**Gradient Descent Method and its Application in three benchmark functions (Sphere, Rosenbrock, and Booth)**

**What is Gradient Descent in simple words?**

An optimization algorithm to find steepest decrease /minimum value of a differentiable function: f(x\*)= min f(x)

To find the optimal or minimizer x\* ,we start from a point x0 and follow the negative gradient direction of f at x0 and go on :

x1 := x0 - (step size ɣ0 \*( ∇f (x0))

But the algorithm may never reach out to x\* where ∇f = 0 but getting closer and closer. So, we define the absolute of ∇f < threshhold (*as a proxy for a number close to x\**) .This threshold is good as is a check against waiting for ever.

Also as a secondary condition to terminate the algorithm, limit the number of iterations based on the size of our step size(*larger step size requires less iterations*) that the algorithm would stop once reach out to one of these two stopping conditions.

**Applying Gradient Descent in three 2-dimentional benchmark functions:**

We are testing our Gradient Descent optimization algorithm such its convergence rate, by using 3 benchmark functions :

**a) 2-dimentional Sphere function f(x)= x1\*\*2 + x2\*\*2:**

The 2-dimentional Sphere function is convex and has 2 local minima except for one global minimum at x\*=(0,0) so during the gradient descent algorithm, we expect to get the global minima in a very close to (0,0).

In the plot of Sphere function can see this function is bowl-shaped and is shallower around (0,0) that it's global minimum :

**Chart, surface chart

Description automatically generated**

Sphere function is the most optimal for testing optimization algorithm in single objective case based but must be careful about choosing suitable parameters such as step size (*a positive number that controls the magnitude of the vector*) that is crucial for efficiency of this function.

The partial derivatives of this function are:

df/dx1 = 2\*x1    , df/dx2 = 2\*x2

So gradient of Sphere function is the row vector: ∇f = [2\*x1 , 2\*x2]

For my experiments in Sphere function ,I fixed the threshold to 0.00001 close to x\*=(0,0) as x\* is not reachable in practice.

In following, I experiment different start points and step sizes to see the impact of those parameters on Sphere function:

By experiencing different **start points** such as (-2.5,2.5) and (2,1), I saw for all points,in the first iterations , the algorithm tries to escape from minima by overpassing the minima, then experience zig-zagging before getting attracted by the global minima and end up to very close point in the neighborhood of (0,0) like bouncing of a heavy ball inside a bowl that the fluctuation size decreases continiously to get very small (see Figures 1 and 2)

With start point (-2.5,2.5) after 44 times fluctuating at iteration 45 converges to a very close point to the global minima , x= [0.00010889035741470056, -0.00010889035741470056] and in next step passes minima and reached to a far point:

x= [-8.711228593176045e-05, 8.711228593176045e-05]

Similarely,with start point (2,1) after 43 times fluctuating at iteration 44 converges to a very close point to the global minima , x= [0.00010889035741470056, 5.444517870735028e-05] and in next step passes minima and reached to a far point:

x= [-8.711228593176045e-05, -4.3556142965880224e-05]

Notice the density of dots in the neigborhood of center(0,0) that showing at first iterations fluctuating in larger sizes and in later interactions flactuating take smaller steps .

A picture containing text, electronics, display

Description automatically generated A screenshot of a computer

Description automatically generated with low confidence

Figure1: start point(-2.5,2.5) Figure 2: start point (2,1)

Iteration No: 44 Iteration No: 44

x= [0.0001361129467683757, -0.0001361129467683757] x= [0.00010889035741470056, 5.444517870735028e-05]

**Learning rate(step size)** saying how far should move in negative gradient direction and controls the speed of the algorithm ,so it is important to choose the optimal step size ɣi(line searching).Small step sizes ensure finding the minimum but needs many steps and slows down gradient descent & too large step sizes run faster, but more likely to overstep the minimum point or fluctuate around the minimum or even fails to converge.

To find a suitable learning rate , I used binary search and wrote a code that computes f(xi) &f(xi+1) for given x1, x2 that I obtained by implementing a guessed maximum learning rate in the gradient descent function to see if the decrease condition f(xi) > f(xi+1) is satisfied , if no ,decided ɣi is too large and means I did something wrong ,so undo the step and decrease ɣi by halfing it(B*inary search technique* )and if f(xi) > f(xi+1) decided it is better, but might could have been larger and we are looking for a sufficiently small step size*( not just small that result too many iterations*), so try for a bit larger ɣi and so on to find the optimal step size:

I started with 8 and was too large , then tried learning rates 4, 2, 1,0.5 that did not satify f(xi) > f(xi+1) then tried 0.25 and as satisfied, began to check for a bit larger and go on, finally found learning rate =0.49 that starts to converge and at step 490 reached to: x= [0.00010041696357939963, 0.00010041696357939963]

In Figures 3, 4 and 5 can see how the small step size 0.49 controls the speed of the algorithm the magnitude of each fluctuate is decreasing smoothly and converge slowly and at step 491 diverges to x= [-9.840862430781163e-05, -9.840862430781163e-05]

A picture containing text, electronics, display

Description automatically generated Chart

Description automatically generated with low confidence

Figure 3: learning rate =0.49 Figure 4: learning rate =0.49

Iteration No: 50 Iteration No: 140

x= [0.7283393601742334, 0.7283393601742334] x= [0.11821717926817424, 0.11821717926817424]

A picture containing text, electronics, display

Description automatically generated

Figure 5: learning rate =0.49

Iteration No: 490

x= [0.00010041696357939963, 0.00010041696357939963]

I tried smaller learning rate 0.45 as the previous one although satisfied the decrease condition but needs 490 iterations that slows down the algorithm:Leraning rate=0.45 satisfied decrease condition that guarantees convergence to a minima. At iteration 93 obtained to x= [-0.00011106657345087347, -0.00011106657345087347] (a bit farther than x in learning rate 0.49 but faster)and then jumped over minima to reach x= [1.2152792193295453e-05, 1.2152792193295453e-05]

A screenshot of a computer

Description automatically generated with low confidence

Figure 6: learning rate =0.45

Iteration No: 93

x= [-0.00011106657345087347, -0.00011106657345087347]

Notice with learning rate 0.40 at iteration 44 we reached similar x in the neighborhood of global minima instead of iteration 490 with learning rate 0.49, so a suitable learning rate can speed up the algorithm in convergence. Also for optimal ɣi ,if increases i, ɣi would be smaller and smaller .

To correct the impact of step size ɣ, we can shrink it by multiplying to a coefficient **momentum**. so we 'll get following gradient descent algorithm:

x1 := x0 \* momentum **-** (step size ɣ0 \*( ∇f (x0)) for momentum is in [0,1]

I fix parameters learning rate for 0.49 , start point=(2,2) and checked for the correction between without momentum *(momentum =0,* Figure5 *)*and with the momentum=0.75and 0.60:

A picture containing text, electronics, display

Description automatically generated A picture containing text, electronics, display

Description automatically generated

Figure 7:momentum =0.75 Figure 8: momentum=0.60

Iteration No: 6 Iteration No: 12

x= [0.0002960717779999999, 0.0002960717779999999] x= [0.00012556423695976466, 0.00012556423695976466]

Notice the markable speed in algorithm by implementing momentum only in 6 steps

reached to a very close point to (0,0) , with same leraning rate. Can say momentum acts like gravity to converge faster and faster with time.Also by decreasing momentum to 0.60 obtained a closer point but in 12 steps.

So can conclude that gradient descent method can be a great choice in Sphere function specially by choosing the optimal step size and implementing momentum to converge to global minimum. The algorithm always converges linearly to the neighborhood of the optimal minima.Also concider that as the Sphere is a convex function, ,this minima is the global minimal x\* and not only a local minima. That could be another reason that gradient descent works really well where the testing function is convex, because everything point downhill.

**b) 2-dimentional Rosenbrock function** **f(x1,x2)= (1-x1)\*\*2 + 100\*(x2-x1\*\*2)\*\*2:**

Rosenbrock function has the global minima lies in a narrow flat valley,so convergence to that is difficult and usually evaluated on xi ∈ [-5.12, 5.12]. and it's global minimum is x\*= (1,1).

In the plot of Rosenbrock function can see is valley-shaped :

Chart, surface chart

Description automatically generated

The partial derivatives of this function are:

df/dx1 =2\*(x1-1) - 400\*(x2 - x1\*\*2)\*x1  = 2\*x1-2-400\*x1\*x2+400\*x1\*\*3

df/dx2 = 200\*(x2-x1\*\*2) = 200\*x2-200\*x1\*\*2

So gradient of Rosenbrock function is the row vector:

∇f = [2\*x1-2-400\*x1\*x2+400\*x1\*\*3, 200\*x2-200\*x1\*\*2]

For my experiments in Rosenbrock function ,I fixed the threshold to 1.00001 as is close to x\*=(1,1).

By applying Gradient Descent for start point=(2,2) at first iteration obtained x= [-1.602, 0.4]:

A picture containing text, electronics, display, computer

Description automatically generated

Figure 9: start point (2,2)

Iteration No: **1** x= [-1.602, 0.4]

I tried a different start point (5,5) and got divergence for ever after iteration 4 for point: x=[1.1884381216107055e+38, 8.905146139893645e+24]So seems finding the global minima is too hard and can not be fixed with choosing a proper start point. I also tried binary search with different step sizes to find the optimal learning rate (*with same start point(2,2)*) for all large sizes got divergence :

In case learning rate =0.01 for both start points (2,2) and (5,5) divergence obtained by gradient descent .Tried smaller 0.001 converges with start point (2,2) and diverges with start point (5,5).Also tried 0.0001 that converges for both start points but then stays and locked up there for ever:

Chart, pie chart

Description automatically generated Chart

Description automatically generated

Figure 10 :start point (2,2) Figure 11 :start point(5,5)

learning rate=0.0001

As see gradient descent method is slow along the valley, can implement momentum as the gravity to help get converge faster. I fixed learning rate=0.0001 and checked for the correction between without momentum(Figure13) and with the momentum=0.75:

A picture containing text, electronics, display, screenshot

Description automatically generated A screenshot of a computer

Description automatically generated with low confidence

Figure12: with momentum=0.75 Figure13: with momentum=0.50

Iteration No: 1 Iteration No: 1

x= [1.3397999999999999, 1.54] x= [0.8398, 1.04]

With momentum has a great correction to converge .Here with momentum=0.5 even get a closer point x= [0.8398, 1.04] with a faster converge at first iteration .

With these small learning rates 0.001 or 0.0001 can conclude that gradient over a nearly flat region has small magnitude.

This experiments has shown that both start points and learning rates are really crucial for covergence in rosenbrook function and kind of really hard to find them and gradient descent is not a good method to use in this function.Based on more research on this, found out can define Rosenbrock function as following with k=1 instead k=100 that is defined earlier:

f(x1,x2)= (1-x1)\*\*2 +K (x2-x1\*\*2)\*\*2

That with k=1 could have a stronger convergence , even golden section method resulted in the fastest convergence rate in comparison with Gradient Descent algorithm with fixed step size.So Gradient descent is not a good choice for Rosenbrock function.

**c) 2-dimentional Booth function f(x1,x2)= (x1+ 2x2 -7)\*\*2 + (2x1 +x2 -5)\*\*2:**

 Booth function has several local minimas and global minimum x\*= (1,3).

In the plot of Booth function can see is plate-shaped :

Chart, surface chart

Description automatically generated

The partial derivatives of this function are:

df/dx1 = 2\*(x1+2\*x2-7) +4\*(2\*x1+x2-5)  = 10\*x1 + 8\*x2-34

df/dx2= 4\*(x1+2\*x2-7) +2\*(2\*x1+x2-5)]) =  8\*x1 + 10\*x2-38

So gradient of Booth function is the row vector:

∇f = [10\*x1 + 8\*x2-34 , 8\*x1 + 10\*x2-38]

By applying Gradient Descent for start point=(-10,10) and learning rate=0.01,we'll get the Figure 14 that at first step converges rapidly to x= [0.054, 0.018000000000000002] but then convergence got so slow even after 100000 iterations it is not reached and stuck around point x= [0.03336744977628241, 0.03735946574434628]:

Chart

Description automatically generated Chart

Description automatically generated

Figure 14:start point=(-10,10) Figure 15: start point=(-5,5)

Iteration No: 1 Iteration No: 1

x= [0.054, 0.018000000000000002] x=[0.044,0.028]

stucked x= [0.03336744977628241, 0.03735946574434628] around x= [0.03336744977628241, 0.03735946574434628]

As shown, in all start points at first step have a sharp convergence to the valley but then stuck around a point for ever and never converge to optimal point.

I also tried binary search with different step sizes to find the optimal learning rate (*with same start point(-5,5)* ) for all learning rates0.5,0.3, 0.1(diverges), 0.01(converges):

Chart

Description automatically generatedChart

Description automatically generated

Figure 16: learning rate=0.01 Figure 17: learning rate=0.001

Iteration No: 1 x= [0.44, 0.28] Iteration No: 1 x= [0.044, 0.028]

stuck x= [0.28547690262545694, 0.3246925888999667] x= [0.03336744977628241, 0.03735946574434628]

Chart

Description automatically generated

Figure 18: learning rate=0.0001 momentum =0

Iteration No: 1 x= [0.0044, 0.0028]

x= [0.0033935716350441234, 0.003793491651040924]

These figures illustrate that from any start point with any step size,at first step we get a sharp convergence that jumpes over minima and then stuck around a single point for ever

To see the impact of momentum, I fixed learning rate=0.0001 and start point(-5,5) and check for the correction between without momentum (Figure 18) and with the momentum=0.75(Figure19):

A picture containing text, screenshot, electronics, display

Description automatically generated

Figure 19: learning rate=0.0001 momentum = 0.75

Iteration No: 2 x= [-2.80505664, 2.81764368]

x= [0.013497700648059784, 0.015096421671241249]

Although now the convergence is smoothier here by implementing momentum, but still no improvement in results as still in trouble of stucking around a single point of x= [0.013497700648059784, 0.015096421671241249]for ever that even is far from optimal (1,3).Can see by gradient descent method convergence might takes long and this is not a good option for Booth function, too.

From all these experiments, conclude that although Gradient Descent gives meaningful results for both convex and non-convex functions, but it really works well (such it's convergence rate) with convex function (such as Sphere function) since

everything points downhill in comparison with non-convex functions (such as Rosenbrock and Booth) that we got trouble for convergence to optimal x\* and stuck around a point far away from x\*.

References:

[Test\_functions\_for\_optimization](https://en.wikipedia.org/wiki/Test_functions_for_optimization)

<http://www.geatbx.com/ver_3_3/fcnfun2.html>

[https://machinelearningmastery.com/2d-test-functions-for-function-optimization](https://machinelearningmastery.com/2d-test-functions-for-function-optimization/)

<https://arxiv.org/pdf/2101.10546.pdf>