1、什么是校准?

校准是为模型权重和激活选择边界的过程。为了简单起见，这里只描述了对称范围的校准（工业界一般对称量化即可满足需求），如对称量化所需。这里考虑三种校准方法：

- Max：使用校准期间的最大绝对值；
- Histogram：将范围设置为校准期间看到的绝对值分布的百分位。
- Entropy：使用 KL 散度来最小化原始浮点值和量化比特表示的值之间的信息损失。

2、Max校准

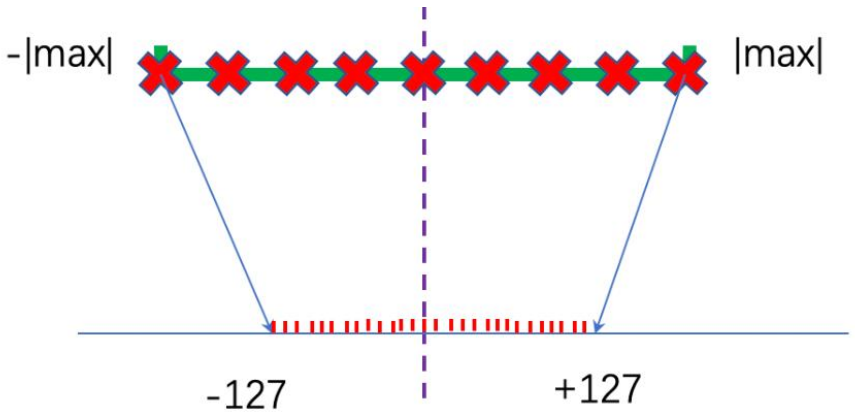
Max校准主要使用校准期间的最大绝对值；

校准过程如下所示：

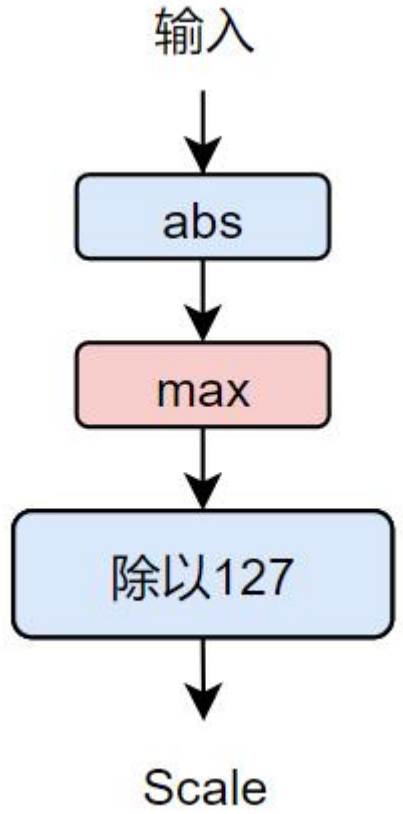
1、对输入求取绝对值；

2、算出绝对值的最大值；

3、以TensorRT的int8对称量化为例，这里如果想得到Scale就要除以127。



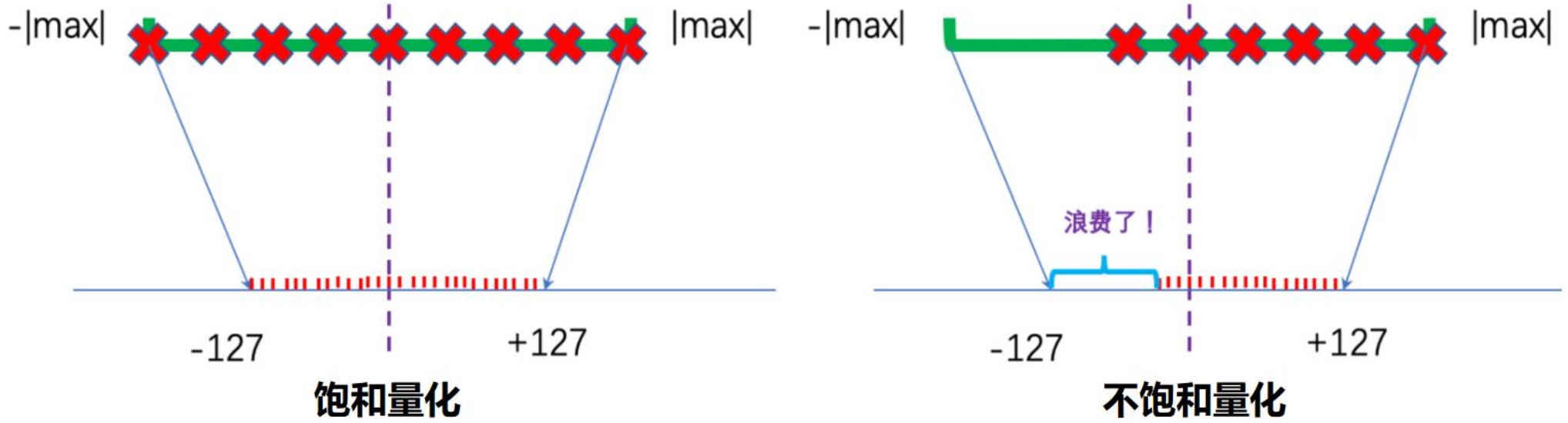
数据分布均匀




```
def maxq(value):
    dynamic_range = np.abs(value).max()
    scale = dynamic_range / 127.0
    return scale
```

3、Histogram校准

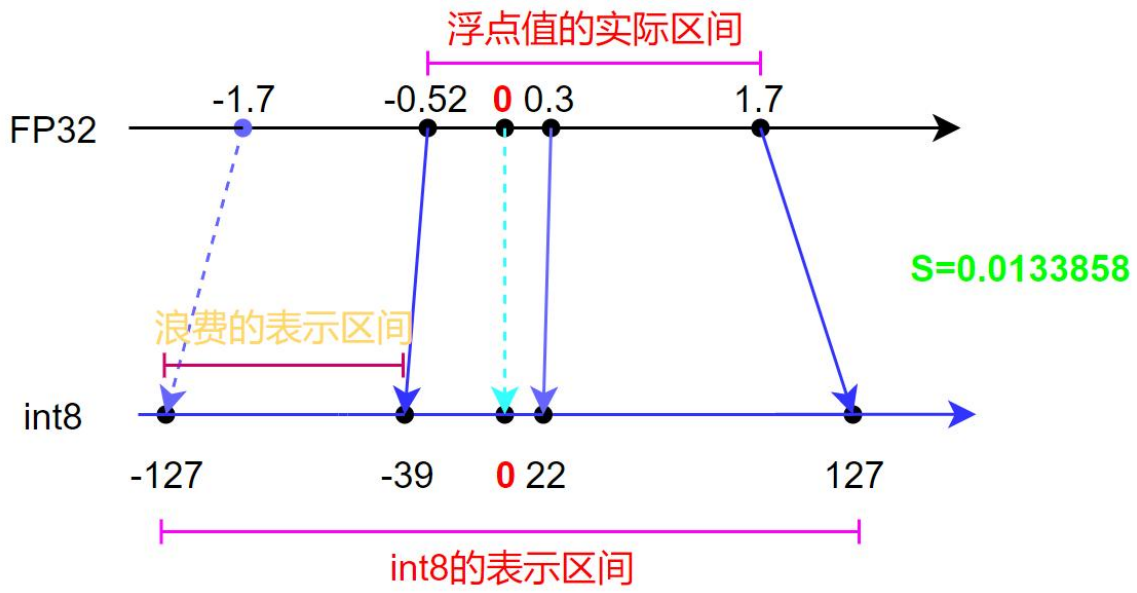
我们都知道量化的过程与数据的分布有关。如下图所示：当数据的直方图分布比较均匀时，高精度向低精度进行映射就会将表示空间利用比较充分；如果分布不均匀，就会浪费很大的表示空间。



关于上面这种直接将量化阈值设置为 |max| 的方法，它的显著特点是int8的表示空间没有充分利用，因此称为不饱和量化(no saturation quantization)。

针对这种情况，可以选择一个合适的量化阈值(threshold)，舍弃那些超出范围的数进行量化，这种量化方式充分利用了int8的表示空间，因此称为饱和量化(saturation quantization)。

3、Histogram校准



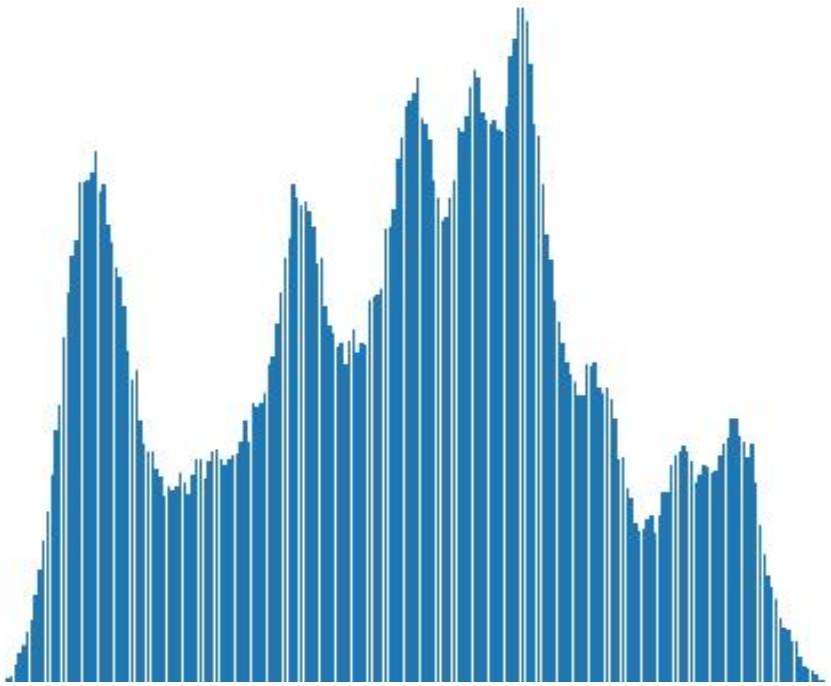
对比两种量化方式可以发现，它们各有优缺点：不饱和量化方式的量化范围大，但是可能会浪费一些低比特的表示空间从而导致量化精度低；

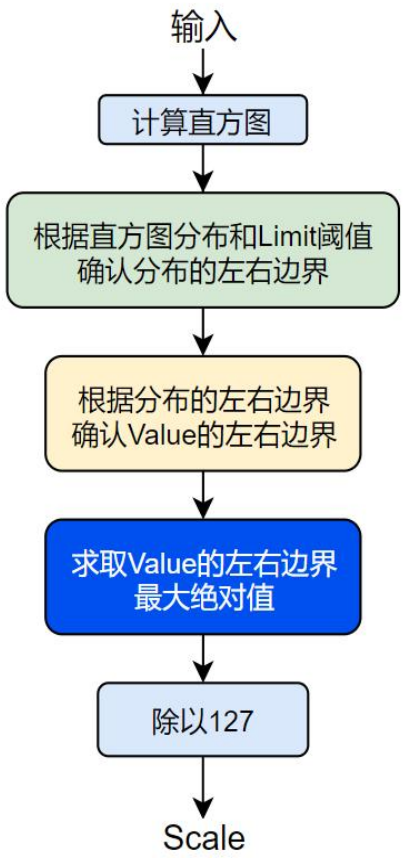
饱和量化方式虽然充分利用了低比特表示空间，但是会舍弃一些量化范围。因此这两种方式其实是一个量化精度和量化范围之间的平衡。

那么问题来了，对于数据流的饱和量化，怎么在数据流中找到这个最佳阈值(threshold)？

3、Histogram校准

最佳阈值(threshold)的选择主要有2种方式，分别是Histogram与Entropy的方式，关于Histogram方法的流程如下：






```python
def histogramq(value):
    # 计算直方图
    hist, bins = np.histogram(value, 100)
    total = len(value)
    left, right = 0, len(hist)
    limit = 0.99
    while True:
        nleft = left + 1
        nright = right - 1
        left_cover = hist[nleft:right].sum() / total
        right_cover = hist[left:nright].sum() / total
        # 判断是否left和right都小于limit的限度，True的化退出
        if left_cover < limit and right_cover < limit:
            break
        if left_cover > right_cover:
            left += 1
        else:
            right -= 1
    # 根据直方图占比和limit计算的left和right边界,确定value种的数值边界
    low, high = bins[left], bins[right - 1]
    # 计算最大绝对值边界
    dynamic_range = max(abs(low), abs(high))
    # 计算scale
    scale = dynamic_range / 127.0
    return scale
```

4、Entropy校准

TensorRT采用的校准就是KL散度校准。选择使FP32和int8的激活值分布的KL距离最小的阈值，校准过程如下所示：

统计输入 *value* 的 *max*(|*x*|)值；

统计输入 *value* 的 *Histogram*；

遍历直方图分布，*stride* 为 *bin* 的长度，默认为 128：

生成 p 分布；

计算 q 分布；

归一化 p 分布和 q 分布；

计算 p 分布和 q 分布的 KL 散度；

求取 KL 散度的最小值对应的位置；

求取 *threshold*；

计算 *scale* 的结果；

这里先假设 P 的分布为：[5,3,1,7,5,9,0,2]，

这里目标 bin_num 数为 4，然后计算非 0 的数组如下：

is_zeros = [1,1,1,1,1,1,1,0]

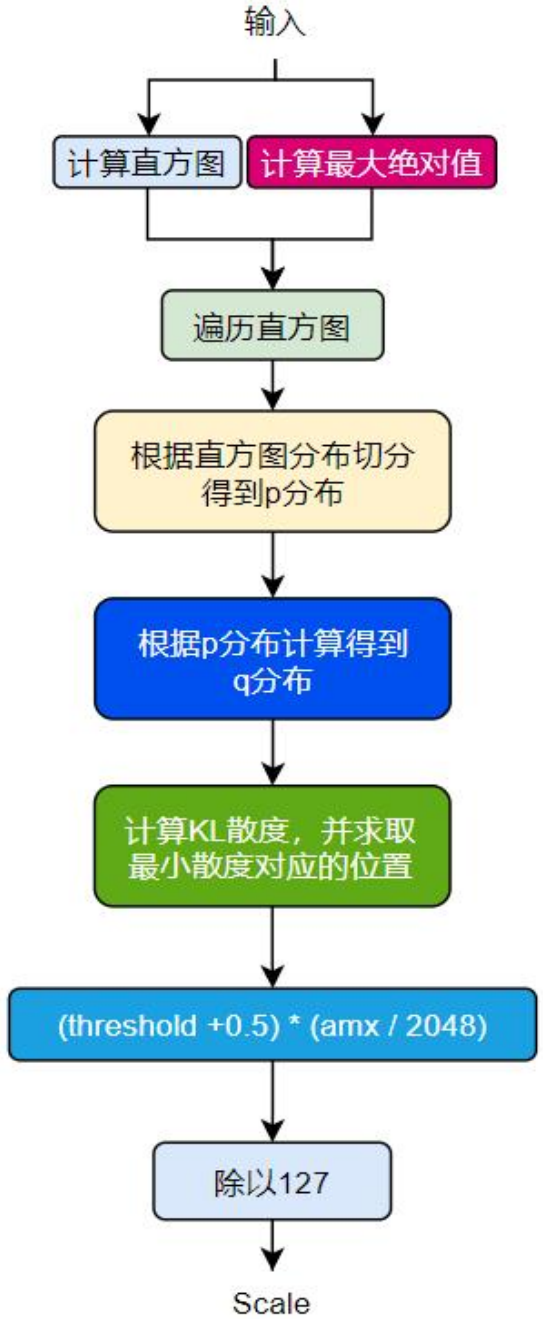
这里通过 stride=len(P)//bin_num=2 来进行 P 的 bin 分布合并，

得到 quant_p 为：

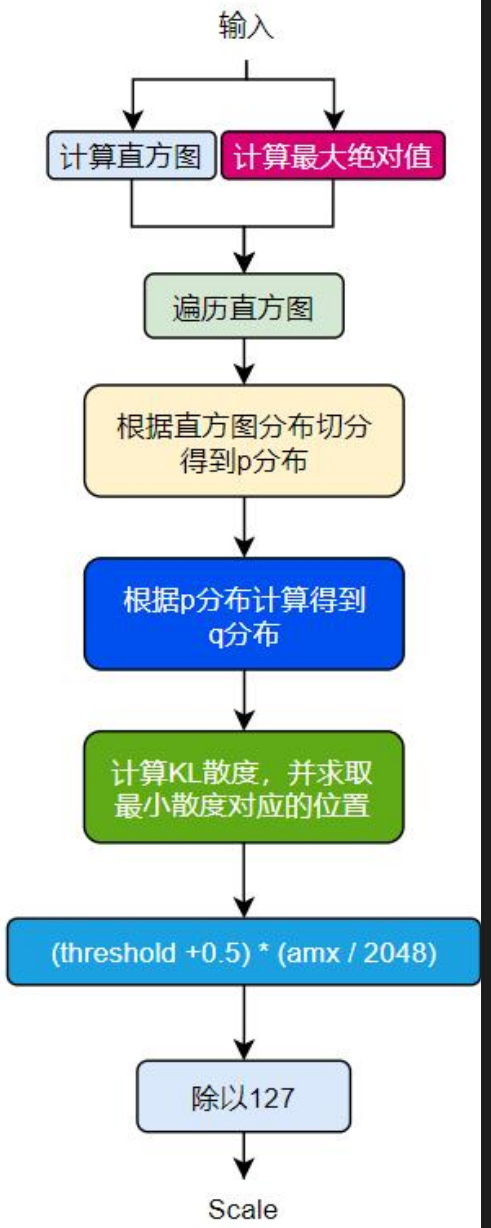
quant_p=[8,8,14,2]

然后结合 stride 和 is_zeros 将 quant_p 反合并为 Q：

Q=quant_p / [2,2,2,1]=[4.,4.,4.,4.,7.,7.,0.,0.]



4、Entropy校准




```python
def entropy(value, target_bin=128):
    # 计算最大绝对值
    amax = np.abs(value).max()
    # 计算直方图分布
    distribution, _ = np.histogram(value, bins=2048, range=(0, amax))
    # 遍历直方图分布，区间为[1:2048]
    distribution = distribution[1:]
    length = distribution.size
    # 定义KL散度
    kl_divergence = np.zeros(length - target_bin)
    # 遍历[128:2047]
    for threshold in range(target_bin, length):
        # slice分布，区间为[:threshold]
        sliced_nd_hist = copy.deepcopy(distribution[:threshold])
        # 复制切分分布为：p
        p = sliced_nd_hist.copy()
        threshold_sum = sum(distribution[threshold:])

        # 边界外的组加到边界P[i-1]上，没有直接丢掉
        p[threshold-1] += threshold_sum
        is_nonzeros = (p != 0).astype(np.int64)
        # 合并bins，步长为：num_merged_bins=sliced_nd_hist.size // target_bin=16
        quantized_bins = np.zeros(target_bin, dtype=np.int64)
        num_merged_bins = sliced_nd_hist.size // target_bin
        for j in range(target_bin):
            start = j * num_merged_bins
            stop = start + num_merged_bins
            quantized_bins[j] = sliced_nd_hist[start:stop].sum()
        quantized_bins[-1] += sliced_nd_hist[target_bin * num_merged_bins:].sum()
        # 定义分布：q ，这里的size要和p分布一致，也就是和sliced_nd_hist分布一致
        q = np.zeros(sliced_nd_hist.size, dtype=np.float64)
        # 根据步长结合p的非零以及quant_p，来以步长填充q
        for j in range(target_bin):
            start = j * num_merged_bins
            stop = -1 if j == target_bin - 1 else start + num_merged_bins
            norm = is_nonzeros[start:stop].sum()
            q[start:stop] = float(quantized_bins[j]) / float(norm) if norm != 0 else q[start:stop]
        # 归一化操作
        p = p / sum(p)
        q = q / sum(q)
        # 计算KL散度
        kl_divergence[threshold - target_bin] = KL(p, q)
```

5、校准方法对比

```python
if __name__ == '__main__':
    # x -> Q1 -> conv1 -> Q2 -> conv2 -> y
    np.random.seed(31)
    nelem = 1000
    # 生成随机权重、输入与偏置向量
    x = np.random.randn(nelem)
    weight1 = np.random.randn(nelem)
    bias1 = np.random.randn(nelem)

    # 计算第一层卷积计算的结果输出（fp32）
    t = x * weight1 + bias1
    weight2 = np.random.randn(nelem)
    bias2 = np.random.randn(nelem)

    # 计算第二层卷积计算的结果输出（fp32）
    y = t * weight2 + bias2
    # 分别对输入、权重以及中间层输出（也是下一层的输入）进行量化校准
    xQ  = Quant(x)
    w1Q = Quant(weight1)
    tQ  = Quant(t)
    w2Q = Quant(weight2)
    qt  = Quant_Conv(x, weight1, bias1, xQ, w1Q, tQ)
    # int8计算的结果输出
    y2  = Quant_Conv(qt, weight2, bias2, tQ, w2Q)
    # 计算量化计算的均方差
    y_diff = (np.abs(y - y2) ** 2).mean()
```

```python
class Quant:
    def __init__(self, value):
        # 这里是对称量化,动态范围选取有多种方法，max/histogram/entropy等等
        self.scale = maxq(value)
        # self.scale = histogramq(value)
        # self.scale = entropy(value)

    def __call__(self, f):
        # 进行阶段
        return saturate(f / self.scale)
```

```python
def Quant_Conv(x, w, b, iq, wq, oq=None):
    alpha = iq.scale * wq.scale
    out_int32 = iq(x) * wq(w)

    if oq is None:
        # float32 output
        return out_int32 * alpha + b
    else:
        # int8 quant output
        return saturate((out_int32 * alpha + b) / oq.scale)
```

```
'''
max mse error        : 35.1663
histogramq mse error : 8.6907
entropy mse error    : 1.8590
'''
```

```python
# int8截断，注意，-128丢掉了不要
def saturate(x):
    return np.clip(np.round(x), -127, +127)
```

对比Max、Histogram以及KL这3种校准方法，量化误差依次是：Max > Histogram > KL。