EE278 Mini project 2

Deekshith Krishnegowda

San Jose State University

EE278

Contents

[Mini Project 2 Report 2](#_Toc527990662)

[Introduction 2](#_Toc527990663)

[1. Parametrized fixed-point representation 2](#_Toc527990664)

[1.1 Testcase 1 2](#_Toc527990665)

[1.2 Testcase 2 3](#_Toc527990666)

[1.3 Screenshot 4](#_Toc527990667)

[2. Mu-law representation 5](#_Toc527990668)

[2.1 Testcase 1 5](#_Toc527990669)

[2.2 Testcase 2 5](#_Toc527990670)

[2.3 Screenshot 5](#_Toc527990671)

[3. Floating Point representation 6](#_Toc527990672)

[3.1 Testcase 1 6](#_Toc527990673)

[3.2 Testcase 2 6](#_Toc527990674)

[3.3 Screenshot 6](#_Toc527990675)

[4. Python code 7](#_Toc527990676)

[4.1 Fixed Point parameterized 7](#_Toc527990677)

[4.2 Mu-law 10](#_Toc527990678)

[4.3 Floating Point Half Precision 15](#_Toc527990679)

# Mini Project 2 Report

## Introduction

In this mini project, I have made used of previously developed bit accurate model for addition and multiplication of floating point, fixed point and mu-law numbers. The input activation point is weighed summed with theta point which are the trained neuron weights in actual model to obtain neuron activation point. The weighted sum in then passed through a function, in this case ReLU function, to obtain final activation point.

## Parametrized fixed-point representation

The input activation point and thetha values are given to the code through a an excel file. The code reads the excel file and then performs weighted sum operation on it. The code then passes the results through an ReLU function to get final activation point. Previously developed fixed point number representation model is used here to display the activation point of each neuron in fixed point format.

### 1.1 Testcase 1





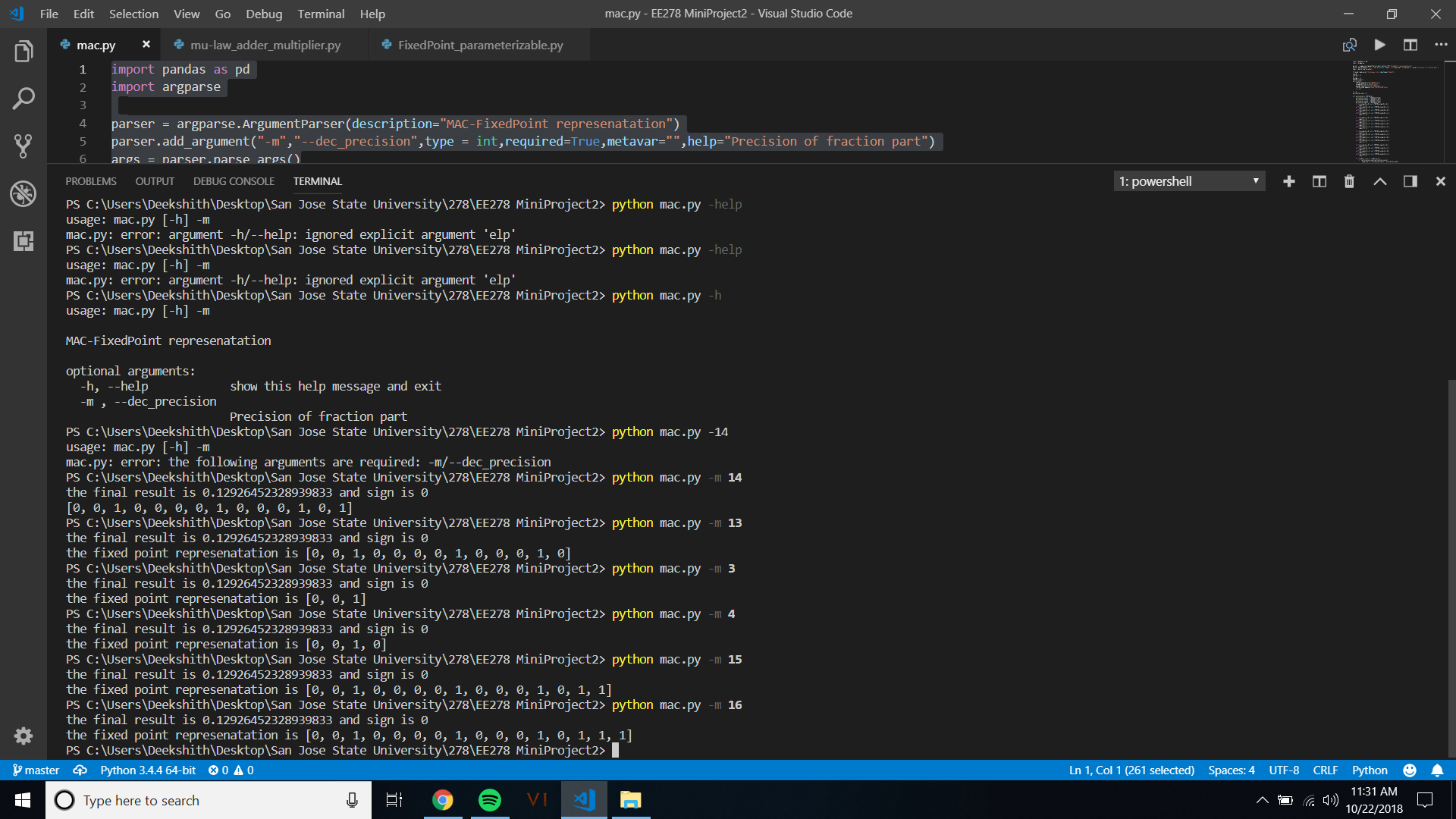
### 1.2 Testcase 2





The input parameter is passed through the commandline. I have made use of argeparse package for this.

### 1.3 Screenshot



## Mu-law representation

The input activation point and thetha values are given to the code through a an excel file. The code reads the excel file and then performs weighted sum operation on it. The code then passes the results through an ReLU function to get final activation point. Previously developed Mu-law number representation model is used here to display the activation point of each neuron in fixed point format.

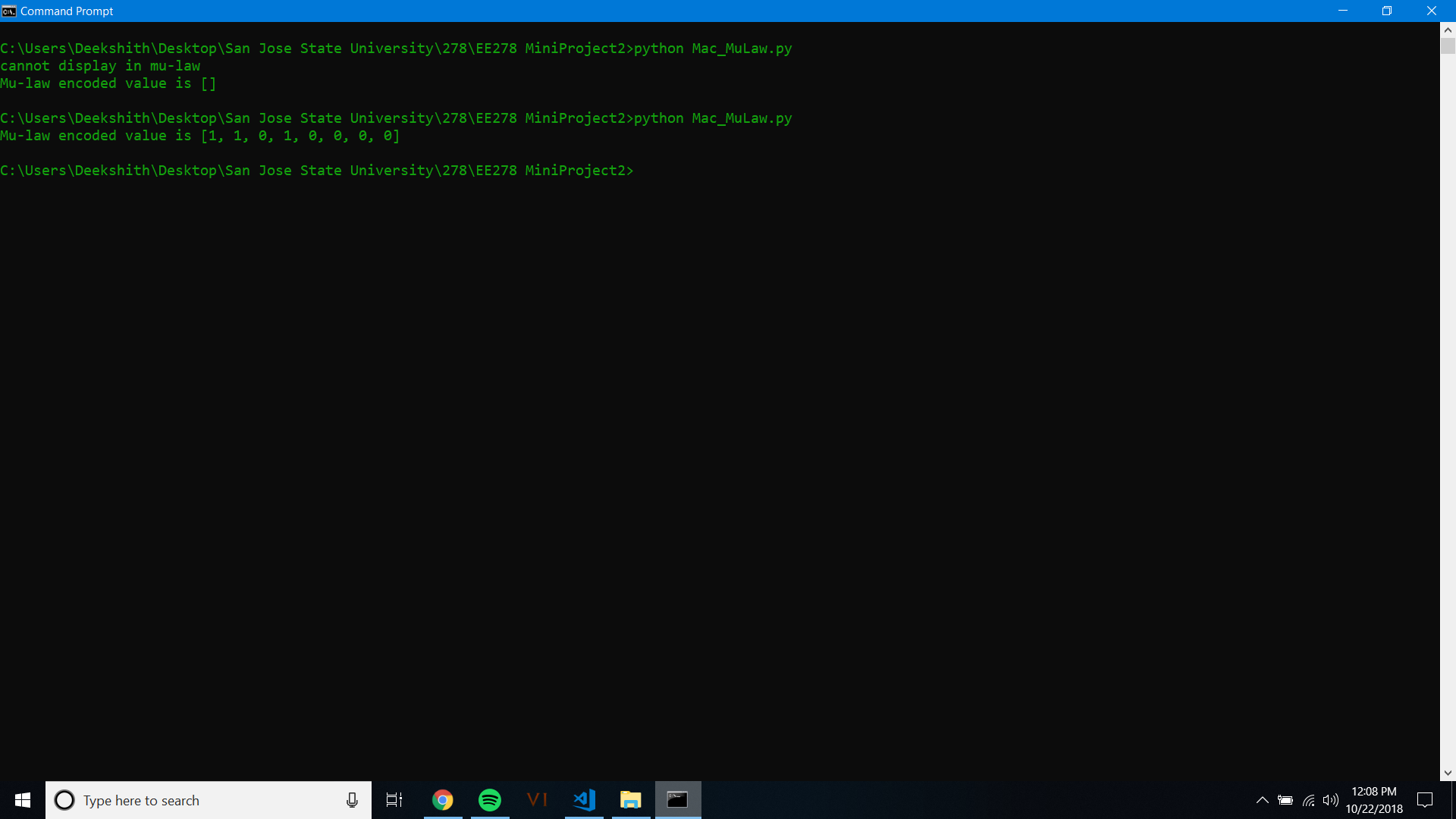
### 2.1 Testcase 1

The activation point after passing through ReLU function is zero. Hence Mu-law representation is zero.

### 2.2 Testcase 2

The activation point after passing through ReLU function is 0.12926452328939833. Hence Mu-law representation is [1, 1, 0, 1, 0, 0, 0, 0].

### 2.3 Screenshot



## Floating Point representation

The input activation point and thetha values are given to the code through a an excel file. The code reads the excel file and then performs weighted sum operation on it. The code then passes the results through an ReLU function to get final activation point. Previously developed floating point half precision number representation model is used here to display the activation point of each neuron in fixed point format.

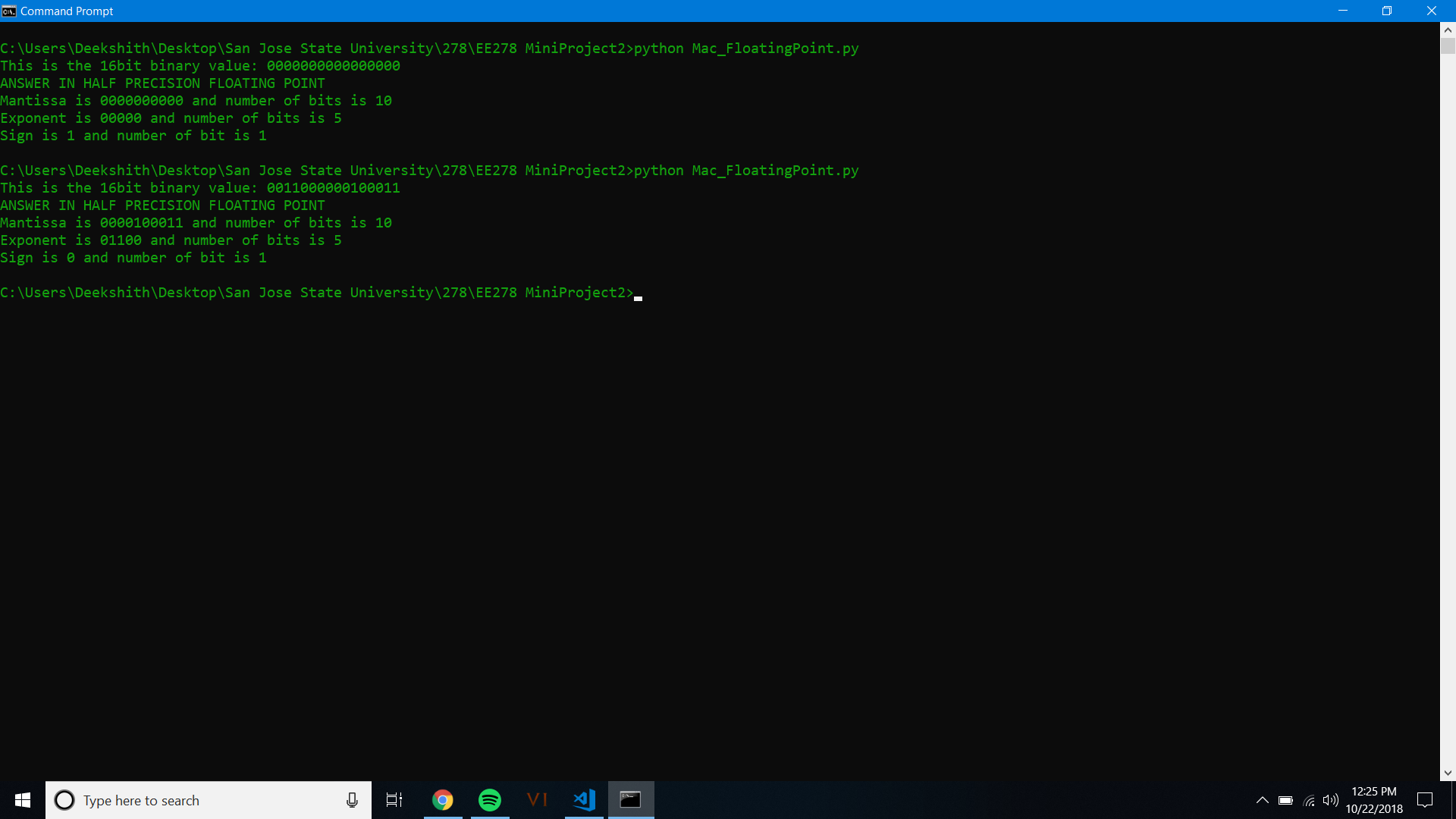
### 3.1 Testcase 1

The activation point after passing through ReLU function is zero. Hence sign, mantissa and exponent are all zero.

### 3.2 Testcase 2

The activation point after passing through ReLU function is 0.12926452328939833. Hence Mantissa is 0000100011, exponent is 01100 and sign is 0. The sixteen-bit number is 0011000000100011.

### 3.3 Screenshot



## Python code

### 4.1 Fixed Point parameterized

import pandas as pd

import argparse

parser = argparse.ArgumentParser(description="MAC-FixedPoint represenatation")

parser.add\_argument("-m","--dec\_precision",type = int,required=True,metavar="",help="Precision of fraction part")

args = parser.parse\_args()

# the input is passed through the commandline argument. I have used argparse package.

# the input parameter is the fixed point parameter

file=pd.read\_excel("testcases.xlsx",sheetname="Sheet3")

i = 0

THETHA = []

A = []

A\_sign = []

THETHA\_sign=[]

while (i<4):

THETHA.append(file["THETHA"][i])

A.append(file["A\_SCALED"][i])

A\_sign.append(file["A\_SIGN"][i])

THETHA\_sign.append(file["THETHA\_SIGN"][i])

i = i+1

# the input activation point and the thetha values are entered in the excel sheet. The excel sheet is read from

# using the panda package. The scaled values of input activation point and thetha are then put inside a list

i = 0

activation\_sum = 0

# weighted multiplication operation is done on input activation point and thetha.

def activation(i,THETHA,A):

activation\_sum\_1 = THETHA[0]\*A[0]

activation\_sum\_2 = THETHA[1]\*A[1]

activation\_sum\_3 = THETHA[2]\*A[2]

activation\_sum\_4 = THETHA[3]\*A[3]

if ((A\_sign[0]==0) and (THETHA\_sign[0]==0)):

sign\_1 = 0

elif((A\_sign[0]==0) and (THETHA\_sign[0]==1)):

sign\_1 = 1

elif((A\_sign[0]==1) and (THETHA\_sign[0]==0)):

sign\_1 = 1

elif((A\_sign[0]==1) and (THETHA\_sign[0]==1)):

sign\_1 = 0

if ((A\_sign[1]==0) and (THETHA\_sign[1]==0)):

sign\_2 = 0

elif((A\_sign[1]==0) and (THETHA\_sign[1]==1)):

sign\_2 = 1

elif((A\_sign[1]==1) and (THETHA\_sign[1]==0)):

sign\_2 = 1

elif((A\_sign[1]==1) and (THETHA\_sign[1]==1)):

sign\_2 = 0

if ((A\_sign[2]==0) and (THETHA\_sign[2]==0)):

sign\_3 = 0

elif((A\_sign[2]==0) and (THETHA\_sign[2]==1)):

sign\_3 = 1

elif((A\_sign[2]==1) and (THETHA\_sign[2]==0)):

sign\_3 = 1

elif((A\_sign[2]==1) and (THETHA\_sign[2]==1)):

sign\_3 = 0

if ((A\_sign[3]==0) and (THETHA\_sign[3]==0)):

sign\_4 = 0

elif((A\_sign[3]==0) and (THETHA\_sign[3]==1)):

sign\_4 = 1

elif((A\_sign[3]==1) and (THETHA\_sign[3]==0)):

sign\_4 = 1

elif((A\_sign[3]==1) and (THETHA\_sign[3]==1)):

sign\_4 = 0

if ((sign\_1 ==1) or (sign\_2==1)):

if (activation\_sum\_1>activation\_sum\_2):

temp\_sum\_1 = activation\_sum\_1 - activation\_sum\_2

temp\_sign\_1 = sign\_1

else:

temp\_sum\_1 = activation\_sum\_2 - activation\_sum\_1

temp\_sign\_1 = sign\_2

else:

temp\_sum\_1 = activation\_sum\_1 + activation\_sum\_2

temp\_sign\_1 = 0

if ((sign\_3 ==1) or (sign\_4==1)):

if (activation\_sum\_3>activation\_sum\_4):

temp\_sum\_2 = activation\_sum\_3 - activation\_sum\_4

temp\_sign\_2 = sign\_3

else:

temp\_sum\_2 = activation\_sum\_4 - activation\_sum\_3

temp\_sign\_2 = sign\_4

else:

temp\_sum\_2 = activation\_sum\_3 + activation\_sum\_4

temp\_sign\_2 = 0

if ((temp\_sign\_1==1) or (temp\_sign\_2==1)):

if (temp\_sum\_1>temp\_sum\_2):

activation\_sum = temp\_sum\_1 - temp\_sum\_2

sign = temp\_sign\_1

else:

activation\_sum = temp\_sum\_2 - temp\_sum\_1

sign = temp\_sign\_2

else:

activation\_sum = temp\_sum\_1 + temp\_sum\_2

sign = 0

return activation\_sum,sign

def ReLU (value,sign):

if (sign ==1):

final\_result = 0

else:

final\_result = value

return final\_result

# the neuron activation number is then passed through ReLu function. If the neuron activation point is negative, the the

# the output if ReLU function is zero.

ReLU\_input,final\_sign = activation(i,THETHA,A)

ReLU\_result = ReLU(ReLU\_input,final\_sign)

# the nueron activation point is the converted into appropriate format. In this case it is fixed point number.

# A parameter should be passed to fixed point.

d\_80 = ReLU\_result

def dec\_int\_conv(d\_80):

temp=2\*d\_80

if(temp>=1):

ret\_bit = 1

ret\_sub = temp-1

return ret\_bit,ret\_sub

else:

ret\_bit = 0

ret\_sub = temp

return ret\_bit,ret\_sub

dec\_list\_80 = []

n\_80 = 0

while n\_80<=args.dec\_precision:

b\_80,a\_80=dec\_int\_conv(d\_80)

d\_80=a\_80

dec\_list\_80.insert(n\_80,b\_80)

n\_80=n\_80+1

del dec\_list\_80[-1]

# The final result in fixed point format is displayed.

if(ReLU\_result>0):

print("the final result is {} and sign is {}".format(ReLU\_result,final\_sign))

print("the fixed point represenatation is {}".format(dec\_list\_80))

else:

print ("number quantized to 0 after ReLU function")

### 4.2 Mu-law

import pandas as pd

file=pd.read\_excel("testcases.xlsx",sheetname="Sheet3")

i = 0

THETHA = []

A = []

A\_sign = []

THETHA\_sign=[]

while (i<4):

THETHA.append(file["THETHA"][i])

A.append(file["A\_SCALED"][i])

A\_sign.append(file["A\_SIGN"][i])

THETHA\_sign.append(file["THETHA\_SIGN"][i])

i = i+1

# the input activation point and the thetha values are entered in the excel sheet. The excel sheet is read from

# using the panda package. The scaled values of input activation point and thetha are then put inside a list

i = 0

activation\_sum = 0

# weighted multiplication operation is done on input activation point and thetha.

def activation(i,THETHA,A):

activation\_sum\_1 = THETHA[0]\*A[0]

activation\_sum\_2 = THETHA[1]\*A[1]

activation\_sum\_3 = THETHA[2]\*A[2]

activation\_sum\_4 = THETHA[3]\*A[3]

if ((A\_sign[0]==0) and (THETHA\_sign[0]==0)):

sign\_1 = 0

elif((A\_sign[0]==0) and (THETHA\_sign[0]==1)):

sign\_1 = 1

elif((A\_sign[0]==1) and (THETHA\_sign[0]==0)):

sign\_1 = 1

elif((A\_sign[0]==1) and (THETHA\_sign[0]==1)):

sign\_1 = 0

if ((A\_sign[1]==0) and (THETHA\_sign[1]==0)):

sign\_2 = 0

elif((A\_sign[1]==0) and (THETHA\_sign[1]==1)):

sign\_2 = 1

elif((A\_sign[1]==1) and (THETHA\_sign[1]==0)):

sign\_2 = 1

elif((A\_sign[1]==1) and (THETHA\_sign[1]==1)):

sign\_2 = 0

if ((A\_sign[2]==0) and (THETHA\_sign[2]==0)):

sign\_3 = 0

elif((A\_sign[2]==0) and (THETHA\_sign[2]==1)):

sign\_3 = 1

elif((A\_sign[2]==1) and (THETHA\_sign[2]==0)):

sign\_3 = 1

elif((A\_sign[2]==1) and (THETHA\_sign[2]==1)):

sign\_3 = 0

if ((A\_sign[3]==0) and (THETHA\_sign[3]==0)):

sign\_4 = 0

elif((A\_sign[3]==0) and (THETHA\_sign[3]==1)):

sign\_4 = 1

elif((A\_sign[3]==1) and (THETHA\_sign[3]==0)):

sign\_4 = 1

elif((A\_sign[3]==1) and (THETHA\_sign[3]==1)):

sign\_4 = 0

if ((sign\_1 ==1) or (sign\_2==1)):

if (activation\_sum\_1>activation\_sum\_2):

temp\_sum\_1 = activation\_sum\_1 - activation\_sum\_2

temp\_sign\_1 = sign\_1

else:

temp\_sum\_1 = activation\_sum\_2 - activation\_sum\_1

temp\_sign\_1 = sign\_2

else:

temp\_sum\_1 = activation\_sum\_1 + activation\_sum\_2

temp\_sign\_1 = 0

if ((sign\_3 ==1) or (sign\_4==1)):

if (activation\_sum\_3>activation\_sum\_4):

temp\_sum\_2 = activation\_sum\_3 - activation\_sum\_4

temp\_sign\_2 = sign\_3

else:

temp\_sum\_2 = activation\_sum\_4 - activation\_sum\_3

temp\_sign\_2 = sign\_4

else:

temp\_sum\_2 = activation\_sum\_3 + activation\_sum\_4

temp\_sign\_2 = 0

if ((temp\_sign\_1==1) or (temp\_sign\_2==1)):

if (temp\_sum\_1>temp\_sum\_2):

activation\_sum = temp\_sum\_1 - temp\_sum\_2

sign = temp\_sign\_1

else:

activation\_sum = temp\_sum\_2 - temp\_sum\_1

sign = temp\_sign\_2

else:

activation\_sum = temp\_sum\_1 + temp\_sum\_2

sign = 0

return activation\_sum,sign

def ReLU (value,sign):

if (sign ==1):

final\_result = 0

else:

final\_result = value

return final\_result

# the neuron activation number is then passed through ReLu function. If the neuron activation point is negative, the the

# the output if ReLU function is zero.

ReLU\_input,final\_sign = activation(i,THETHA,A)

ReLU\_result = ReLU(ReLU\_input,final\_sign)

# the nueron activation point is the converted into appropriate format. In this case it is Mu-Law.

if final\_sign == 1:

final\_sign\_Mu = 0

else:

final\_sign\_Mu = 1

list1\_80 = [0,0,0,0,0,0,0,1]

list2\_80 = [0,0,0,0,0,0,1]

list3\_80 = [0,0,0,0,0,1]

list4\_80 = [0,0,0,0,1]

list5\_80 = [0,0,0,1]

list6\_80 = [0,0,1]

list7\_80 = [0,1]

list8\_80 = [1]

# the above list is used as the encoding table

list1\_1\_80 = [0,0,0]

list2\_1\_80 = [0,0,1]

list3\_1\_80 = [0,1,0]

list4\_1\_80 = [0,1,1]

list5\_1\_80 = [1,0,0]

list6\_1\_80 = [1,0,1]

list7\_1\_80 = [1,1,0]

list8\_1\_80 = [1,1,1]

# this list is concatinated with some value later.

# this function is used to covert the fractional part which is in decimal into binary.

# This is done because python does not has binary format datatype.

def dec\_int\_conv(d\_80):

temp\_80=2\*d\_80

if(temp\_80>=1):

ret\_bit\_80 = 1

ret\_sub\_80 = temp\_80-1

return ret\_bit\_80,ret\_sub\_80

else:

ret\_bit\_80 = 0

ret\_sub\_80 = temp\_80

return ret\_bit\_80,ret\_sub\_80

dec\_list\_80 = []

n\_80 = 0

dec\_precision = 13

d\_80 = ReLU\_result

dec\_int\_conv(d\_80)

while n\_80<=dec\_precision:

b\_80,a\_80=dec\_int\_conv(d\_80)

d\_80=a\_80

dec\_list\_80.insert(n\_80,b\_80)

n\_80=n\_80+1

#print (dec\_list\_80,type(dec\_list\_80[0]))

dec\_list\_updated\_80 = []

for i\_80 in dec\_list\_80:

if(i\_80==0):

dec\_list\_updated\_80.append(i\_80)

else:

break

if (d\_80<1):

dec\_list\_updated\_80.append(1)

final\_list\_80 = []

if (dec\_list\_80[0:1] == dec\_list\_updated\_80):

final\_list\_80 = list8\_1\_80 + dec\_list\_80[1:5]

final\_list\_80.insert(0, final\_sign\_Mu)

elif (dec\_list\_80[0:2] == dec\_list\_updated\_80):

final\_list\_80 = list7\_1\_80 + dec\_list\_80[2:6]

final\_list\_80.insert(0, final\_sign\_Mu)

elif (dec\_list\_80[0:3] == dec\_list\_updated\_80):

final\_list\_80 = list6\_1\_80 + dec\_list\_80[3:7]

final\_list\_80.insert(0, final\_sign\_Mu)

elif (dec\_list\_80[0:4] == dec\_list\_updated\_80):

final\_list\_80 = list5\_1\_80 + dec\_list\_80[4:8]

final\_list\_80.insert(0, final\_sign\_Mu)

elif (dec\_list\_80[0:5] == dec\_list\_updated\_80):

final\_list\_80 = list4\_1\_80 + dec\_list\_80[5:9]

final\_list\_80.insert(0, final\_sign\_Mu)

elif (dec\_list\_80[0:6] == dec\_list\_updated\_80):

final\_list\_80 = list3\_1\_80 + dec\_list\_80[6:10]

final\_list\_80.insert(0, final\_sign\_Mu)

elif (dec\_list\_80[0:7] == dec\_list\_updated\_80):

final\_list\_80 = list2\_1\_80 + dec\_list\_80[7:11]

final\_list\_80.insert(0, final\_sign\_Mu)

elif (dec\_list\_80[0:8] == dec\_list\_updated\_80):

final\_list\_80 = list1\_1\_80 + dec\_list\_80[8:12]

final\_list\_80.insert(0, final\_sign\_Mu)

# the results are displayed in the following section.

# After dividing the inputs by 8192, if the results is not comparable with the encoding table, then the number cannot be represented

# using mu-law encoding.

else:

print ("cannot display in mu-law ")

print ("Mu-law encoded value is {}".format(final\_list\_80))

### 4.3 Floating Point Half Precision

import pandas as pd

import numpy as np

file=pd.read\_excel("testcases.xlsx",sheetname="Sheet3")

i = 0

THETHA = []

A = []

A\_sign = []

THETHA\_sign=[]

while (i<4):

THETHA.append(file["THETHA"][i])

A.append(file["A\_SCALED"][i])

A\_sign.append(file["A\_SIGN"][i])

THETHA\_sign.append(file["THETHA\_SIGN"][i])

i = i+1

# the input activation point and the thetha values are entered in the excel sheet. The excel sheet is read from

# using the panda package. The scaled values of input activation point and thetha are then put inside a list

i = 0

activation\_sum = 0

# weighted multiplication operation is done on input activation point and thetha.

def activation(i,THETHA,A):

activation\_sum\_1 = THETHA[0]\*A[0]

activation\_sum\_2 = THETHA[1]\*A[1]

activation\_sum\_3 = THETHA[2]\*A[2]

activation\_sum\_4 = THETHA[3]\*A[3]

if ((A\_sign[0]==0) and (THETHA\_sign[0]==0)):

sign\_1 = 0

elif((A\_sign[0]==0) and (THETHA\_sign[0]==1)):

sign\_1 = 1

elif((A\_sign[0]==1) and (THETHA\_sign[0]==0)):

sign\_1 = 1

elif((A\_sign[0]==1) and (THETHA\_sign[0]==1)):

sign\_1 = 0

if ((A\_sign[1]==0) and (THETHA\_sign[1]==0)):

sign\_2 = 0

elif((A\_sign[1]==0) and (THETHA\_sign[1]==1)):

sign\_2 = 1

elif((A\_sign[1]==1) and (THETHA\_sign[1]==0)):

sign\_2 = 1

elif((A\_sign[1]==1) and (THETHA\_sign[1]==1)):

sign\_2 = 0

if ((A\_sign[2]==0) and (THETHA\_sign[2]==0)):

sign\_3 = 0

elif((A\_sign[2]==0) and (THETHA\_sign[2]==1)):

sign\_3 = 1

elif((A\_sign[2]==1) and (THETHA\_sign[2]==0)):

sign\_3 = 1

elif((A\_sign[2]==1) and (THETHA\_sign[2]==1)):

sign\_3 = 0

if ((A\_sign[3]==0) and (THETHA\_sign[3]==0)):

sign\_4 = 0

elif((A\_sign[3]==0) and (THETHA\_sign[3]==1)):

sign\_4 = 1

elif((A\_sign[3]==1) and (THETHA\_sign[3]==0)):

sign\_4 = 1

elif((A\_sign[3]==1) and (THETHA\_sign[3]==1)):

sign\_4 = 0

if ((sign\_1 ==1) or (sign\_2==1)):

if (activation\_sum\_1>activation\_sum\_2):

temp\_sum\_1 = activation\_sum\_1 - activation\_sum\_2

temp\_sign\_1 = sign\_1

else:

temp\_sum\_1 = activation\_sum\_2 - activation\_sum\_1

temp\_sign\_1 = sign\_2

else:

temp\_sum\_1 = activation\_sum\_1 + activation\_sum\_2

temp\_sign\_1 = 0

if ((sign\_3 ==1) or (sign\_4==1)):

if (activation\_sum\_3>activation\_sum\_4):

temp\_sum\_2 = activation\_sum\_3 - activation\_sum\_4

temp\_sign\_2 = sign\_3

else:

temp\_sum\_2 = activation\_sum\_4 - activation\_sum\_3

temp\_sign\_2 = sign\_4

else:

temp\_sum\_2 = activation\_sum\_3 + activation\_sum\_4

temp\_sign\_2 = 0

if ((temp\_sign\_1==1) or (temp\_sign\_2==1)):

if (temp\_sum\_1>temp\_sum\_2):

activation\_sum = temp\_sum\_1 - temp\_sum\_2

sign = temp\_sign\_1

else:

activation\_sum = temp\_sum\_2 - temp\_sum\_1

sign = temp\_sign\_2

else:

activation\_sum = temp\_sum\_1 + temp\_sum\_2

sign = 0

return activation\_sum,sign

def ReLU (value,sign):

if (sign ==1):

final\_result = 0

else:

final\_result = value

return final\_result

# the neuron activation number is then passed through ReLu function. If the neuron activation point is negative, the the

# the output if ReLU function is zero.

ReLU\_input,final\_sign = activation(i,THETHA,A)

ReLU\_result = ReLU(ReLU\_input,final\_sign)

output\_80 = ReLU\_result

# the nueron activation point is the converted into appropriate format. In this case it is floating point number with half precision.

int16bits\_80 = np.asarray(output\_80, dtype=np.float16).view(np.int16).item()

# print("this is value of int16bits: %d " %int16bits)

#print('{:016b}'.format(int16bits\_80))

# the addition product or multiplication product is converted into half precision floating point using the numpy package.

# the result is in integer format. this is later converted into string because int cannot be indexed in python

y\_80=bin(int16bits\_80)

sbox\_80=''

k\_80=list(y\_80)

x\_80=k\_80[2:]

l\_80=len(x\_80)

noOfZeroes\_80=16-l\_80

for i in range(noOfZeroes\_80):

sbox\_80=sbox\_80+'0'

for j in x\_80:

sbox\_80=sbox\_80+j

print("This is the 16bit binary value: %s " %sbox\_80)

#print (sbox\_80,len(sbox\_80))

#sign\_bit\_80=(sbox\_80[0])

exp\_part\_80=(sbox\_80[1:6])

man\_part\_80=(sbox\_80[6:])

# the mantissa and exponent part are given to exp\_part\_80 and man\_part\_80. This is displayed later.

# this portion calculated the sign of the addition or multiplication operation

# During multiplication , if both the signs are same, then sign of product is positive. Else it is negative.

# During addition , the sign of the largest integer is given to the product.

# this portion displays the mantissa and the exponent portions of the product along with the number of bits

print("ANSWER IN HALF PRECISION FLOATING POINT")

print("Mantissa is {} and number of bits is {}".format((man\_part\_80),len(man\_part\_80)))

print("Exponent is {} and number of bits is {}".format((exp\_part\_80),len(exp\_part\_80)))

print("Sign is {} and number of bit is 1".format(final\_sign))