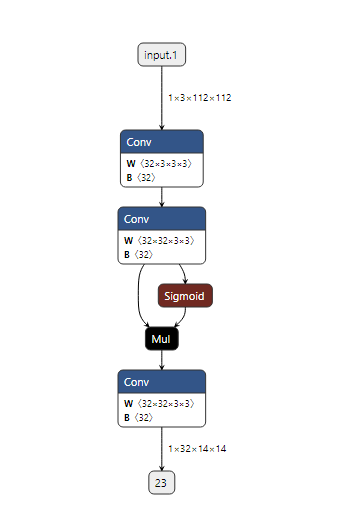
# 后量化查找表实例

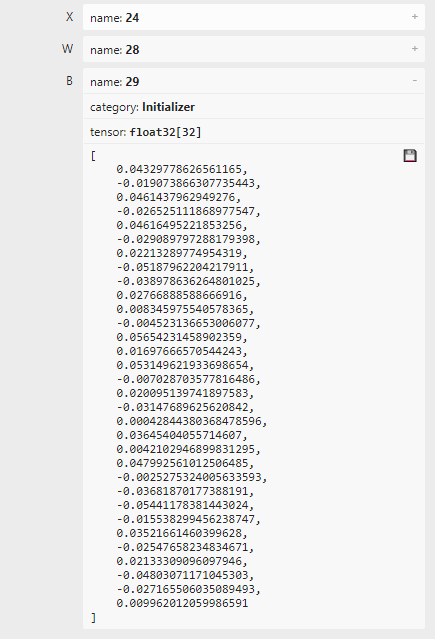
后量化lut替换的激活有mish,hardswish,sigmoid,tanh,swish，实例根据Swish激活实现，

图6为swish的激活函数

|  |
| --- |
| 9 |
| 图6 swish激活函数 |

1.我们构建一个有swish激活的小网络，如下图:

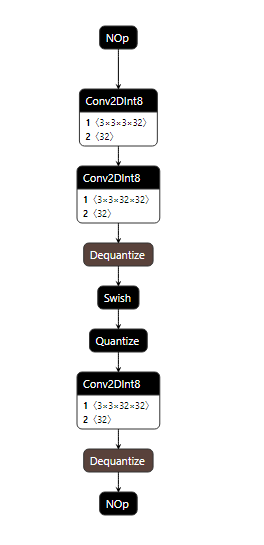
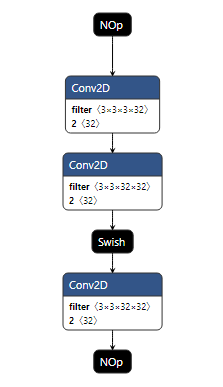




原生bias如下

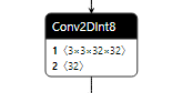
1. 后量化转换：

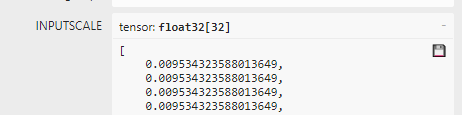
识别成 swish激活 量化：



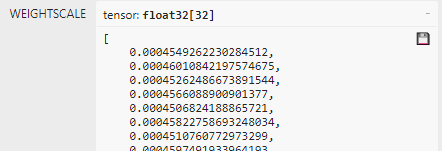
**在这里我们看一些必要的参数：**

量化的卷积：

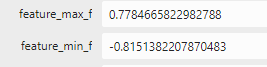


Input\_scale: 输入的反量化scale ：0.00935 input\_offset: 119 输入的offset

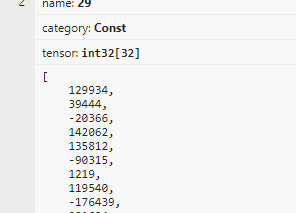
W\_scale: 反量化weight的scale：



当前卷积层输出的min，max统计值：



量化的bias： int32类型



量化的卷积层推理公式 (Input - input\_offset)\*input\_scale @ W\_int \* w\_scale + bias

提出scale：input\_scale\*w\_scale\*[(Input- input\_offset)@W\_int] + bias

量化的bias：过程：

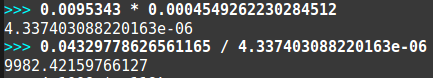
第一步 将bias 提到以上公式[]中：

input\_scale\*w\_scale[(Input- input\_offset)@W\_int + bias /input\_scale\*w\_scale]

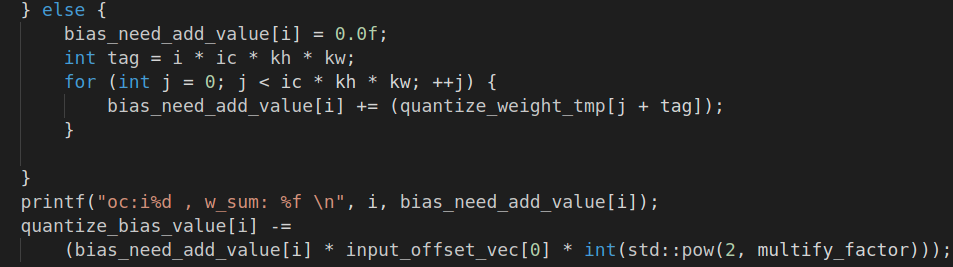
Q\_bias = int(bias /input\_scale\*w\_scale) - input\_offset\*W\_int\_sum(按通道求和)

举例：float bias第一个数 0.043297

0.043297 / (0.0095343 \* 0.00045492）= 9982



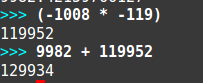
W\_int\_sum: 如下打印第一个数：

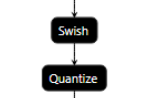


print_w_sum

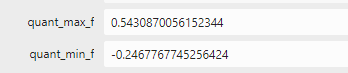
最终的量化Bias:

129934 = 9982 - (input\_offset\*W\_int\_sum) = 9982 - (119 \* -1008)





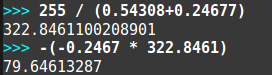
Quantize: 对swish激活统计的feature，min和max值



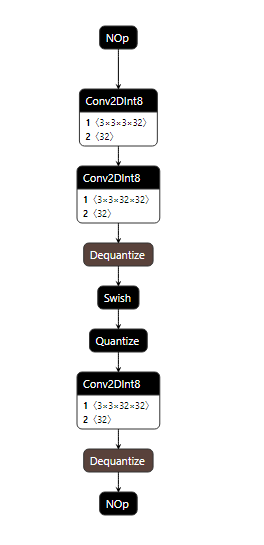
offset

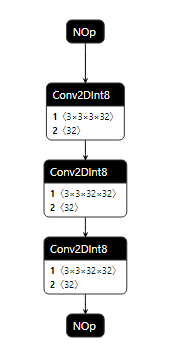
Q\_scale = 255 / (max-min) = 255 / (0.5430 - -0.2467)

Offset: = -(-min \* Q\_scale) =79



优化网络结构：

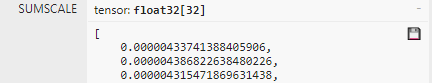




结构优化后卷积的激活类型由：None 变成了swish

多了sum\_scale 属性

jihuo



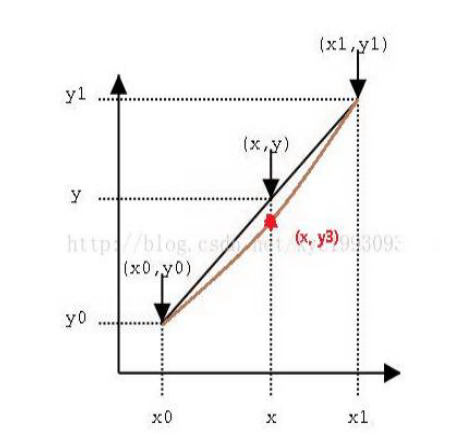
SUMSCALE = input\_scale \* weight , SUMSCALE计算验证图在上面卷积计算量化bias有体现

SUMSCALE的意义相当于卷积结果的Dequantize\_scale

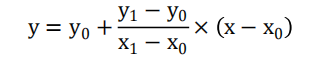
有了以上必要量化参数作基础，接下来是查找表的实现：

（1）理论分析

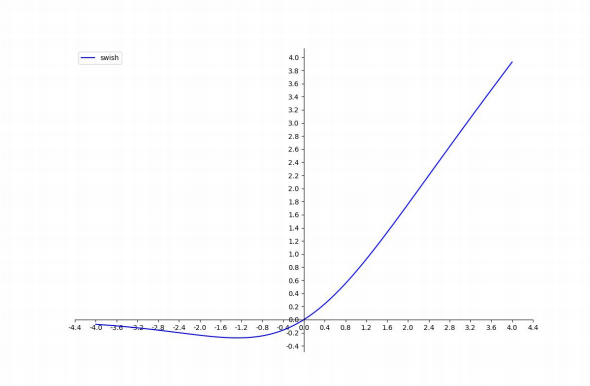
查找表是根据最小值和最大值进行等距切分，输入的值在哪一段再根据左右相邻的值通过线性乘加获取估计值，该操作根据线性差值原理，在一维序列中需要插值的点根据左右相邻的点进行数值估计。估计值计算如图所示；



图中的弧线为模拟复杂激活的一段，x0 ~ x1为激活中切分的一段，x为输入值，y3为待求的真实值，根据线性插值公式可以求出y的值，其中x0,x1,y0,y1已知。计算入公式。



只要切分的足够小，(x0,y0)和(x1,y1)之间就是直线，当(x0,y0)和(x1,y1)距离越近y3的值越趋近y，即y是待求的真实值



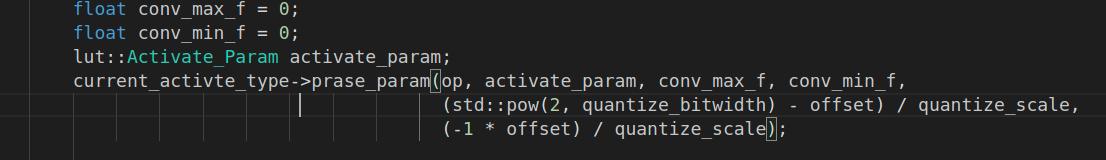
表的生成：

1. 我们要获根据统计的激活输出值的范围获取 输入的范围：真实的x值的范围

y’ ={y0,y1,y2,...............y(table\_num)};

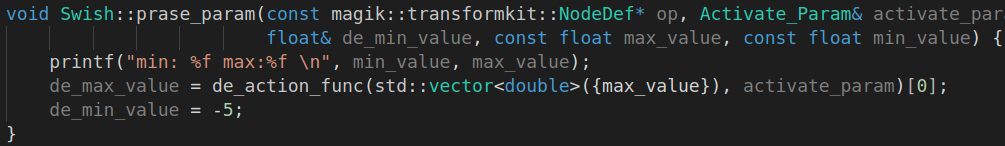
dy = {y1 - y0,y2 -y1,y3-y2,...............y(table\_num+1)-y(table\_num)};

代码实现：



Pow(2,quantize\_bitwidth) - offset / quantize\_scale 相当于反量化激活输出的最大值

-1 \* offset/ quantize\_scale 反量化激活输出的最小值：



打印结果：

min_max

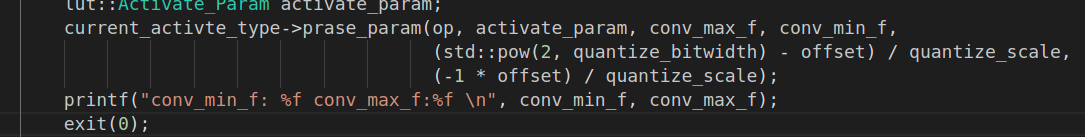
这里最小值默认设置成了-5 根据swish激活函数图可以看出 -5 为最小数是比较合理的。

根据最大值 我们反向求出输入激活前的最小值和最大值：

为**-5.0， 1.0**

这里因为对输入的值做了向上取整，所以输入的x都为整数

up_int

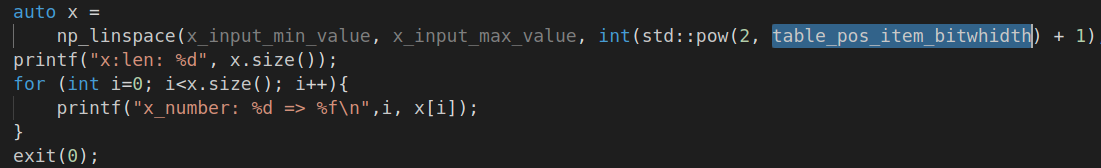


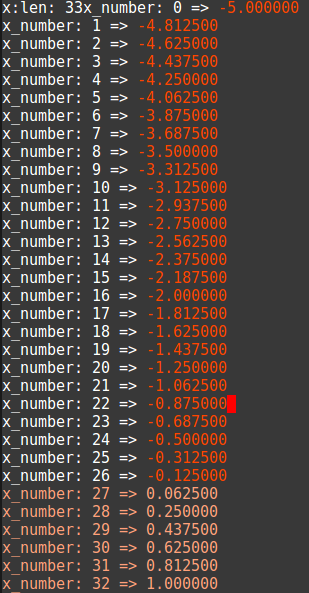
minma

到此第一步，我们求出了输入的激活前x的值为**-5，1**

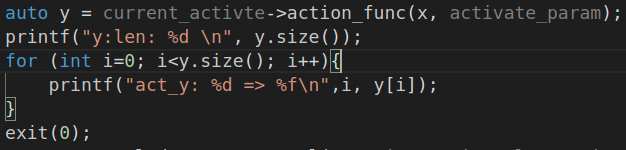
1. 对x的输入范围 进行等距切分成32份，33个数：

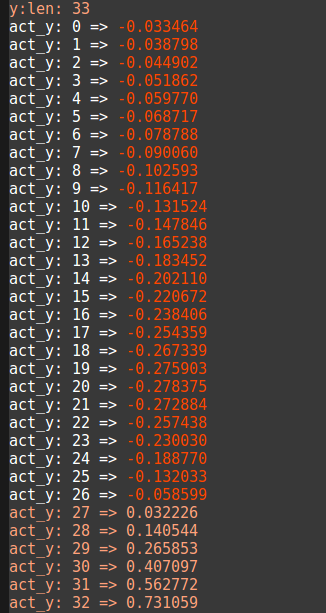
Table\_pos\_item\_bitwhidth = 5





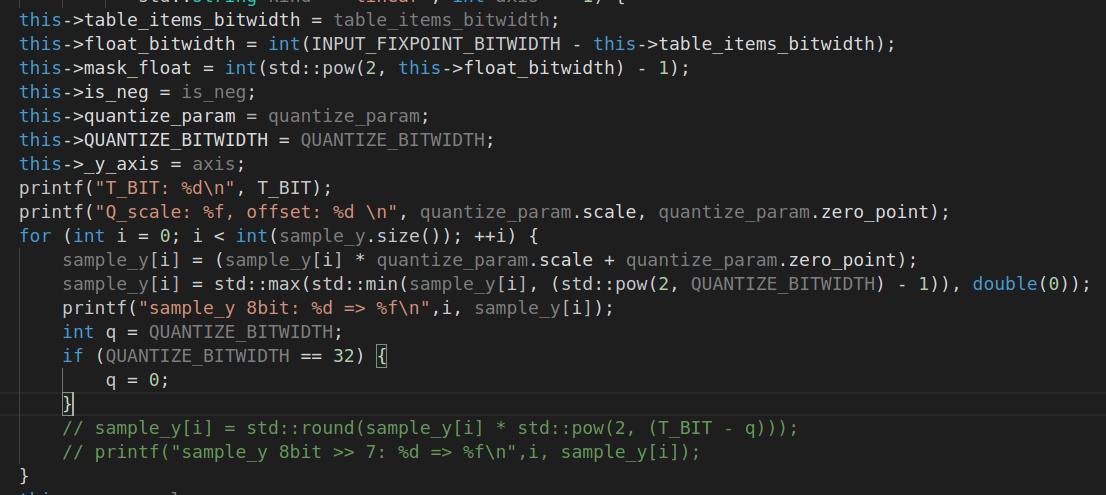
1. 对所有x值求对应的激活结果：



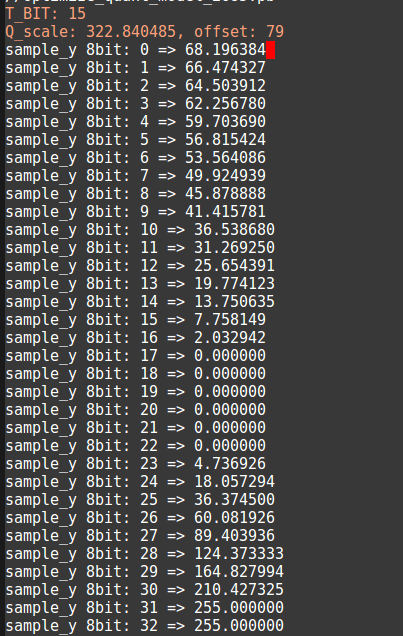


1. 对y值求对应的量化值：

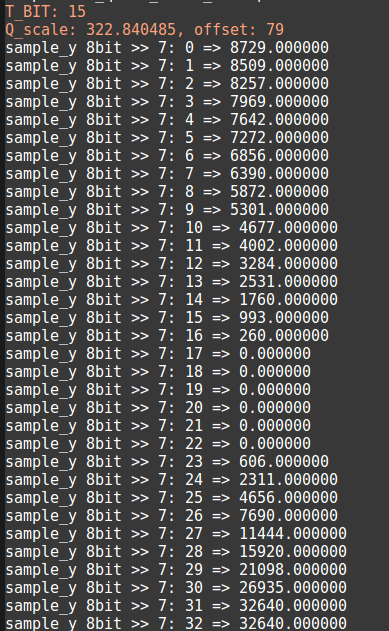
对以上求得激活的y值 求其对应的量化值，作如下打印：



先量化到8bit，并不四舍五入，保留小数部分，体高精度：



定点化y操作：将y值左移7 位数

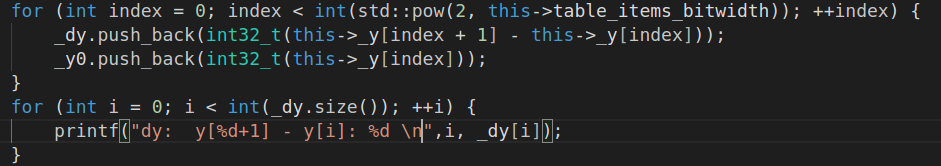


以上我们得到了**定点化的 y值**

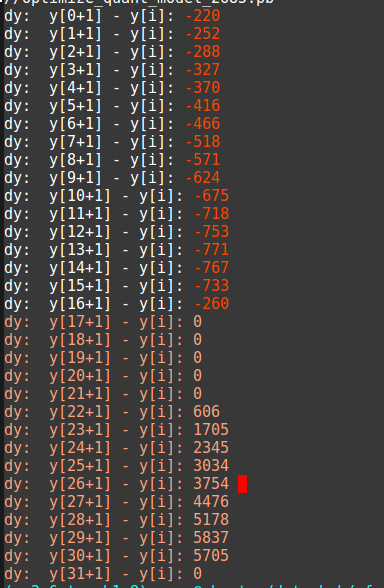
生成表 y， dy：

y’ ={y0,y1,y2,...............y(table\_num)};

dy = {y1 - y0,y2 -y1,y3-y2,...............y(table\_num+1)-y(table\_num)};



打印如下： 32个值

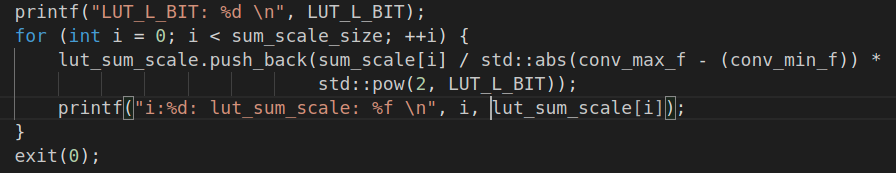


到这里我们知道表的y值是做了一次定点化操作的，左移了 15 -8 =7位

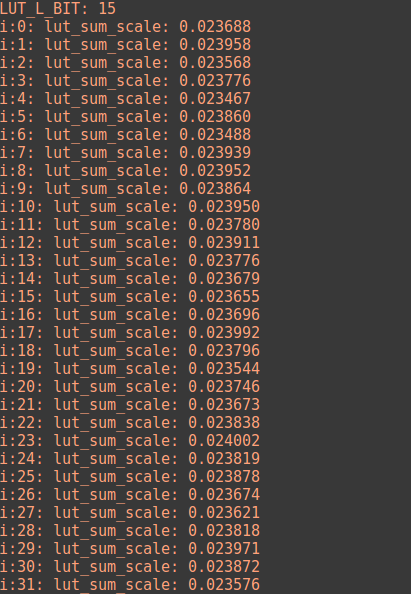
在介绍怎么使用表之前，还有一些参数需要搞清楚：

Lut\_sum\_scale：这个scale要和之前提到的反量化sum和的sum\_scale 基础上乘以lut\_l\_bit

的量化scale，相当于再融合一次量化操作：



打印看一下：



到目前为止 我们输入sum和的公式：

Sum\_scale(sum + Q\_bias) => 输出范围为我们设定的（-5，1）

如果此时不对结果范围做调整，我们乘以量化到Lut\_l\_bit的数会是带负数的。

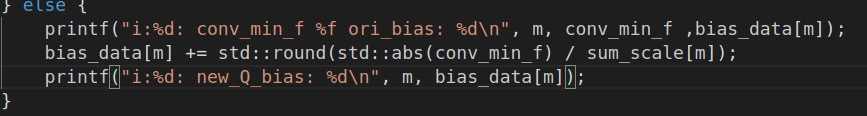
我们期望是无符号数所以需要对量化的bias做调整

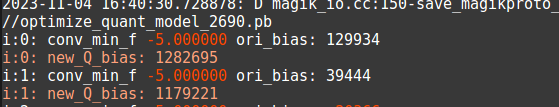
Sum\_scale(sum + Q\_bias) + abs（min） =》让输出范围为正

Sum\_scale(sum + Q\_bias + int(abs(min)/Sum\_scale))

所以量化的Q\_bias 需要再次加上 int(abs(min)/Sum\_scale)

对应代码：

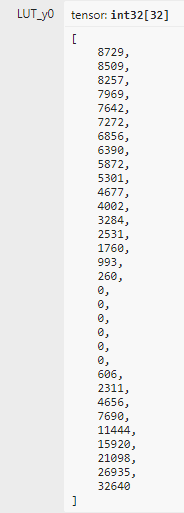
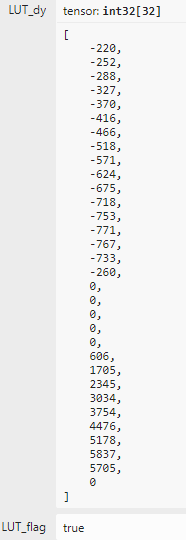


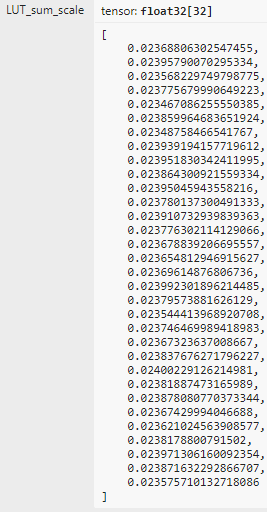


Sum\_scale(sum + Q\_bias)/(x\_max-x\_min)\*2\*\*(lut\_l\_bit)

Lut\_sum\_scale = Sum\_scale/(x\_max-x\_min)\*2\*\*(lut\_l\_bit)

Lut\_sum\_scale(sum + Q\_bias) =>范围为（0 ~ 2\*\*lut\_l\_bit）

将需要的参数保存到模型中：



luttbitLUT\_L\_BIT = 15

float_bitwidhtFloat\_bitwidth = 10

Float\_mask = 2\*\*10 -1 = 1023

mask_floatLUT\_T\_BIT = 15

lutlbitTable\_item\_bitwidth = 5

接下来是查找表流程：

1. 找到输入的值对应的index：

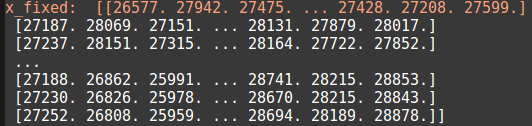
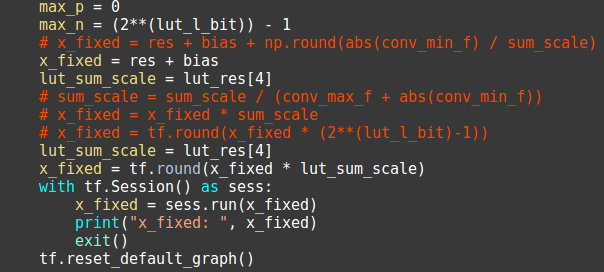
现在我们当前层的输入步骤一：

Lut\_sum\_scale × (SUM + Q\_Bias)

Lut\_sum\_scale 前面我们提到，他融合了对sum\_scale 和量化到lut\_l\_bit的scale，Q\_bias

也加了偏置保证输出结果均为正数，所以至此输出的范围为（0， 2\*\*lut\_l\_bit）

也就是【0， 2\*\*15】

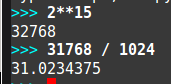


如何获取对应的32个表项的index

Lut\_l\_bit 为15，Table\_item\_bitwidth =5 即表示2\*\*5=32个表项， float\_bitwidth=10小数点的表示位宽

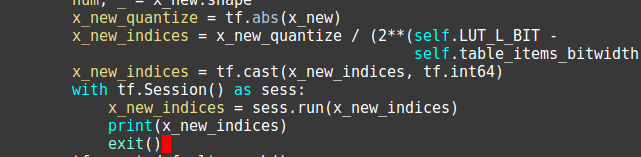
5

10

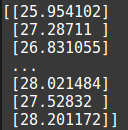
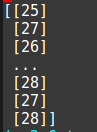
 Item\_bitwidth float\_bitwidth

所以直接整除float\_bitwidth 能被多少整除，的index就是多少：

推理代码：



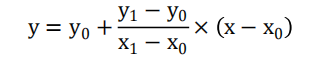
未转换成整型的值 转换后的

到此我们已经输入的x，找到了对应表中的index

1. 使用查找表：

查找表公式：



=> **y1-y0)**

到此 根据已知的index ， 我们可以从y0表中得到对应的y0, 从dy表中得到对应的(y1-y0)

现在只差得到 (x-x0/x1-x0)这个值。

x0 x x1

25 25.95 26

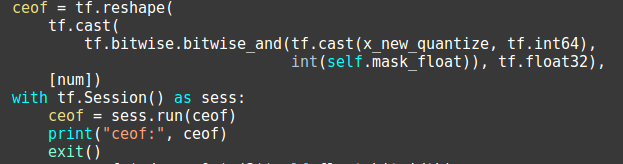
(x-x0/x1-x0) 因为这是一个比值，只要我们单位统一就可以等价计算，

那以第一个数x值（整除完2\*\*float\_bitwidth）为25.954102

(x-x0)/(x1-x0) = （25.954102 - 25）/ (26-25) = 0.954102

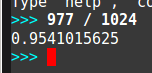
因为我们单位x1-x0始终等于1, 即(x-x0)/(x1-x0) 始终为小数部分。所以我们直接获取对应值的小数部分即可，我们知道float\_bitwidth=10 ，即Float\_mask = 2\*\*10 -1 = 1023，直接做按位与运算即可得到小数位的值

代码：

print4

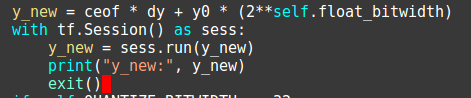
Ceof 就是我们要求得(x-x0)/(x1-x0)

还是第一个数我们得到了ceof=977，这个比值小数是（float\_bitwidth）表示的，我们把他还原到真实小数：



是和真实小数一致的，到此所有未知变量已知道，接下来套入插值公式即可：

因为ceof是10bit表示的，所以需要将y0的表示也统一到同一表征范围，对y0要乘以2\*\*float\_bitwidth。



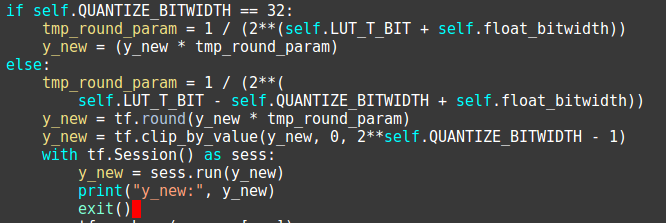
y_new1

到此我们回顾一下 表值的生成过程

激活（x） =》 浮点的激活输出 =》 量化到output\_bitwidth位宽，保留小数位 =》

左移(LUT\_T\_bit - output\_bitwidth)位，进行定点化 =》现在查找表公式统一表征范围左移float\_bitwidth

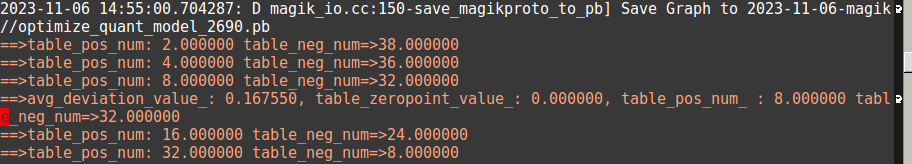
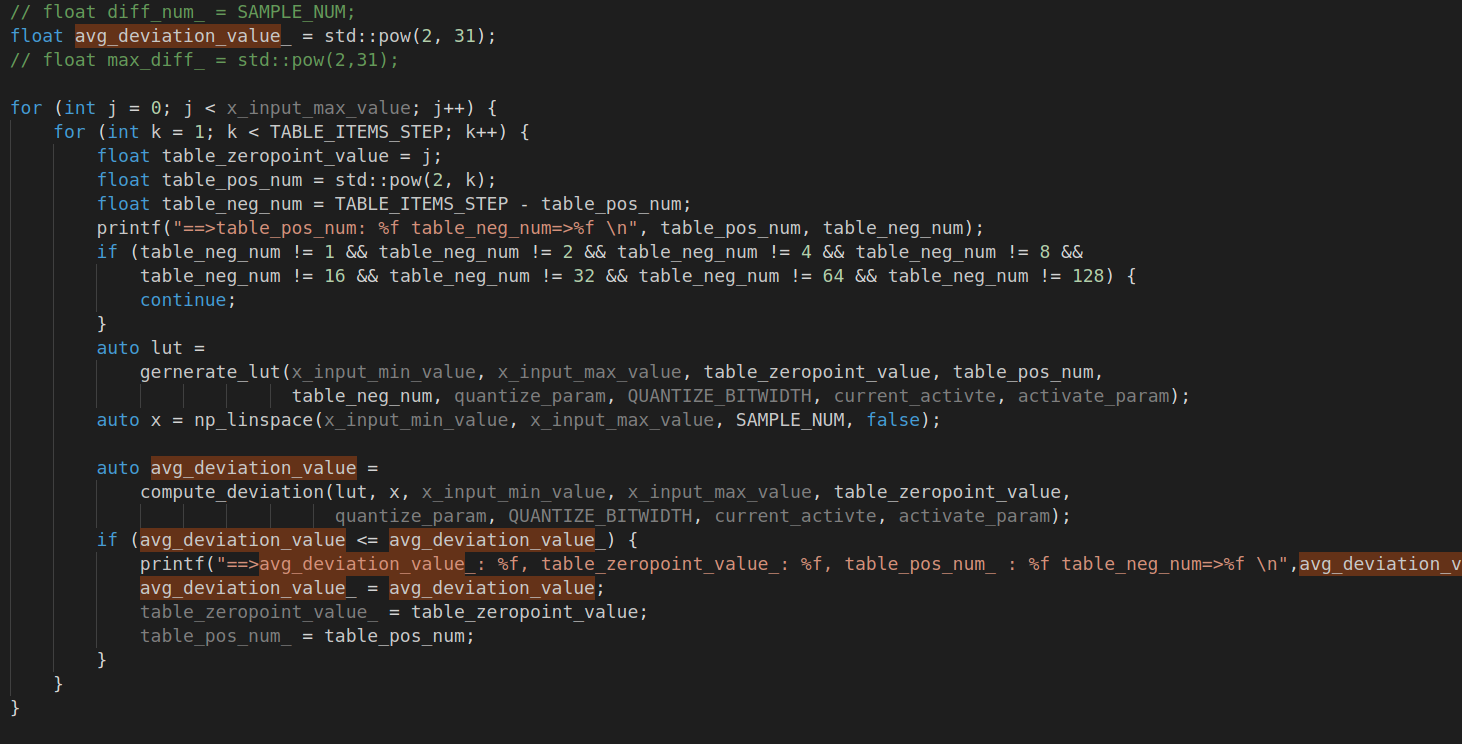
所以，如果我们要得到量化的output\_bitwidth 值，需要将最终结果右移(LUT\_T\_bit - output\_bitwidth) + float\_bitwidth

代码：y_newa

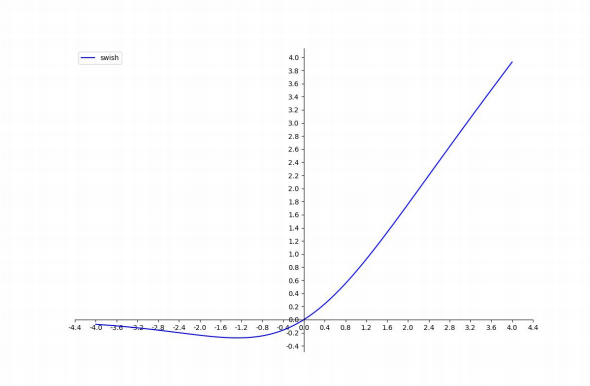
**带正负半轴查找表：**

较不分正负半轴查找表，代码流程中进行了对最优tabel\_zeropoint\_value的搜索：

TABEL\_ITEMS\_STEP = 40



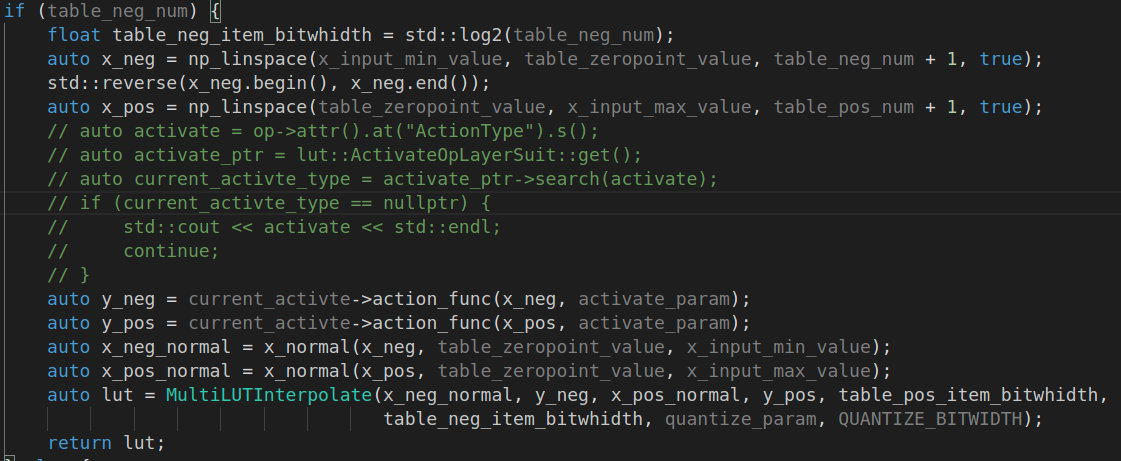
如何理解呢，还是以swish激活函数为参考，正半轴更像是线性函数，负半轴更为复杂，所以负半轴应该分的更细一些，正半轴可以适当分的粗一些。

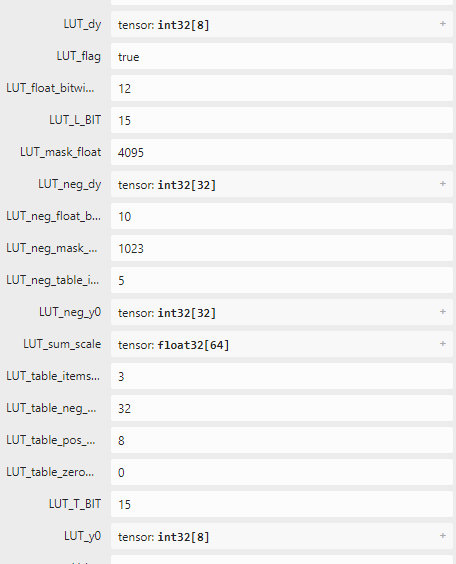


然后在区间【x\_min\_value, tabel\_zeropoint\_value】 分为32个表项

区间【table\_zeropoint\_value】分为8个表项，分别建立pos表和neg表

代码：





Pos表的dy

Neg表的dy

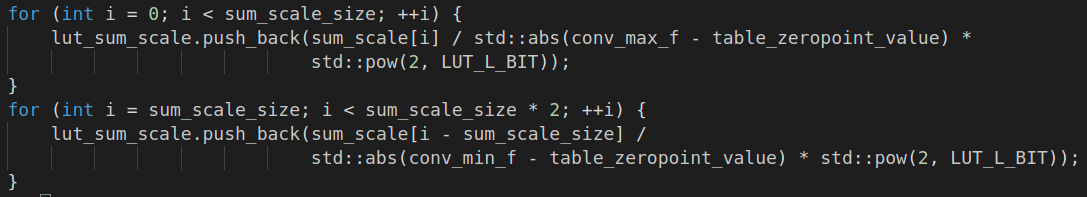
Neg表的d0

前32个为Pos，后32个为Neg

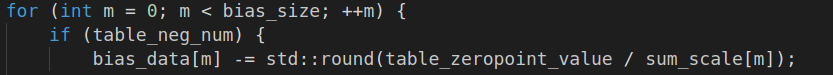
Pos表的d0

Pos表的y0

对应的Lut\_sum\_scale 计算：



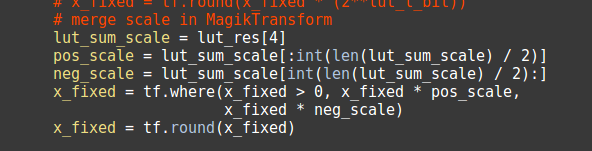
带有负半轴的Q\_bias 会将sum+bias 的范围变成table\_zeropoint\_value值分割正负半轴【min， 0】【0， max】



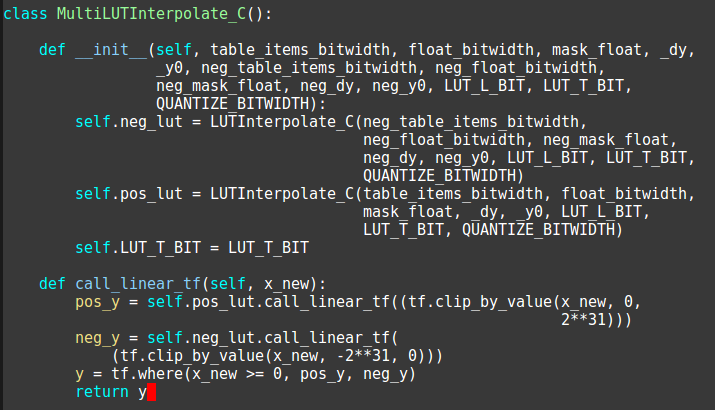
推理：

分别查找对应的表和lut\_sum\_scale

X的计算



查找对应的表



查表流程完全一致。