I want to know if I can prove somehow this is the global optima, or even a local optima. I wonder if this is in some of the books posted here.
Thanks.

Jack February 11, 2022 at 1:44 am #

REPLY

How can we use GA to find the optimal stringing for solar systems can you make an example of that if possible

James Carmichael February 11, 2022 at 8:24 am #

REPLY

Hi Jack…Thank you for the question! While I do not have capacity to address your specific application, I would be more than happy to help answer any specific questions you have regarding our materials.

Milad March 10, 2022 at 8:48 pm #

REPLY

Hi
I want to use an LSTM network for the objective with 10 variables and the n_step=21 (I mean from t-21 to t) which all of these 10 variables have their own bound… I have some problems with the decoder and the input dimension of the LSTM network… Any kind of help will be appreciated

James Carmichael March 11, 2022 at 1:07 pm #

REPLY

Hi Milad…the following is a great starting point:

https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/

Milad March 11, 2022 at 5:46 pm #

REPLY

Hi

Thanks for your response

I have read all of the machinelearningmastery's articles about implementing different kinds of machine learning methods in python. (Big fan of your website) But I couldn't find a solution…

Silviu March 18, 2022 at 6:55 pm #

REPLY

The very first line of the code
pop = [randint(0, 2, n_bits).tolist() for _ in range(n_pop)]

does not work

Am i right?

James Carmichael March 20, 2022 at 7:22 am #

REPLY

Hi Silviu…Can you provide the exact error message you encountered so that we may better assist you?

Lupus Solitarius April 18, 2022 at 9:31 am #

REPLY

Easy to follow, I am adept with python, but just learning GA now. The code is full of errors and omissions when copied from the web page; I was able to fix.

In particular, the MUTATION function lacks the RETURN statement. I also determined the GA works BETTER without mutation!

James Carmichael April 19, 2022 at 7:15 am #

REPLY

Thank you for the feedback Lupus! Let us know if you have any questions we can help you with.

Iman May 3, 2022 at 8:54 am #

REPLY

Hi Jason,
Why did you use “binary number” in continuous function optimization?
Can we use real number directly into the genetic algorithm function?

Thanks,
Iman

REPLY

James Carmichael May 3, 2022 at 11:09 pm #

Hi Iman..You may find the following of interest:

https://pubs.acs.org/doi/pdf/10.1021/jp063998e

REPLY

Boutine May 18, 2022 at 7:42 pm #

Hi James, Thank you for the great explanation. Can you send me the full code you used in this article?

REPLY

James Carmichael May 19, 2022 at 6:26 am #

Hi Boutine…You are very welcome! The full code listing is found below the following text in the article:

“Tying this together, the complete example of the genetic algorithm for continuous function optimization is listed below.”

REPLY

matt August 18, 2022 at 8:09 pm #

Hi Jason,

My name is Matt.
Thanks for the tutorial.

I have trained a GBM model which is able to predict three target variables. Now, I am gonna integrate the GBM model with GA to find the optimum set of predictor variables that minimize the target variables.
I have searched the web, however, I was not successful in finding a resource or tutorial.
Would you be able to assist?

Thanks heaps

REPLY

James Carmichael August 19, 2022 at 7:31 am #

Hi Matt…the following resource may be of interest to you:

https://towardsdatascience.com/genetic-algorithm-to-optimize-machine-learning-hyperparameters-72bd6e2596fc

REPLY

Matt August 19, 2022 at 10:39 am #

Thanks James for sharing the resource.
But What I am after is not using GA for hyperparameter tuning.
I am going to use the GA for finding an optimum set of “predictor variables” that I have used for training a GBM model. The set of predictor variables by which the defined target variables are getting minimized.

Does that make sense?

Thanks in advance.

REPLY

Matt August 19, 2022 at 10:20 am #

Thanks James for sharing the resource.
But What I am after is not using GA for hyperparameter tuning.
I am going to use the GA for finding an optimum set of “predictor variables” that I have used for training a GBM model. The set of predictor variables by which the defined target variables are getting minimized.

Does that make sense?

Thanks in advance.

REPLY

Aamir Aman September 13, 2022 at 4:24 am #

Hi James, dear can you answer me is it possible that we get gif animation of GA algorithm the same like you did in PSO algorithm?

REPLY

James Carmichael September 13, 2022 at 7:37 am #

Hi Amir…The following may be of interest to you:

https://ieeexplore.ieee.org/document/6016133

REPLY

Matt September 22, 2022 at 2:23 pm #

Hi James,
What if there is a constraint for our output? How could we add that constraint to your code? Can you advise?

James Carmichael September 23, 2022 at 5:55 am #

REPLY

Hi Matt…Please see my previous comment.

matt September 22, 2022 at 8:16 pm #

REPLY

Hi James,
How can I add constraint for inputs to this code?

Thanks

James Carmichael September 23, 2022 at 5:52 am #

REPLY

Hi Matt…Please clarify the goals and intention of constraints so that we may better assist you.

Matt September 24, 2022 at 8:36 am #

REPLY

Hi James,
Sure, the constraint for the problem that I am trying to solve is budget constraint. The inputs are construction material. So the constraint that I would like to add is when a population is selected by GA code, its cost implication is checked and if it is below e.g $10,000, it can go to other steps of GA. Otherwise, the population needs to be change until the budget constraint is met.

Please let me know if it requires further elaboration.

Thanks

Matt September 29, 2022 at 7:54 pm #

REPLY

Hi James,
Sure, the constraint for the problem that I am trying to solve is budget constraint. The inputs are construction material. So the constraint that I would like to add is when a population is selected by GA code, its cost implication is checked and if it is below e.g $10,000, it can go to other steps of GA. Otherwise, the population needs to be change until the budget constraint is met.

Please let me know if it requires further elaboration.

Thanks

Matty October 3, 2022 at 11:35 am #

REPLY

Hi James,
Can GA be used for discrete optimization?
If so, do you have any reference explaining how to do that?

Thanks

James Carmichael October 4, 2022 at 7:06 am #

REPLY

Hi Matty…You may find the following of interest:

https://www.researchgate.net/publication/2404185_Genetic_Algortihms_For_Mixed_DiscreteContinuous_Optimization_In_Multidisciplinary_Design

Shenglin Li April 14, 2023 at 6:36 am #

REPLY

Hi James,

It could repeatedly select the same person from all candidates ‘selected = [selection(pop, scores) for _ in range(n_pop)]’. For example, selected = [pop[2], pop[3], pop[2]…], so p1 and p2 could be pop[2] and pop[2], The person and the person’s copy can’t have children in the real world.

Shenglin Li April 14, 2023 at 7:01 am #

REPLY

Hi James,

It could repeatedly select the same person in ‘selected = [selection(pop, scores) for _ in range(n_pop)]’, for example, ‘selected = [pop[3], pop[0], pop[3], …], so the parents, p1 and p2, could be pop[3] and pop[3]. The person can the person’s copy can’t have children in the real world.

Muhammad Ruma June 2, 2023 at 6:04 am #

REPLY

Hey, pls I can get recommender system genetic algorithm source codes from you? Thanks.

James Carmichael June 2, 2023 at 10:03 am #

REPLY

Hi Muhammad…In general we do not provide source code as a service. We do include source code for each of our ebooks to help get you started on your own projects.

https://machinelearningmastery.com/products/

Tareque November 5, 2023 at 10:36 am #

REPLY

In crossover function, for selecting crossover point in following line, it would be len(p1)-1 as randint uses high as exclusive.
pt = randint(1, len(p1)-2)

James Carmichael November 6, 2023 at 9:27 am #

REPLY

Thank you Tareque for your feedback!

Leave a Reply

Name (required)

Email (will not be published) (required)

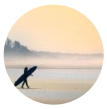
SUBMIT COMMENT

Welcome!
I'm *Jason Brownlee* PhD
and I **help developers** get results with **machine learning**.
Read more

Never miss a tutorial:

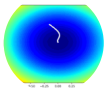
Picked for you:


Simple Genetic Algorithm From Scratch in Python


Curve Fitting With Python


A Gentle Introduction to Particle Swarm Optimization


Optimization for Machine Learning Crash Course


Code Adam Optimization Algorithm From Scratch