cos_dist_example_fail

July 7, 2023

[1]:
```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # suppress tensorflow warnings https://
 ↪stackoverflow.com/a/40871012
from deepface import DeepFace
import subprocess
import numpy as np
from decimal import Decimal # for proper rounding
import random
import time
import pandas as pd
from datetime import datetime
import struct


# CONSTANTS
EXECUTABLE_PATH = "ABY/build/bin"
INPUT_FILE_NAME = "input_vecs.txt"
EXECUTABLE_NAME_SCENARIO = 'cos_dist'
CMD_SCENARIO = f"./{EXECUTABLE_NAME_SCENARIO} -r 1 -f {INPUT_FILE_NAME} & (./
 ↪{EXECUTABLE_NAME_SCENARIO} -r 0 -f {INPUT_FILE_NAME} 2>&1 > /dev/null)"

# random number generator
rng = np.random.default_rng()
```

[2]:
```
def run_sfe(x, y, y_0, y_1):
    # write the original 2 vectors to a file (second vector used only for␣
 ↪verification)
    with open(f"{EXECUTABLE_PATH}/{INPUT_FILE_NAME}", 'w') as f:
        for x_i, y_i in zip(x, y):
            f.write(f"{x_i} {y_i}\n")

    # write the shares into separate files
    with open(f"{EXECUTABLE_PATH}/share0.txt", 'w') as f:
        for i in y_0:
            f.write(f"{i}\n")
    with open(f"{EXECUTABLE_PATH}/share1.txt", 'w') as f:
        for i in y_1:
            f.write(f"{i}\n")

    # execute the ABY cos sim computation
    output = subprocess.run(CMD_SCENARIO, shell=True, capture_output=True,␣
 ↪text=True, cwd=EXECUTABLE_PATH)
    assert (output.returncode == 0) # make sure the process executed␣
 ↪successfully

    return output

def get_embedding(imagepath):
    return DeepFace.represent(img_path = imagepath, model_name="SFace",␣
 ↪enforce_detection=True)[0]["embedding"]

def get_two_random_embeddings(same_person):
    """Get two random embeddings of either the same person or two different␣
 ↪people out of all the images available"""
    people = os.listdir('lfw') # list of all people that have images
    people_with_multiple_images = [p for p in people if len(os.listdir(f"lfw/
 ↪{p}")) > 1] # list of people with more than one image in folder
    embedding1, embedding2 = None, None # face embeddings
    while embedding1 is None or embedding2 is None: # try until the chosen␣
 ↪images have detectable faces
        try:
            if same_person:
                # same person should have more than one image (we might still␣
 ↪end up choosing the same image of that person with prob 1/n, but that's ok)
                person1 = random.choice(people_with_multiple_images)
                person2 = person1
            else:
                # two persons chosen should be different
                person1 = random.choice(people)
                person2 = random.choice([p for p in people if p != person1])
            # get two random images
            img1 = f"lfw/{person1}/{random.choice(os.listdir(f'lfw/
 ↪{person1}'))}"
            img2 = f"lfw/{person2}/{random.choice(os.listdir(f'lfw/
 ↪{person2}'))}"
            # try to extract embeddings from both images
            embedding1 = get_embedding(img1)
            embedding2 = get_embedding(img2)
        except Exception as e:
            # failed to detect faces in images, try again
            # print(e)
            pass
    return np.array(embedding1), np.array(embedding2)
```

[3]:
```
# 1st try with real data

# Get two embeddings of images that we will be comparing.
a, b = get_two_random_embeddings(same_person=False)

# This is how they look like raw
a
```

[3]:
```
array([-1.05887663,  1.1577996 ,  0.6159687 ,  0.92861074,  0.54197681,
        0.1466969 , -0.98524421,  0.57258457, -0.44633129,  0.98824561,
        0.98547626,  0.61538422,  1.88092852, -0.75701547, -0.3482213 ,
       -0.73323172, -0.31149465,  0.10216737,  0.19858135,  0.78104699,
       -0.42418146,  0.74085951, -0.4718197 , -0.35617095, -0.04339859,
        1.2745738 ,  0.19090196,  0.57039332, -0.29325372,  1.48540282,
       -0.71376383,  1.09925592,  1.79980183,  0.40419835, -0.083152  ,
        0.78684235, -0.10030162,  1.45961678, -0.71542764,  0.4785786 ,
       -0.38525358, -0.11947311,  1.152475  ,  0.50615567, -0.85845679,
       -0.46455294, -0.2670826 ,  0.45135278,  0.74665898,  0.57165748,
       -0.49452114, -0.58083856, -0.16251093,  0.16182835,  0.66316724,
        0.72607446, -0.01495208,  0.23632735,  0.4999547 ,  0.38669533,
        1.13880157,  0.30691868,  0.71149141, -1.86636066, -1.14623201,
       -0.19288938,  0.21882766,  0.36012024,  0.49171108, -0.1915514 ,
        0.21039471, -0.28006303, -0.40239754, -0.52252084,  0.10193172,
        0.23203063, -0.71445179,  0.66201377, -0.79982138, -0.47707921,
       -0.65932757,  0.59670687, -0.18353012, -0.16381007, -0.88789183,
       -0.59239727,  0.02188   ,  0.06982686,  0.55349922,  0.020704  ,
       -1.85362375, -0.717574  ,  1.78896022, -0.49059129, -0.25995997,
       -1.13007164,  2.02428913,  0.23665711,  0.0587305 ,  1.20122004,
        0.06428985, -0.70902276, -0.05982707, -0.25526357,  1.23449826,
       -0.72945988, -0.41808292, -0.63352889, -0.94629824, -0.58669686,
        0.24389631,  0.25320318,  0.3431409 , -0.733684  ,  2.03171349,
        0.37167704,  1.00190759,  0.39158776, -0.12941344, -0.30270082,
        0.92834449, -0.27996159, -0.18637286,  0.11075099,  1.21908152,
        0.26202917,  0.30031919,  0.58085823])
```

[4]:
```
# Make the shares
# x is the captured face, y is the face in the database
# First, scale the values up by 10 000, then get rid of the decimal part then␣
 ↪cast to int

x = (a * 10000).round().astype(int)
y = (b * 10000).round().astype(int)

# so far so good
x
```

```
[4]: array([-10589,  11578,   6160,   9286,   5420,   1467,  -9852,   5726,
             -4463,   9882,   9855,   6154,  18809,  -7570,  -3482,  -7332,
             -3115,   1022,   1986,   7810,  -4242,   7409,  -4718,  -3562,
              -434,  12746,   1909,   5704,  -2933,  14854,  -7138,  10993,
             17998,   4042,   -832,   7868,  -1003,  14596,  -7154,   4786,
             -3853,  -1195,  11525,   5062,  -8585,  -4646,  -2671,   4514,
              7467,   5717,  -4945,  -5808,  -1625,   1618,   6632,   7261,
              -150,   2363,   5000,   3867,  11388,   3069,   7115, -18664,
            -11462,  -1929,   2188,   3601,   4917,  -1916,   2104,  -2801,
             -4024,  -5225,   1019,   2320,  -7145,   6620,  -7998,  -4771,
             -6593,   5967,  -1835,  -1638,  -8879,  -5924,    219,    698,
              5535,    207, -18536,  -7176,  17890,  -4906,  -2600, -11301,
             20243,   2367,    587,  12012,    643,  -7090,   -598,  -2553,
             12345,  -7295,  -4181,  -6335,  -9463,  -5867,   2439,   2532,
              3431,  -7337,  20317,   3717,  10019,   3916,  -1294,  -3027,
              9283,  -2800,  -1864,   1108,  12191,   2620,   3003,   5809])
```

```
[5]: y
```

```
[5]: array([-16508,   2478,   3515,   5364,  -4384,  -6482, -10675,   5372,
              6219,   9170,   6066,  12958,   5320,   4692,   2490, -13918,
               427,  -2340,  -2719,  -3679,  -1055, -15020,  -4362, -14098,
            -16823,   9443,  -9983,  -2793, -12924,   9181,  14357,   8604,
              2475,   1941,  -1032, -10003,  -6795,  -4177,   1531, -21650,
              3147, -11965,   2702,   3689,  -1963,  -6735, -16125,   -680,
             10188,  -4535, -23408,   8503,   4969,  -2346,   1839,  -5496,
              4698,  -8531,    759,  10408,   2790,   9064,  -3052,  -7809,
               -60,   8928,   3731,   7731,   4980,  11867,  -6397,  -1221,
             10542,   8245,   2672,  13680,   6439,   7669,  17691, -18345,
            -21586,    262,  -1048,  10555,  -3763,  -5072,   5933,   5811,
             -5660, -13243, -11868,   1003,   5175,  11692,  11571,    324,
             10985,   5753,    919,   8974, -10931,   8265,   3987,   3548,
             -6452,    169, -10740,  -1539,  15030, -13081,  -3254,   -972,
             -3699,  -4482,  15416,  -4989,   4314,  -2096,   2036,  13738,
             -5444, -13047,  -3933,  -7949,  -9325,   5095,   8396,   -559])
```

```
[6]: # Now create the shares

     # random nonces, values in the same range as the embeddings after scaling
     r = rng.integers(-30000, 30000, 128)
     r
```

```
[6]: array([ 15171, -20268,  -1642,  -7067,  12123,   2691,  24042,   6836,
             23327,  29941,  18876,  16442,  15019, -13634,  -9084,   2347,
            -11805, -11072,  14033,  25948,  17573,  16380, -28445,  22127,
             17116,  29213,   1837, -14380, -28687,  16323,  -2549,  10011,
            -18245, -26726,  11995,   8765,  -7395,  -3899,  -5189,  -5920,
            -29890, -10534,  26611, -19613,   -101,   8116, -13512,   9852,
             -1689,  19731,  21610, -15688, -13517,   5467, -12183, -10634,
              4456,   6386,  12281,   2268,  26098,  20925, -21550, -19598,
             25181,  20171,  26360, -27762,   3311,  -6774, -16094,  -5613,
             11455, -24897,  -1644,  10886,  24448,  -1045,  14236,   7961,
             -3222,  18980,  -6023,  29903, -26093,  -3718,  -2442,  12428,
              4879, -10710,  13391,   1472,  -2457,  21100,  -8597,  23604,
             22967,  15754,   1410, -20259,  15099,  24542,  22472,  -2850,
            -13104,  20732,  -9680,  15732, -20734,  23182,  18968,   4588,
            -28319,  -9073,  27963,  18666, -21330, -17552,  24754, -23255,
            -20565,  29324,   6909,   3671, -28682,  29372, -26733, -21571])
```

```
[7]: # y_1 is the server's share, simply the nonces
     y_1 = r

     # y_0 is the mb's share, it's the nonces XORed with y
     y_0 = np.bitwise_xor(y, r)

     y_0
```

```
[7]: array([-31545, -18054,  -3027,  -3951, -15941,  -5075, -29785,   3656,
             17236,  22311,  24078,  29348,  11875, -10006, -10946, -16247,
            -12216,   8732, -15440, -27395, -16572,  -1368,  32277, -24959,
              -875,  22270,  -8660,  12995,  17013,   7198, -12770,   1671,
            -20208, -28657, -10973,  -1328,   1640,   8042,  -4544,  17294,
            -30859,   1945,  28029, -17142,   1998,  -1531,   2619,  -9436,
             -8533, -23718,  -3846,  -7281, -10150,  -7283, -10426,  15614,
               818, -14753,  11534,   8308,  28436,  29397,  24518,  21005,
            -25191,  27691,  26731, -29251,   8091, -13359,   9761,   4392,
              1425, -16758,  -3100,   8182,  18087,  -6626,  29319, -22706,
             22724,  19234,   5009,  24052,  27486,   7498,  -7845,   9791,
             -1301,   6767,  -6677,   1579,  -7600,  32704,  -3240,  23920,
             29534,  11251,   1557, -27693,  -4170,  32663,  22619,  -1790,
             10780,  20565,   3132, -15223, -27212, -27031, -18094,  -4648,
             24812,  13041,  20739, -23447, -17292,  19616,  26438, -28541,
             17687, -16507,  -5538,  -4444,  21605,  24923, -18593,  22124])
```

```
[8]: # To check that the xoring works, we can XOR y_0 with y_1 and we expect to␣
      ↪obtain y

     y == np.bitwise_xor(y_0, y_1)
```

```
[8]: array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True])
```

[9]:

```
# Let's run the ABY code (I am providing y for verification, it's not needed␣
 ↪nor used in the circuit)

output = run_sfe(x, y, y_0, y_1)
```

[10]:

```
"""inspect the results
After the statistics from ABY I am printing:
- the input x,y and the share (in this case output is from the server,so we see␣
 ↪share y_1)
- the verification results and the circuit results
"""
# Unfortunately, circuit result is incorrect.
print(output.stdout)
```

```
INPUT FILE NAME: input_vecs.txt
OUTPUT FILE NAME:
s_product nvals: 128
s_product bitlen: 64
Online time is distributed as follows:
Bool: local gates: 270.3640000000, interactive gates: 312.9220000000, layer
finish: 206.2940000000
Yao: local gates: 5.5410000000, interactive gates: 4.6240000000, layer finish:
2.3260000000
Yao Rev: local gates: 4.8740000000, interactive gates: 4.4850000000, layer
finish: 2.0570000000
Arith: local gates: 5.6370000000, interactive gates: 4.6430000000, layer finish:
5.1130000000
SPLUT: local gates: 4.9980000000, interactive gates: 4.6110000000, layer finish:
26.7410000000
Communication: 1588.8240000000

Complexities:
Boolean Sharing: ANDs: 5975939 (1-bit) ; Depth: 10990
Total Vec AND: 5975939
Total Non-Vec AND: 5975939
XOR vals: 5896327 gates: 1386811
```

```
Comb gates: 0, CombStruct gates: 0, Perm gates: 0, Subset gates: 24576, Split
gates: 0
Yao: ANDs: 0 ; Depth: 0
Reverse Yao: ANDs: 0 ; Depth: 0
Arithmetic Sharing: MULs: 0 ; Depth: 0
SP-LUT Sharing: OT-gates: Total OT gates = 0; Depth: 1
Total number of gates: 3171644 Total depth: 10990
Timings:
Total =         3577.1810000000 ms
Init =          0.0970000000 ms
CircuitGen =    0.1100000000 ms
Network =       168.2130000000 ms
BaseOTs =       207.7170000000 ms
Setup =         1108.7750000000 ms
OTExtension =   1102.9960000000 ms
Garbling =      0.0010000000 ms
Online =        2468.4050000000 ms

Communication:
Total Sent / Rcv        98625089  bytes / 98626121 bytes
BaseOTs Sent / Rcv      49956  bytes / 49956 bytes
Setup Sent / Rcv        97023145  bytes / 97023145 bytes
OTExtension Sent / Rcv  97023145  bytes / 97023145 bytes
Garbling Sent / Rcv     0  bytes / 0 bytes
Online Sent / Rcv       1601944  bytes / 1602976 bytes
INPUT EMBEDDINGS:
X:
-10589.0000000000, 11578.0000000000, 6160.0000000000, 9286.0000000000,
5420.0000000000, 1467.0000000000, -9852.0000000000, 5726.0000000000,
-4463.0000000000, 9882.0000000000, 9855.0000000000, 6154.0000000000,
18809.0000000000, -7570.0000000000, -3482.0000000000, -7332.0000000000,
-3115.0000000000, 1022.0000000000, 1986.0000000000, 7810.0000000000,
-4242.0000000000, 7409.0000000000, -4718.0000000000, -3562.0000000000,
-434.0000000000, 12746.0000000000, 1909.0000000000, 5704.0000000000,
-2933.0000000000, 14854.0000000000, -7138.0000000000, 10993.0000000000,
17998.0000000000, 4042.0000000000, -832.0000000000, 7868.0000000000,
-1003.0000000000, 14596.0000000000, -7154.0000000000, 4786.0000000000,
-3853.0000000000, -1195.0000000000, 11525.0000000000, 5062.0000000000,
-8585.0000000000, -4646.0000000000, -2671.0000000000, 4514.0000000000,
7467.0000000000, 5717.0000000000, -4945.0000000000, -5808.0000000000,
-1625.0000000000, 1618.0000000000, 6632.0000000000, 7261.0000000000,
-150.0000000000, 2363.0000000000, 5000.0000000000, 3867.0000000000,
11388.0000000000, 3069.0000000000, 7115.0000000000, -18664.0000000000,
-11462.0000000000, -1929.0000000000, 2188.0000000000, 3601.0000000000,
4917.0000000000, -1916.0000000000, 2104.0000000000, -2801.0000000000,
-4024.0000000000, -5225.0000000000, 1019.0000000000, 2320.0000000000,
-7145.0000000000, 6620.0000000000, -7998.0000000000, -4771.0000000000,
-6593.0000000000, 5967.0000000000, -1835.0000000000, -1638.0000000000,
-8879.0000000000, -5924.0000000000, 219.0000000000, 698.0000000000,
5535.0000000000, 207.0000000000, -18536.0000000000, -7176.0000000000,
17890.0000000000, -4906.0000000000, -2600.0000000000, -11301.0000000000,
20243.0000000000, 2367.0000000000, 587.0000000000, 12012.0000000000,
643.0000000000, -7090.0000000000, -598.0000000000, -2553.0000000000,
12345.0000000000, -7295.0000000000, -4181.0000000000, -6335.0000000000,
-9463.0000000000, -5867.0000000000, 2439.0000000000, 2532.0000000000,
3431.0000000000, -7337.0000000000, 20317.0000000000, 3717.0000000000,
10019.0000000000, 3916.0000000000, -1294.0000000000, -3027.0000000000,
9283.0000000000, -2800.0000000000, -1864.0000000000, 1108.0000000000,
12191.0000000000, 2620.0000000000, 3003.0000000000, 5809.0000000000,
Y:
-16508.0000000000, 2478.0000000000, 3515.0000000000, 5364.0000000000,
-4384.0000000000, -6482.0000000000, -10675.0000000000, 5372.0000000000,
6219.0000000000, 9170.0000000000, 6066.0000000000, 12958.0000000000,
5320.0000000000, 4692.0000000000, 2490.0000000000, -13918.0000000000,
427.0000000000, -2340.0000000000, -2719.0000000000, -3679.0000000000,
-1055.0000000000, -15020.0000000000, -4362.0000000000, -14098.0000000000,
-16823.0000000000, 9443.0000000000, -9983.0000000000, -2793.0000000000,
-12924.0000000000, 9181.0000000000, 14357.0000000000, 8604.0000000000,
2475.0000000000, 1941.0000000000, -1032.0000000000, -10003.0000000000,
-6795.0000000000, -4177.0000000000, 1531.0000000000, -21650.0000000000,
3147.0000000000, -11965.0000000000, 2702.0000000000, 3689.0000000000,
-1963.0000000000, -6735.0000000000, -16125.0000000000, -680.0000000000,
10188.0000000000, -4535.0000000000, -23408.0000000000, 8503.0000000000,
4969.0000000000, -2346.0000000000, 1839.0000000000, -5496.0000000000,
4698.0000000000, -8531.0000000000, 759.0000000000, 10408.0000000000,
2790.0000000000, 9064.0000000000, -3052.0000000000, -7809.0000000000,
-60.0000000000, 8928.0000000000, 3731.0000000000, 7731.0000000000,
4980.0000000000, 11867.0000000000, -6397.0000000000, -1221.0000000000,
10542.0000000000, 8245.0000000000, 2672.0000000000, 13680.0000000000,
6439.0000000000, 7669.0000000000, 17691.0000000000, -18345.0000000000,
-21586.0000000000, 262.0000000000, -1048.0000000000, 10555.0000000000,
-3763.0000000000, -5072.0000000000, 5933.0000000000, 5811.0000000000,
-5660.0000000000, -13243.0000000000, -11868.0000000000, 1003.0000000000,
5175.0000000000, 11692.0000000000, 11571.0000000000, 324.0000000000,
10985.0000000000, 5753.0000000000, 919.0000000000, 8974.0000000000,
-10931.0000000000, 8265.0000000000, 3987.0000000000, 3548.0000000000,
-6452.0000000000, 169.0000000000, -10740.0000000000, -1539.0000000000,
15030.0000000000, -13081.0000000000, -3254.0000000000, -972.0000000000,
-3699.0000000000, -4482.0000000000, 15416.0000000000, -4989.0000000000,
4314.0000000000, -2096.0000000000, 2036.0000000000, 13738.0000000000,
-5444.0000000000, -13047.0000000000, -3933.0000000000, -7949.0000000000,
-9325.0000000000, 5095.0000000000, 8396.0000000000, -559.0000000000,
SHARE:
15171.0000000000, -20268.0000000000, -1642.0000000000, -7067.0000000000,
12123.0000000000, 2691.0000000000, 24042.0000000000, 6836.0000000000,
23327.0000000000, 29941.0000000000, 18876.0000000000, 16442.0000000000,
15019.0000000000, -13634.0000000000, -9084.0000000000, 2347.0000000000,
-11805.0000000000, -11072.0000000000, 14033.0000000000, 25948.0000000000,
17573.0000000000, 16380.0000000000, -28445.0000000000, 22127.0000000000,
17116.0000000000, 29213.0000000000, 1837.0000000000, -14380.0000000000,
-28687.0000000000, 16323.0000000000, -2549.0000000000, 10011.0000000000,
-18245.0000000000, -26726.0000000000, 11995.0000000000, 8765.0000000000,
-7395.0000000000, -3899.0000000000, -5189.0000000000, -5920.0000000000,
-29890.0000000000, -10534.0000000000, 26611.0000000000, -19613.0000000000,
-101.0000000000, 8116.0000000000, -13512.0000000000, 9852.0000000000,
-1689.0000000000, 19731.0000000000, 21610.0000000000, -15688.0000000000,
-13517.0000000000, 5467.0000000000, -12183.0000000000, -10634.0000000000,
4456.0000000000, 6386.0000000000, 12281.0000000000, 2268.0000000000,
26098.0000000000, 20925.0000000000, -21550.0000000000, -19598.0000000000,
25181.0000000000, 20171.0000000000, 26360.0000000000, -27762.0000000000,
3311.0000000000, -6774.0000000000, -16094.0000000000, -5613.0000000000,
11455.0000000000, -24897.0000000000, -1644.0000000000, 10886.0000000000,
24448.0000000000, -1045.0000000000, 14236.0000000000, 7961.0000000000,
-3222.0000000000, 18980.0000000000, -6023.0000000000, 29903.0000000000,
-26093.0000000000, -3718.0000000000, -2442.0000000000, 12428.0000000000,
4879.0000000000, -10710.0000000000, 13391.0000000000, 1472.0000000000,
-2457.0000000000, 21100.0000000000, -8597.0000000000, 23604.0000000000,
22967.0000000000, 15754.0000000000, 1410.0000000000, -20259.0000000000,
15099.0000000000, 24542.0000000000, 22472.0000000000, -2850.0000000000,
-13104.0000000000, 20732.0000000000, -9680.0000000000, 15732.0000000000,
-20734.0000000000, 23182.0000000000, 18968.0000000000, 4588.0000000000,
-28319.0000000000, -9073.0000000000, 27963.0000000000, 18666.0000000000,
-21330.0000000000, -17552.0000000000, 24754.0000000000, -23255.0000000000,
-20565.0000000000, 29324.0000000000, 6909.0000000000, 3671.0000000000,
-28682.0000000000, 29372.0000000000, -26733.0000000000, -21571.0000000000,

VERIFICATION:
x dot y: 1818201534.0000000000
norm(x): 85596.0019802327
norm(y): 98478.3852781919
cos sim: 0.7843012538
CIRCUIT RESULTS:
x dot share: 0.0000000000
norm(x) : 85596.0019802327
norm(share): 0.0000000000
cos sim: -inf
```

[11]:

```
# Since we are getting 'inf' values, I scale by a smaller amount and see if␣
  ↪that helps
# Let's go extreme and round the floats

x = a.round().astype(int)
```

```
y = b.round().astype(int)



x
```

[11]:
```
array([-1,  1,  1,  1,  1,  0, -1,  1,  0,  1,  1,  1,  2, -1,  0, -1,  0,
        0,  0,  1,  0,  1,  0,  0,  0,  1,  0,  1,  0,  1, -1,  1,  2,  0,
        0,  1,  0,  1, -1,  0,  0,  0,  1,  1, -1,  0,  0,  0,  1,  1,  0,
       -1,  0,  0,  1,  1,  0,  0,  0,  0,  1,  0,  1, -2, -1,  0,  0,  0,
        0,  0,  0,  0,  0, -1,  0,  0, -1,  1, -1,  0, -1,  1,  0,  0, -1,
       -1,  0,  0,  1,  0, -2, -1,  2,  0,  0, -1,  2,  0,  0,  1,  0, -1,
        0,  0,  1, -1,  0, -1, -1, -1,  0,  0,  0, -1,  2,  0,  1,  0,  0,
        0,  1,  0,  0,  0,  1,  0,  0,  1])
```

[12]:
```
# Now create the shares, everything the same way as above

# random nonces, values in the same range as the embeddings after scaling
r = rng.integers(-3, 3, 128)

# y_1 is the server's share, simply the nonces
y_1 = r

# y_0 is the mb's share, it's the nonces XORed with y
y_0 = np.bitwise_xor(y, r)

# To check that the xoring works, we can XOR y_0 with y_1 and we expect to␣
 ↪obtain y
y == np.bitwise_xor(y_0, y_1)
```

[12]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True])
```

[13]:
```
# Let's run the ABY code (I am providing y for verification, it's not needed␣
 ↪nor used in the circuit)
```

```
output = run_sfe(x, y, y_0, y_1)
```

[14]:
```
# inspect the results

# Unfortunately, circuit result is incorrect.
print(output.stdout)
```

```
INPUT FILE NAME: input_vecs.txt
OUTPUT FILE NAME:
s_product nvals: 128
s_product bitlen: 64
Online time is distributed as follows:
Bool: local gates: 205.4120000000, interactive gates: 227.6930000000, layer
finish: 147.3740000000
Yao: local gates: 4.1610000000, interactive gates: 3.3780000000, layer finish:
1.6280000000
Yao Rev: local gates: 3.5980000000, interactive gates: 3.3880000000, layer
finish: 1.5550000000
Arith: local gates: 4.2370000000, interactive gates: 3.5320000000, layer finish:
3.5650000000
SPLUT: local gates: 3.7480000000, interactive gates: 3.4630000000, layer finish:
20.2480000000
Communication: 1032.8790000000

Complexities:
Boolean Sharing: ANDs: 5975939 (1-bit) ; Depth: 10990
Total Vec AND: 5975939
Total Non-Vec AND: 5975939
XOR vals: 5896327 gates: 1386811
Comb gates: 0, CombStruct gates: 0, Perm gates: 0, Subset gates: 24576, Split
gates: 0
Yao: ANDs: 0 ; Depth: 0
Reverse Yao: ANDs: 0 ; Depth: 0
Arithmetic Sharing: MULs: 0 ; Depth: 0
SP-LUT Sharing: OT-gates: Total OT gates = 0; Depth: 1
Total number of gates: 3171644 Total depth: 10990
Timings:
Total =         2750.6730000000 ms
Init =          0.0550000000 ms
CircuitGen =    0.0590000000 ms
Network =       0.7410000000 ms
BaseOTs =       203.5020000000 ms
Setup =         1070.6390000000 ms
OTExtension =   1063.8640000000 ms
Garbling =      0.0010000000 ms
Online =        1680.0320000000 ms
```

```
Communication:
Total Sent / Rcv        98625089  bytes / 98626121 bytes
BaseOTs Sent / Rcv      49956  bytes / 49956 bytes
Setup Sent / Rcv        97023145  bytes / 97023145 bytes
OTExtension Sent / Rcv  97023145  bytes / 97023145 bytes
Garbling Sent / Rcv     0  bytes / 0 bytes
Online Sent / Rcv       1601944  bytes / 1602976 bytes
INPUT EMBEDDINGS:
X:
-1.0000000000, 1.0000000000, 1.0000000000, 1.0000000000, 1.0000000000,
0.0000000000, -1.0000000000, 1.0000000000, 0.0000000000, 1.0000000000,
1.0000000000, 1.0000000000, 2.0000000000, -1.0000000000, 0.0000000000,
-1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 1.0000000000, 0.0000000000, 1.0000000000,
-1.0000000000, 1.0000000000, 2.0000000000, 0.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 1.0000000000, -1.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 1.0000000000, 1.0000000000, -1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 1.0000000000,
0.0000000000, -1.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 1.0000000000, -2.0000000000, -1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, -1.0000000000, 0.0000000000,
0.0000000000, -1.0000000000, 1.0000000000, -1.0000000000, 0.0000000000,
-1.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, -1.0000000000,
-1.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
-2.0000000000, -1.0000000000, 2.0000000000, 0.0000000000, 0.0000000000,
-1.0000000000, 2.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, -1.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
-1.0000000000, 0.0000000000, -1.0000000000, -1.0000000000, -1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, -1.0000000000, 2.0000000000,
0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 1.0000000000,
Y:
-2.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
-1.0000000000, -1.0000000000, 1.0000000000, 1.0000000000, 1.0000000000,
1.0000000000, 1.0000000000, 1.0000000000, 0.0000000000, 0.0000000000,
-1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
0.0000000000, -2.0000000000, 0.0000000000, -1.0000000000, -2.0000000000,
1.0000000000, -1.0000000000, 0.0000000000, -1.0000000000, 1.0000000000,
1.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
-1.0000000000, -1.0000000000, 0.0000000000, 0.0000000000, -2.0000000000,
0.0000000000, -1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
-1.0000000000, -2.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
-2.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
-1.0000000000, 0.0000000000, -1.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 1.0000000000, 0.0000000000, -1.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 1.0000000000, 0.0000000000, 1.0000000000,
-1.0000000000, 0.0000000000, 1.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 1.0000000000, 1.0000000000, 2.0000000000, -2.0000000000,
-2.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
-1.0000000000, 1.0000000000, 1.0000000000, -1.0000000000, -1.0000000000,
-1.0000000000, 0.0000000000, 1.0000000000, 1.0000000000, 1.0000000000,
0.0000000000, 1.0000000000, 1.0000000000, 0.0000000000, 1.0000000000,
-1.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, -1.0000000000,
0.0000000000, -1.0000000000, 0.0000000000, 2.0000000000, -1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 2.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
-1.0000000000, -1.0000000000, 0.0000000000, -1.0000000000, -1.0000000000,
1.0000000000, 1.0000000000, 0.0000000000,
SHARE:
0.0000000000, -1.0000000000, 1.0000000000, -3.0000000000, -1.0000000000,
-3.0000000000, 0.0000000000, -2.0000000000, 2.0000000000, 0.0000000000,
-1.0000000000, -1.0000000000, -2.0000000000, 2.0000000000, -2.0000000000,
-2.0000000000, 0.0000000000, -2.0000000000, 1.0000000000, 0.0000000000,
-1.0000000000, 1.0000000000, -2.0000000000, 2.0000000000, -1.0000000000,
-1.0000000000, -3.0000000000, 1.0000000000, 1.0000000000, -2.0000000000,
-1.0000000000, -2.0000000000, -3.0000000000, 0.0000000000, 0.0000000000,
2.0000000000, 2.0000000000, 0.0000000000, 1.0000000000, -2.0000000000,
-3.0000000000, 0.0000000000, -2.0000000000, 2.0000000000, -3.0000000000,
0.0000000000, 1.0000000000, 1.0000000000, -2.0000000000, -2.0000000000,
2.0000000000, -2.0000000000, -3.0000000000, 2.0000000000, 0.0000000000,
-1.0000000000, 2.0000000000, 0.0000000000, -2.0000000000, 1.0000000000,
1.0000000000, 0.0000000000, -3.0000000000, 1.0000000000, 0.0000000000,
2.0000000000, 0.0000000000, 1.0000000000, -1.0000000000, 2.0000000000,
2.0000000000, -3.0000000000, 0.0000000000, -3.0000000000, 1.0000000000,
-2.0000000000, -3.0000000000, -3.0000000000, -2.0000000000, 2.0000000000,
-2.0000000000, 1.0000000000, -1.0000000000, -3.0000000000, 2.0000000000,
-1.0000000000, -1.0000000000, -2.0000000000, -3.0000000000, 2.0000000000,
0.0000000000, -2.0000000000, 2.0000000000, -3.0000000000, 0.0000000000,
0.0000000000, -2.0000000000, 2.0000000000, -3.0000000000, 0.0000000000,
-2.0000000000, -3.0000000000, 0.0000000000, -1.0000000000, -1.0000000000,
2.0000000000, 0.0000000000, 1.0000000000, 0.0000000000, -2.0000000000,
2.0000000000, 1.0000000000, -1.0000000000, -1.0000000000, -3.0000000000,
-1.0000000000, -2.0000000000, -2.0000000000, 1.0000000000, 1.0000000000,
-2.0000000000, 0.0000000000, -3.0000000000, 1.0000000000, 1.0000000000,
-1.0000000000, -3.0000000000, -1.0000000000,

VERIFICATION:
x dot y: 16.0000000000
norm(x): 9.1651513899
norm(y): 10.2956301410
cos sim: 0.8304384386
```

```
CIRCUIT RESULTS:
x dot share: inf
norm(x) : 9.1651513899
norm(share): inf
cos sim: -inf
```

[15]:

```
# I discovered by accident that if the vectors are only 0s and 1s then the␣
 ↪circuit works as expected...
# see example

# some arrays with only 0s or 1s
x = rng.integers(0, 2, 128)
y = rng.integers(0, 2, 128)
r = rng.integers(0, 2, 128)

# create the shares in the same way as before

# y_1 is the server's share, simply the nonces
y_1 = r

# y_0 is the mb's share, it's the nonces XORed with y
y_0 = np.bitwise_xor(y, r)

output = run_sfe(x, y, y_0, y_1)

print(output.stdout)
```

```
INPUT FILE NAME: input_vecs.txt
OUTPUT FILE NAME:
s_product nvals: 128
s_product bitlen: 64
Online time is distributed as follows:
Bool: local gates: 162.2930000000, interactive gates: 179.0490000000, layer
finish: 110.6040000000
Yao: local gates: 3.4560000000, interactive gates: 3.1720000000, layer finish:
1.3480000000
Yao Rev: local gates: 2.9190000000, interactive gates: 2.8540000000, layer
finish: 1.3140000000
Arith: local gates: 3.3060000000, interactive gates: 2.8830000000, layer finish:
2.5250000000
SPLUT: local gates: 3.0230000000, interactive gates: 2.9300000000, layer finish:
16.3380000000
Communication: 744.6760000000

Complexities:
Boolean Sharing: ANDs: 5975939 (1-bit) ; Depth: 10990
Total Vec AND: 5975939
Total Non-Vec AND: 5975939
XOR vals: 5896327 gates: 1386811
Comb gates: 0, CombStruct gates: 0, Perm gates: 0, Subset gates: 24576, Split
gates: 0
Yao: ANDs: 0 ; Depth: 0
Reverse Yao: ANDs: 0 ; Depth: 0
Arithmetic Sharing: MULs: 0 ; Depth: 0
SP-LUT Sharing: OT-gates: Total OT gates = 0; Depth: 1
Total number of gates: 3171644 Total depth: 10990
Timings:
Total =         2321.2710000000 ms
Init =          0.0440000000 ms
CircuitGen =    0.0460000000 ms
Network =       11.9860000000 ms
BaseOTs =       238.3230000000 ms
Setup =         1070.4180000000 ms
OTExtension =   1064.7010000000 ms
Garbling =      0.0020000000 ms
Online =        1250.8520000000 ms

Communication:
Total Sent / Rcv        98625089  bytes / 98626121 bytes
BaseOTs Sent / Rcv      49956  bytes / 49956 bytes
Setup Sent / Rcv        97023145  bytes / 97023145 bytes
OTExtension Sent / Rcv  97023145  bytes / 97023145 bytes
Garbling Sent / Rcv     0  bytes / 0 bytes
Online Sent / Rcv       1601944  bytes / 1602976 bytes
INPUT EMBEDDINGS:
X:
1.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000, 1.0000000000,
1.0000000000, 1.0000000000, 1.0000000000, 0.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 1.0000000000, 1.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 1.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 1.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
1.0000000000, 1.0000000000, 1.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 1.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
0.0000000000, 1.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 1.0000000000, 1.0000000000, 1.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
1.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
1.0000000000, 1.0000000000, 1.0000000000,
Y:
1.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
1.0000000000, 1.0000000000, 1.0000000000, 0.0000000000, 1.0000000000,
1.0000000000, 1.0000000000, 1.0000000000, 1.0000000000, 0.0000000000,
0.0000000000, 1.0000000000, 1.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 1.0000000000, 1.0000000000, 1.0000000000,
1.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
1.0000000000, 1.0000000000, 0.0000000000, 1.0000000000, 1.0000000000,
1.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 1.0000000000,
0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 1.0000000000, 1.0000000000, 1.0000000000,
1.0000000000, 1.0000000000, 1.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 1.0000000000, 1.0000000000, 1.0000000000, 1.0000000000,
0.0000000000, 1.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 1.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 1.0000000000, 1.0000000000, 0.0000000000, 1.0000000000,
1.0000000000, 1.0000000000, 1.0000000000, 0.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 1.0000000000, 1.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
1.0000000000, 0.0000000000, 1.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 1.0000000000,
SHARE:
1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
1.0000000000, 1.0000000000, 1.0000000000, 1.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
1.0000000000, 1.0000000000, 1.0000000000, 1.0000000000, 1.0000000000,
1.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
1.0000000000, 1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 1.0000000000,
1.0000000000, 0.0000000000, 1.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
0.0000000000, 1.0000000000, 0.0000000000, 1.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
1.0000000000, 0.0000000000, 1.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, 0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000,
1.0000000000, 1.0000000000, 0.0000000000, 1.0000000000, 1.0000000000,
1.0000000000, 1.0000000000, 1.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 1.0000000000, 1.0000000000, 1.0000000000, 1.0000000000,
0.0000000000, 0.0000000000, 1.0000000000, 0.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 1.0000000000, 1.0000000000,
0.0000000000, 1.0000000000, 1.0000000000,

VERIFICATION:
x dot y: 30.0000000000
norm(x): 7.3484692283
norm(y): 8.3066238629
cos sim: 0.5085268128
CIRCUIT RESULTS:
x dot share: 30.0000000000
norm(x) : 7.3484692283
norm(share): 8.3066238629
cos sim: 0.5085268128
```