# Optimization Techniques for Machine Learning

*Alassane KONE*
*(ISE 2019 – Senior Data Scientist)*

*February 2023*

*Chapter 3*

# Grid & Random Search Algorithms

## 1. Naïve Optimization Algorithms

## Naive Function Optimization Algorithms

There are many different algorithms you can use for optimization, but how do you know whether the results you get are any good?

One approach to solving this problem is to establish a baseline in performance using a naive optimization algorithm.

**A naive optimization algorithm** is an algorithm that assumes nothing about the objective function that is being optimized.

If a more sophisticated algorithm cannot achieve a better result than a naive algorithm on average, then it does not have skill on your problem and should be abandoned.

There are two naive algorithms that can be used for function optimization; they are:

- ▷ **Random Search**
- ▷ **Grid Search**

## 2. Random Search for Function Optimization

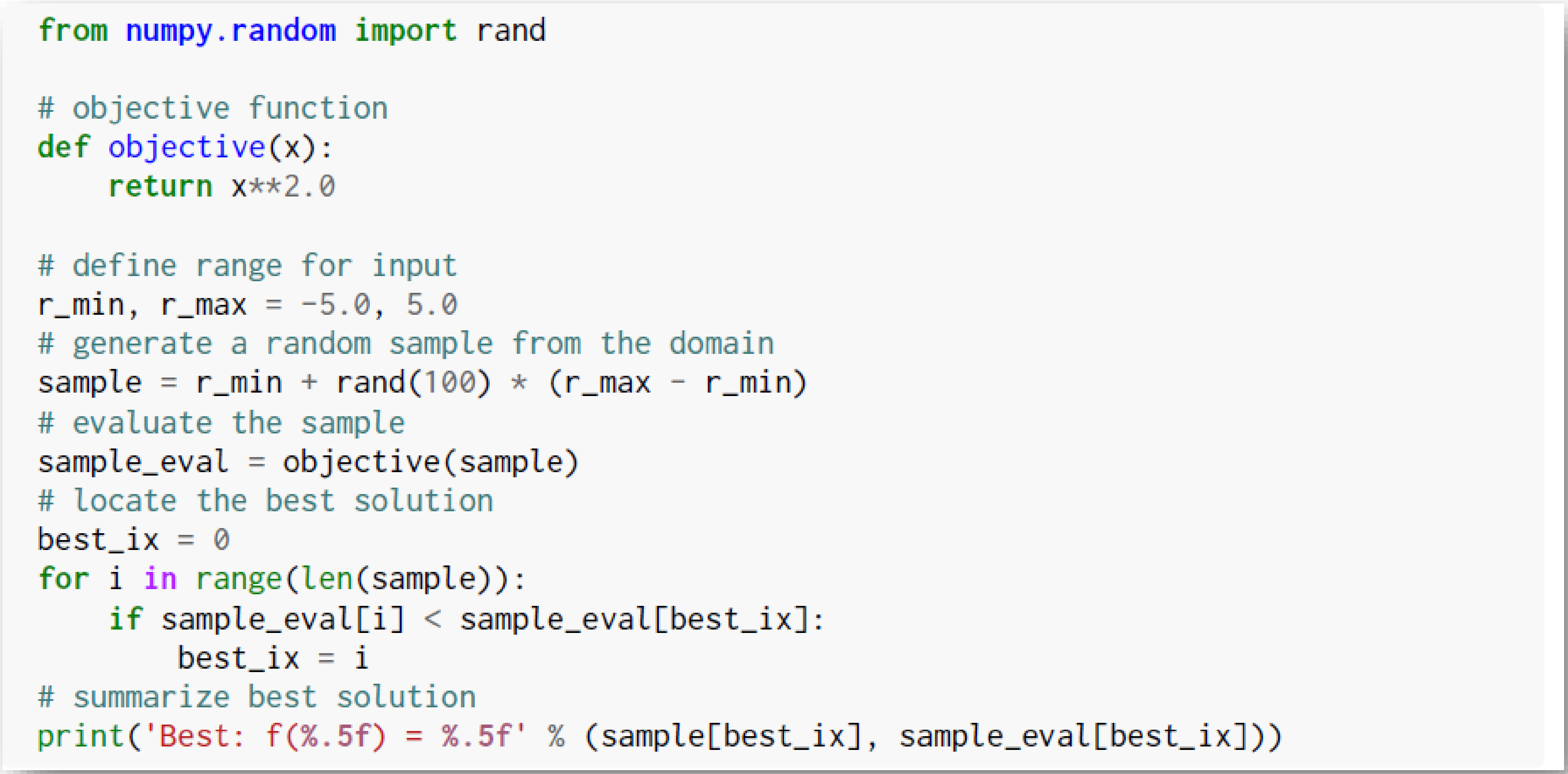
## Random Search Principle

**Random Search** involves generating and evaluating random inputs to the objective function. It's effective because it does not assume anything about the structure of the objective function. This can be beneficial for problems where there is a lot of domain expertise that may influence or bias the optimization strategy, allowing non-intuitive solutions to be discovered.

Generating random samples is computationally trivial and does not take up much memory, therefore, it may be efficient to generate a large sample of inputs, then evaluate them. Each sample is independent, so samples can be evaluated in parallel if needed to accelerate the process.

## Random Search - Example

The example below gives an example of a simple one-dimensional minimization objective function and generates then evaluates a random sample of 100 inputs. The input with the best performance is then reported.



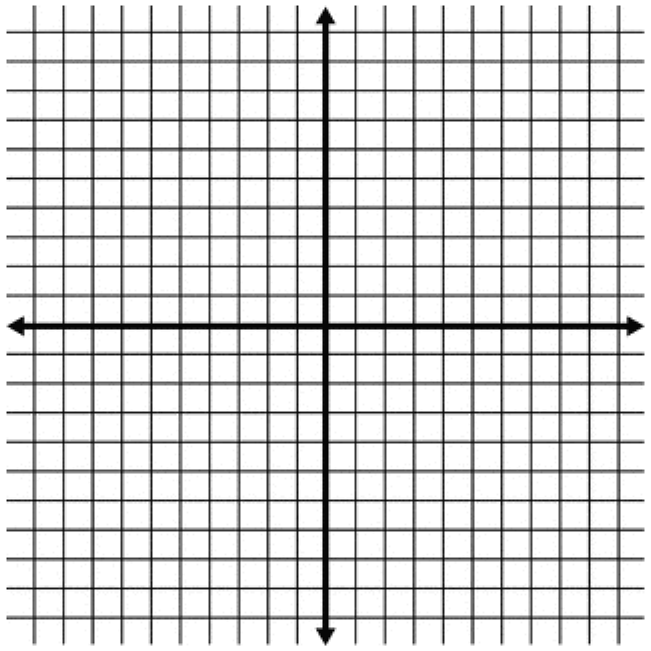
```python
from numpy.random import rand

# objective function
def objective(x):
    return x**2.0

# define range for input
r_min, r_max = -5.0, 5.0
# generate a random sample from the domain
sample = r_min + rand(100) * (r_max - r_min)
# evaluate the sample
sample_eval = objective(sample)
# locate the best solution
best_ix = 0
for i in range(len(sample)):
    if sample_eval[i] < sample_eval[best_ix]:
        best_ix = i
# summarize best solution
print('Best: f(%.5f) = %.5f' % (sample[best_ix], sample_eval[best_ix]))
```

## 3. Grid Search for Function Optimization

## Grid Search Principle

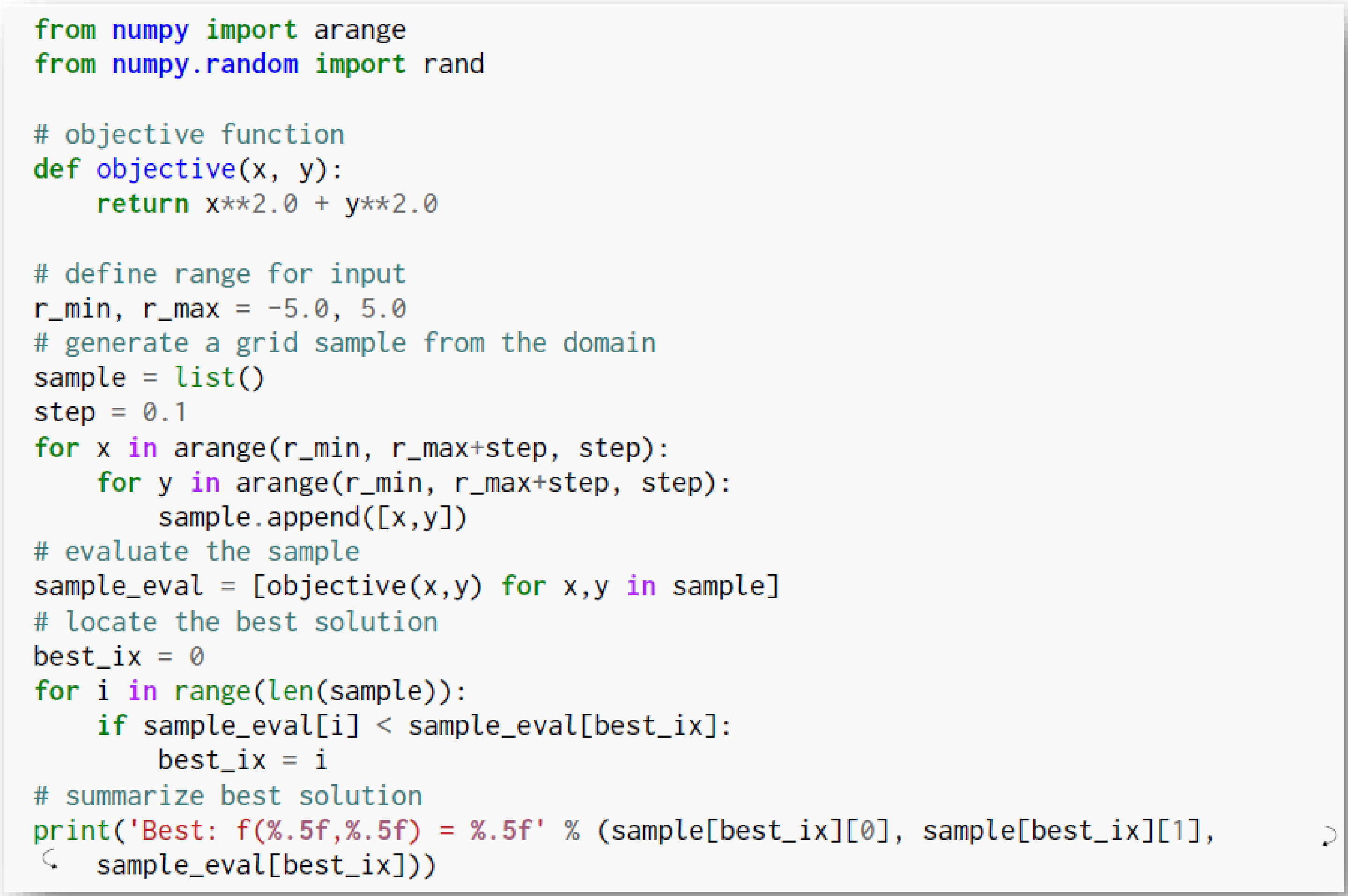
**Grid Search** is also referred to as a grid sampling or full factorial sampling. Grid search involves generating uniform grid inputs for an objective function. In one-dimension, this would be inputs evenly spaced along a line. In two-dimensions, this would be a lattice of evenly spaced points across the surface, and so on for higher dimensions.

Like random search, a grid search can be particularly effective on problems where domain expertise is typically used to influence the selection of specific optimization algorithms. The grid can help to quickly identify areas of a search space that may deserve more attention.



## Grid Search – Example (1/2)

The example below gives an example of a simple two-dimensional minimization objective function and generates then evaluates a grid sample with a spacing of 0.1 for both input variables. The input with the best performance is then reported.



```
from numpy import arange
from numpy.random import rand

# objective function
def objective(x, y):
    return x**2.0 + y**2.0

# define range for input
r_min, r_max = -5.0, 5.0
# generate a grid sample from the domain
sample = list()
step = 0.1
for x in arange(r_min, r_max+step, step):
    for y in arange(r_min, r_max+step, step):
        sample.append([x,y])
# evaluate the sample
sample_eval = [objective(x,y) for x,y in sample]
# locate the best solution
best_ix = 0
for i in range(len(sample)):
    if sample_eval[i] < sample_eval[best_ix]:
        best_ix = i
# summarize best solution
print('Best: f(%.5f,%.5f) = %.5f' % (sample[best_ix][0], sample[best_ix][1],
    sample_eval[best_ix]))
```

## Grid Search – Example (2/2)

We can update the example to plot the objective function and show the sample and best result.

```
from numpy import arange
from numpy import meshgrid
from numpy.random import rand
from matplotlib import pyplot
```

```
# sample input range uniformly at 0.1 increments
xaxis = arange(r_min, r_max, 0.1)
yaxis = arange(r_min, r_max, 0.1)
# create a mesh from the axis
x, y = meshgrid(xaxis, yaxis)
# compute targets
results = objective(x, y)
# create a filled contour plot
pyplot.contourf(x, y, results, levels=50, cmap='jet')
# plot the sample as black circles
pyplot.plot([x for x,_ in sample], [y for _,y in sample], '.', color='black')
# draw the best result as a white star
pyplot.plot(sample[best_ix][0], sample[best_ix][1], '*', color='white')
# show the plot
pyplot.show()
```