# Evolution Strategy Algorithm and You

Similar to Genetic Algorithm, the Evolution Strategy is a kind of Machine learning method without using gradient descent algorithm (used in Neural Network) to find optimized solutions. Instead, it let the ‘environment’ to choose the best fit and learn from it to adapt to the environment (The idea of Evolution).

The Evolution Strategy is very flexible and can be used when we can translate our problem to a ‘environment’ where good or bad can be tell.

Take our demo for example, our goal here is **to find location of each random point created from Archimedean Spiral curve.** To solve this problem, we need to create a ‘environment’ where locations closer to each blue point are considered to be better. My idea here is to think each random blue point as a planet, every planet have same gravity and the raster background in figure 1 can represent the gravitational field in this 2-D space (rescaled). Based on figure 1 we can easily tell that there are higher values / gravities when it close to blue points / planets while lower values when it far away from blue points. As a result, when we place a batch of points in this 2-D space, points closer to blue points / planets can be affected by higher gravity and be considered to be better. Thus, our environment has been successfully established.

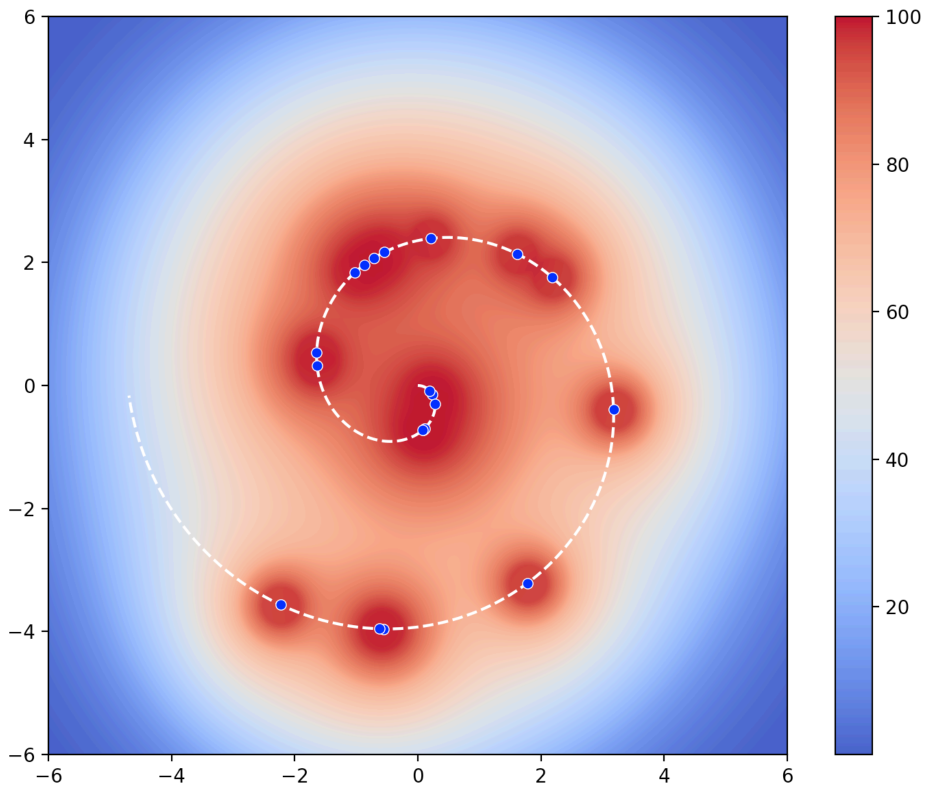

Figure 1

**Part 1, First version of this demo without grouping:**

Let’s then take a look of our workflow in the first version of this demo:

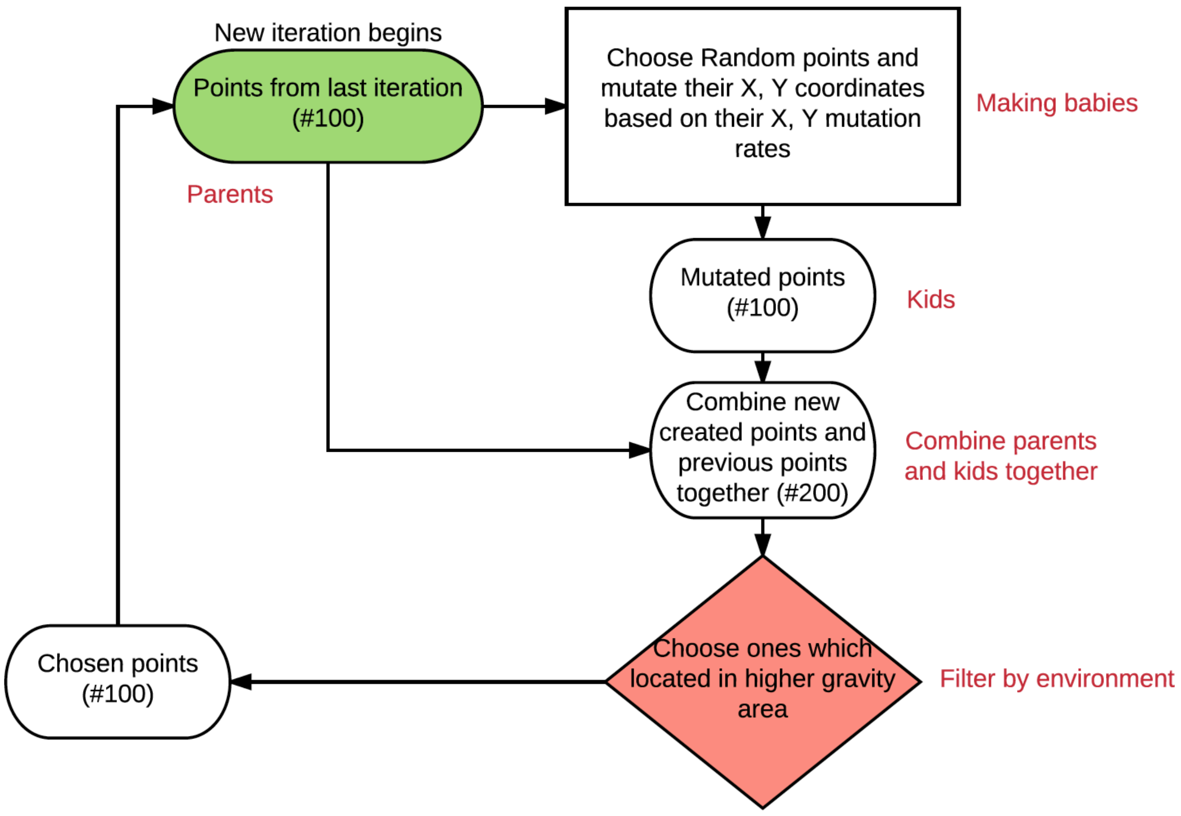
New iteration begins
Points from last iteration (#100)
Choose Random points and mutate their X, Y coordinates based on their X, Y mutation rates
Making babies
Parents
Mutated points (#100)
Kids
Combine new created points and previous points together (#200)
Combine parents and kids together
Chosen points (#100)
Choose ones which located in higher gravity area
Filter by environment

Figure 2

The workflow is relatively simple: 1. Based on 100 old points to create 100 new points (making babies), 2. Combine the old points and new created points together (number: 200), 3. Throw them into our environment and choose ones that can suit environment best (affected by higher gravities), 4. Put the chosen ones / points (number: 100) into another iteration.

Where these points come from and how to mutate their X, Y coordinates? You may ask. Well, let’s take a look of the data sample in our demo created with Evolution Strategy Algorithm:

Like the living beings in real world, in Evolution Strategy Algorithm, each data sample has their ‘DNA’ and ‘Mutation Rate’ properties. The ‘DNA’ controls the appearance of the data sample and the ‘Mutation Rate’ affect this data sample’s ability to change (or so called mutate). Let's take our demo as an example (let's begin with the first version of this demo with no grouping involved), we want to create points in this 2-D space and try to get close to each blue points from Archimedean Spiral curve, as a result, our data sample here is points with X, Y coordinates. The 'appearance' of our data samples is their locations or X and Y coordinates. As we can find in Figure 3, the red point shows us how our basic data structure is. Information is stored in a dictionary with two keys: 'DNA' and 'Mutation', DNA[0] = 0.5 represent the red point's X coordinate and DNA[1] = 0.5 is its Y coordinate. Mutation[0] and Mutation[1] respectively represent the mutation/change magnitude for X and Y coordinate in each iteration/generation.

While this is only the data for one point, the points as whole are store in arrays. For example, the data for these 5 points in Figure 3 is: pop = { 'DNA': array([ [ 0.3,  0.8,  0], [ 0.5,  1.0 ,  0], [ 0.5,  0.5,  0], [ 0.8,  0.3,  0], [ 1.0,  0.5,  0] ]), 'Mutation': array([ [ 0.1,  0.1, 0], [ 0.5,  0.3 ,  0], [ 0.3,  0.1,  0], [ 0.8,  0.1,  0], [ 0.1,  0.5,  0] ]) } We call our whole data sample as population (pop here).

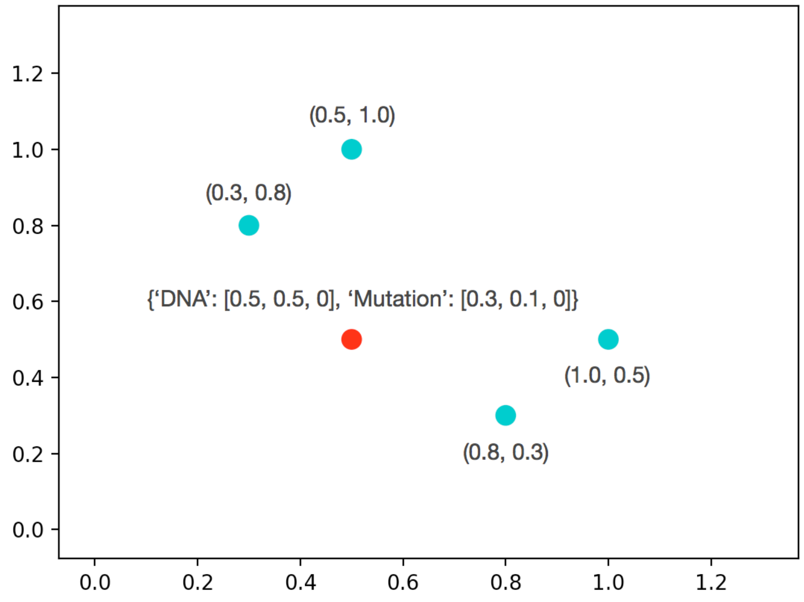

Figure 3

At the beginning of this program, we will initialize our points with random X, Y coordinates and mutation rates. After that, we randomly choose some points from previous points pool and based on the probability of normal distribution to receptively use their X, Y coordinates as mean value and X, Y mutation rate as standard deviation to create new points with new X, Y coordinates, we also change their mutation rates with a Δ (in this version is a random value). In this way, for each new created point, we actually use previous point as center point and randomly choose a point based on its

2-D normal probability distribution. The new created points will have some information inherit from their parents while still be different from their parents.

For each iteration, we will combine these new created points and old points together and calculate which ones are relatively affected by higher gravity. Only the top 100 points will be kept to new iteration.

As we can find in the gif, with choosing 'better' points in each iteration, points created with Evolution Strategy will be closer to blue points, when a point is close enough to a blue point, we will do two things:

1. We mark this point as 'success explorer' and do not move it again.
2. We also let the blue planet lose it attraction so it's easier for us to find other blue planets.

① For the first action, if you still remember the structure of our data, the 'DNA' property has three fields, first two fields respectively represent X and Y coordinates while third field is always zero, in the first version of this demo, once the point is very close to a blue point, its DNA's third field will be changed to -1, and this point will then be kept in the iteration without calculating its gravity.

② The reason for taking second action is to make it easier for us to find other high gravity locations. If the points in last iteration are all close enough to one blue planet (with very high gravity), it will be difficult for them to create kids to jump out of this local maxima to affected by a higher gravity. It's similar to the common problem in gradient descent algorithm that the optimized solution found by gradient descent algorithm is normally the local maxima solution. In our demo, I have two ideas to solve this problem, first is to let the first point which close enough to the blue planet create a repulsive force. Since this point is close enough to the blue planet, the attractive and repulsive forces will be balanced around this blue planet. Second is a easier way, is to simply erase this blue planet out of the planet list. In the first version of this demo, I choose to use the second option.