Unit 8 – Gradient Cost Fucntion

Read the Mayo, 2017 article and then run the tutorial: **gradient_descent_cost_function.ipynb**. Change the iteration number and learning rate and observe how cost decreases.

Mayo (2017) notes –

In order for a neural network to learn, weights associated with neuron connections must be updated after forward passes of data through the network.

Learning Rate

If the learning rate is too low then the gradient descent algorithm will converge very slowly, which means you might need a very large number of iterations to reach the minimum, or you might never get there within a reasonable time.

If the learning rate is too high, the algorithm might overshoot the minimum. In the worst case, it might fail to converge or even diverge.

Number of iterations

Few Iterations: Too few iterations might not be sufficient for the algorithm to converge to the minimum.

Many Iterations: Having too many iterations can lead to unnecessary computation if the minimum has already been reached, or in some cases, if the learning rate is inappropriate, it might start to diverge after staying stable for a while.

A good strategy is to use a logarithmic scale to change the learning rate (e.g., 0.001, 0.01, 0.1) and a linear scale for iterations (e.g., 100, 1000, 10000).

```
# code credit:codebasics https://codebasics.io/coming-soon

import numpy as np

def gradient_descent(x,y):
    m_curr = b_curr = 0
    iterations = 100        #change value
    n = len(x)
    learning_rate = 0.08    #change value

    for i in range(iterations):
        y_predicted = m_curr * x + b_curr
        cost = (1/n) * sum([val**2 for val in (y-y_predicted)])
        md = -(2/n)*sum(x*(y-y_predicted))
        bd = -(2/n)*sum(y-y_predicted)
```

```
        m_curr = m_curr - learning_rate * md
        b_curr = b_curr - learning_rate * bd
        print ("m {}, b {}, cost {} iteration
{}".format(m_curr,b_curr,cost, i))

x = np.array([1,2,3,4,5])
y = np.array([5,7,9,11,13])

gradient_descent(x,y)
```

m 4.96, b 1.44, cost 89.0 iteration 0
m 0.4991999999999983, b 0.26879999999999993, cost 71.10560000000002 iteration 1
m 4.451584000000002, b 1.426176000000001, cost 56.8297702400001 iteration 2
m 0.892231679999997, b 0.5012275199999995, cost 45.43965675929613 iteration 3
m 4.041314713600002, b 1.432759910400001, cost 36.35088701894832 iteration 4
m 1.2008760606719973, b 0.7036872622079998, cost 29.097483330142282 iteration 5
m 3.7095643080294423, b 1.4546767911321612, cost 23.307872849944438 iteration 6
m 1.4424862661541864, b 0.881337636696883, cost 18.685758762535738 iteration 7
m 3.4406683721083144, b 1.4879302070713722, cost 14.994867596913156 iteration 8
m 1.6308855378034224, b 1.0383405553279617, cost 12.046787238456794 iteration 9
m 3.2221235247119777, b 1.5293810083298451, cost 9.691269350698109 iteration 10
m 1.7770832372205707, b 1.1780607551353204, cost 7.8084968312098315 iteration 11
m 3.0439475772474127, b 1.5765710804477953, cost 6.302918117062937 iteration 12
m 1.8898457226770244, b 1.3032248704973899, cost 5.098330841763168 iteration 13
m 2.898169312926714, b 1.6275829443328358, cost 4.133961682056365 iteration 14
m 1.9761515088959358, b 1.4160484030347593, cost 3.361340532576948 iteration 15
m 2.7784216197824048, b 1.6809279342791488, cost 2.741808050753047 iteration 16
m 2.0415541605113807, b 1.5183370872989306, cost 2.244528230107478 iteration 17
m 2.6796170361078637, b 1.735457156285639, cost 1.8449036666988363 iteration 18
m 2.090471617540917, b 1.611567833948162, cost 1.5233119201782324 iteration 19
m 2.5976890103737853, b 1.790290604096816, cost 1.2640979056612756 iteration 20
m 2.1264168621494517, b 1.6969533824619085, cost 1.0547704368105268 iteration 21
m 2.529385561184701, b 1.8447607474362664, cost 0.8853615531285766 iteration 22
m 2.1521818147302194, b 1.7754939584778073, cost 0.7479156468369821 iteration 23
m 2.472104720735685, b 1.898367654050852, cost 0.6360820885229722 iteration 24
m 2.1699839382964696, b 1.8480185634495874, cost 0.5447903801652151 iteration 25
m 2.423763296438881, b 1.950743302915348, cost 0.4699911136477278 iteration 26
m 2.1815831093070837, b 1.9152179921582295, cost 0.4084494012702221 iteration 27
m 2.3826922006906663, b 2.0016232209455125, cost 0.35758014655339476 iteration 28
m 2.1883747814212473, b 1.9776712492627107, cost 0.31531667795040486 iteration 29
m 2.3475529664737507, b 2.0508239542984783, cost 0.280005985849834 iteration 30
m 2.19146424741668, b 2.0358666977033213, cost 0.2503251729489924 iteration 31
m 2.317271157065729, b 2.0982251873107836, cost 0.2252148202231392 iteration 32
m 2.19172583072087, b 2.0902190019495084, cost 0.2038258415569305 iteration 33
m 2.2909832477163747, b 2.143755628915694, cost 0.1854770944836773 iteration 34
m 2.1898500615476015, b 2.1410827139250586, cost 0.16962156815305135 iteration 35

m 2.2679942505397945, b 2.1873814501542004, cost 0.15581941113049289 iteration 36
m 2.1863812735157397, b 2.188763177870427, cost 0.14371641365577967 iteration 37
m 2.247743906750233, b 2.2290980581236037, cost 0.13302683968149862 iteration 38
m 2.181747562970493, b 2.2335252935837153, cost 0.1235197278285518 iteration 39
m 2.2297797112222417, b 2.268922416384484, cost 0.11500795886067308 iteration 40
m 2.176284659606544, b 2.2756005683762908, cost 0.1073395295828815 iteration 41
m 2.213735385878407, b 2.306887840824943, cost 0.10039058653754176 iteration 42
m 2.170254943136438, b 2.315192801071317, cost 0.09405986334903649 iteration 43
m 2.1993136987020745, b 2.3430395801944157, cost 0.08826423771269985 iteration 44
m 2.1638625904931037, b 2.3524826719863134, cost 0.08293518155053264 iteration 45
m 2.1862727486718105, b 2.3774314010318136, cost 0.07801592372722821 iteration 46
m 2.1572656385141533, b 2.3876314575042543, cost 0.07345918129710392 iteration 47
m 2.1744150151272015, b 2.4101229178167802, cost 0.06922534441882239 iteration 48
m 2.150585587951272, b 2.4207840437050385, cost 0.06528102333197575 iteration 49
m 2.1635786121786147, b 2.441177514495622, cost 0.06159788433500965 iteration 50
m 2.1439150477863547, b 2.4520713783305874, cost 0.05815171649232756 iteration 51
m 2.153630302083688, b 2.470660734860243, cost 0.05492168259098864 6 iteration 52
m 2.137323817683481, b 2.4816124722824338, cost 0.05188971727120564 iteration 53
m 2.1444599118649865, b 2.4986390442291735, cost 0.04904004275389065 iteration 54
m 2.1308637257526066, b 2.509516039457312, cost 0.04635877856867898 iteration 55
m 2.1359758694885094, b 2.525178884782891, cost 0.04383362645496491 iteration 56
m 2.124572474492945, b 2.535881845863144, cost 0.04145361541183607 iteration 57
m 2.128101633371053, b 2.5503459627684273, cost 0.03920889490609494 iteration 58
m 2.118476696509155, b 2.5608018247073736, cost 0.03709056666678448 iteration 59
m 2.120772834793503, b 2.5742047184297996, cost 0.03509054742420437 iteration 60
m 2.112594380710634, b 2.5843610027801502, cost 0.03320145649050715 iteration 61
m 2.1139349893254455, b 2.5968179395942217, cost 0.03141652330669783 iteration 62
m 2.1069367971074353, b 2.606638274382932, cost 0.0297295110602709 iteration 63
m 2.1075416624945413, b 2.618246487870094, cost 0.0281346532591438 iteration 64
m 2.1015100223265035, b 2.6277070518134993, cost 0.02662660077102551 iteration 65
m 2.101552998161378, b 2.6385491128066176, cost 0.025200377334927134 iteration 66
m 2.096316147250177, b 2.6476358156400974, cost 0.023851341948628327 iteration 67
m 2.095934536582619, b 2.657782334457597, cost 0.022575156852933542 iteration 68
m 2.0913542316575633, b 2.6664885833847243, cost 0.021367760086655644 iteration 69
m 2.090656263915584, b 2.676000378847538, cost 0.020225341788425083 iteration 70
m 2.0866210575773376, b 2.6843253115524512, cost 0.019144323582910155 iteration 71
m 2.0856918466960472, b 2.693255154066937, cost 0.018121340518098442 iteration 72
m 2.082111722558874, b 2.7012022430021245, cost 0.017153225123473996 iteration 73
m 2.0810180142142363, b 2.709596257293525, cost 0.01623699324146153 iteration 74
m 2.077820105696288, b 2.717172209303728, cost 0.015369831350564987 iteration 75
m 2.0766140592050317, b 2.7250710050809133, cost 0.014549085151541019 iteration 76
m 2.0737392325653374, b 2.732284895849552, cost 0.013772249230347736 iteration 77
m 2.0724614332425584, b 2.7397244808822614, cost 0.013036957645638886 iteration 78
m 2.0698615599121704, b 2.7465870759846718, cost 0.012340975315898037 iteration 79
m 2.0685434179941087, b 2.7535995950692826, cost 0.011682190103287127 iteration 80
m 2.066179196691222, b 2.760122819221025, cost 0.011058605508984853 iteration 81
m 2.064844857288579, b 2.7667371537338745, cost 0.010468333909073289 iteration 82
m 2.0626840746684203, b 2.7729336776379365, cost 0.009909590271584152 iteration 83

m 2.061351937985791, b 2.7791759333750248, cost 0.009380686304666458 iteration 84
m 2.0593680791107873, b 2.785058853801841, cost 0.00888002499345325 iteration 85
m 2.0580520100509183, b 2.7909527592203687, cost 0.00840609548939642 iteration 86
m 2.056223147935525, b 2.7965353529206687, cost 0.007957468320912865 iteration 87
m 2.05493343816708, b 2.8021025854443096, cost 0.00753279089835229 6 iteration 88
m 2.0532413459797505, b 2.8073981214530215, cost 0.007130783289727114 iteration 89
m 2.0519854787579397, b 2.812658575950258, cost 0.0067502342464980354 iteration 90
m 2.050414919687842, b 2.8176801739944057, cost 0.006389997461079277 iteration 91
m 2.0491981775199255, b 2.8226521847051367, cost 0.006048988039717134 iteration 92
m 2.047736336426391, b 2.8274127099427506, cost 0.005726179176078216 iteration 93
m 2.0465622835434223, b 2.8321132348672426, cost 0.005420599012302664 iteration 94
m 2.045198311770722, b 2.836625221187641, cost 0.005131327675503935 iteration 95
m 2.0440691768841837, b 2.8410699961476715, cost 0.0048574944787420265 iteration 96
m 2.042793827417138, b 2.845345591859636, cost 0.004598275276411798 iteration 97
m 2.0417108070703494, b 2.8495492600018677, cost 0.004352889964781892 iteration 98
m 2.0405161418256377, b 2.8536001910078013, cost 0.004120600119124239 iteration 99

```
import numpy as np

def gradient_descent(x,y):
    m_curr = b_curr = 0
    iterations = 1000       #change value
    n = len(x)
    learning_rate = 0.008   #change value

    for i in range(iterations):
        y_predicted = m_curr * x + b_curr
        cost = (1/n) * sum([val**2 for val in (y-y_predicted)])
        md = -(2/n)*sum(x*(y-y_predicted))
        bd = -(2/n)*sum(y-y_predicted)
        m_curr = m_curr - learning_rate * md
        b_curr = b_curr - learning_rate * bd
        print ("m {}, b {}, cost {} iteration
{}".format(m_curr,b_curr,cost, i))

x = np.array([1,2,3,4,5])
y = np.array([5,7,9,11,13])

gradient_descent(x,y)
```

Result: m 2.041932999393225, b 2.8486085669217984, cost 0.004194384603694259 iteration 999
This is still not a lower cost than the original run

```
import numpy as np
```

```
def gradient_descent(x,y):
    m_curr = b_curr = 0
    iterations = 10000       #change value
    n = len(x)
    learning_rate = 0.0008   #change value

    for i in range(iterations):
        y_predicted = m_curr * x + b_curr
        cost = (1/n) * sum([val**2 for val in (y-y_predicted)])
        md = -(2/n)*sum(x*(y-y_predicted))
        bd = -(2/n)*sum(y-y_predicted)
        m_curr = m_curr - learning_rate * md
        b_curr = b_curr - learning_rate * bd
        print ("m {}, b {}, cost {} iteration
{}".format(m_curr,b_curr,cost, i))

x = np.array([1,2,3,4,5])
y = np.array([5,7,9,11,13])

gradient_descent(x,y)
```

It looks like the minimum cost is slowly increasing so let's revert back to iterations of 1000 and a learning rate between 0.008 and 0.08

```
import numpy as np

def gradient_descent(x, y):
    m_curr = b_curr = 0
    iterations = 1000
    n = len(x)
    learning_rate = 0.018
    min_cost = float('inf')  # Initialize to a very large value
    min_cost_iteration = -1  # To store the iteration at which min cost
occurs

    for i in range(iterations):
        y_predicted = m_curr * x + b_curr
        cost = (1/n) * sum([val**2 for val in (y - y_predicted)])
        if cost < min_cost:   # Check if the current cost is lower than
the recorded minimum cost
            min_cost = cost
            min_cost_iteration = i

        md = -(2/n) * sum(x * (y - y_predicted))
        bd = -(2/n) * sum(y - y_predicted)
```

```
        m_curr = m_curr - learning_rate * md
        b_curr = b_curr - learning_rate * bd
        print(f"Iteration {i}: m = {m_curr}, b = {b_curr}, cost = {cost}")

    # After finishing all iterations, print the minimum cost and the
iteration it occurred at
    print(f"Minimum cost of {min_cost} occurred at iteration
{min_cost_iteration}")

# Example data
x = np.array([1, 2, 3, 4, 5])
y = np.array([5, 7, 9, 11, 13])

# Call the function
gradient_descent(x, y)
```

Minimum cost of 4.742248556930931e-06 occurred at iteration 999, when written in full decimal form it is 0.000004742248556930931