深蓝学院
shenlanxueyuan.com

# TensorRT INT8加速

# 课程目标

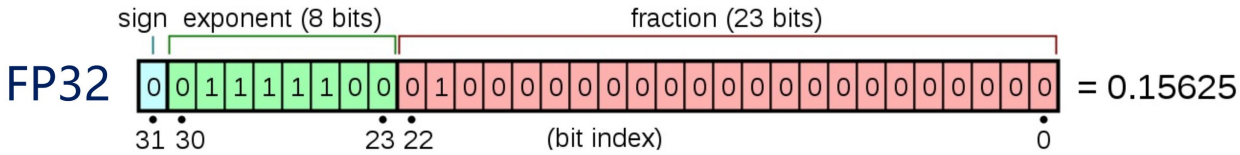
理论部分：

- TensorRT FP16优化加速
- INT8 量化原理
- TensorRT INT8优化加速
- TensorRT INT8 Demo 代码讲解

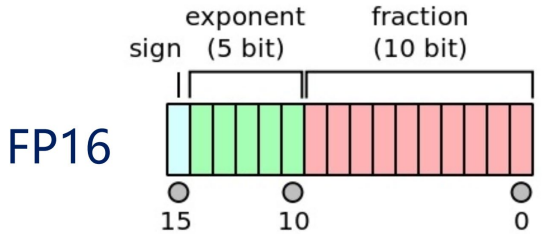
# 课程大纲

| | |
|---|---|
| 基础 | 1. TensorRT FP16优化<br>2. 什么是INT8量化?<br>3. 为什么INT8量化会快?<br>4. 为什么INT8量化不会大幅损失精度?<br>5. INT8量化算法介绍<br>6. 怎么用TensorRT INT8 量化? |
| 进阶 | 7. INT8 一定快么?<br>8. 如何使用TensorRT进行大规模上线 |

# TensorRT FP16优化-FP16是什么




$$\text{Value} = sign \times 2^{(exponent-127)} \times (1 + fraction)$$




$$\text{Value} = sign \times 2^{(exponent-15)} \times (1 + fraction)$$

# TensorRT FP16优化

config->setFlag(BuilderFlag::kFP16);

builder->platformHasFastFp16()　　builder->platformHasFastInt8()

## 4. Hardware And Precision

The following table lists NVIDIA hardware and which precision modes each hardware supports. TensorRT supports all NVIDIA hardware with capability SM 5.0 or higher. It also lists the availability of Deep Learning Accelerator (DLA) on this hardware. Refer to the following tables for the specifics.

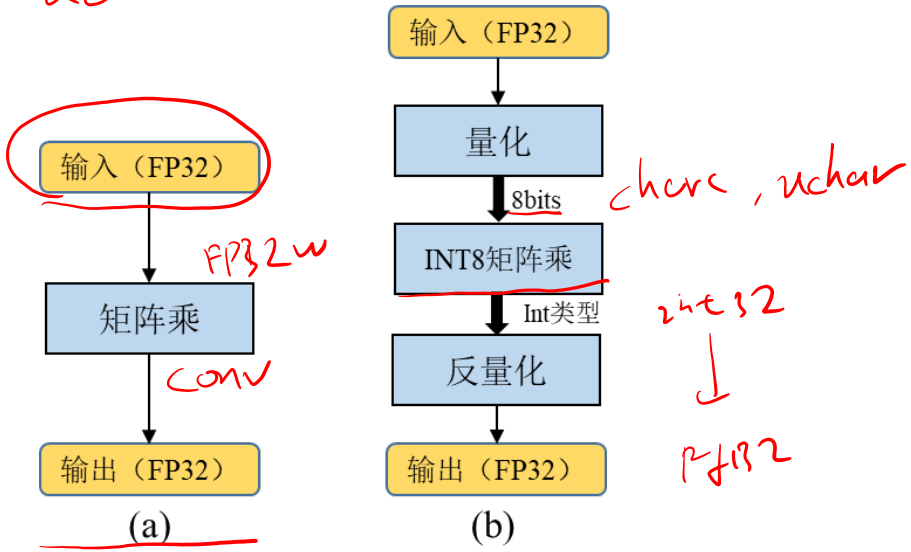
**Note:** Support for CUDA Compute Capability version 3.0 has been removed. Support for CUDA Compute Capability versions below 5.0 may be removed in a future release and is now deprecated.

**Table 4. Supported hardware**

| CUDA Compute Capability | Example Device | TF32 | FP32 | FP16 | INT8 | FP16 Tensor Cores | INT8 Tensor Cores | DLA |
|---|---|---|---|---|---|---|---|---|
| 8.6 | NVIDIA A10 | Yes | Yes | Yes | Yes | Yes | Yes | No |
| 8.0 | NVIDIA A100/GA100 GPU | Yes | Yes | Yes | Yes | Yes | Yes | No |
| 7.5 | Tesla T4 | No | Yes | Yes | Yes | Yes | Yes | No |
| 7.2 | Jetson AGX Xavier | No | Yes | Yes | Yes | Yes | Yes | Yes |
| 7.0 | Tesla V100 | No | Yes | Yes | Yes | Yes | No | No |
| 6.2 | Jetson TX2 | No | Yes | Yes | No | No | No | No |
| 6.1 | Tesla P4 | No | Yes | No | Yes | No | No | No |
| 6.0 | Tesla P100 | No | Yes | Yes | No | No | No | No |
| 5.3 | Jetson TX1 | No | Yes | Yes | No | No | No | No |
| 5.2 | Tesla M4 | No | Yes | No | No | No | No | No |
| 5.0 | Quadro K2200 | No | Yes | No | No | No | No | No |

# 什么是INT8量化?

将基于浮点的模型转换成低精度的int8数值进行运算，以加快推理速度




（a）为正常神经网络的流程　（b）为量化神经网络的流程

# 为什么INT8量化会快?

（1）对于计算能力大于等于SM_61的显卡，如Tesla P4/P40 GPU，NVIDIA提供了新的INT8点乘运算的指令支持-DP4A。该计算过程可以获得理论上最大4倍的性能提升。

（2）Volta架构中引入了Tensor Core也能加速INT8运算。

资料： https://zhuanlan.zhihu.com/p/75753718

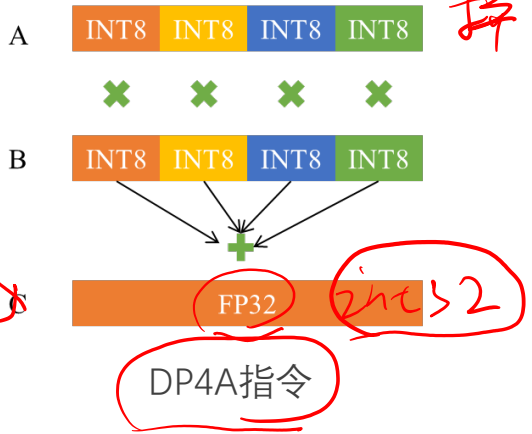
**延伸**

✓ X86 AVX AVX2 AVX512指令

资料：https://zhuanlan.zhihu.com/p/136099964

✓ ARM NEON指令

资料：https://aijishu.com/a/1060000000193827




**FP16 和 INT8能加速的本质**：

通过指令 或者 硬件技术，在单位时钟周期内，

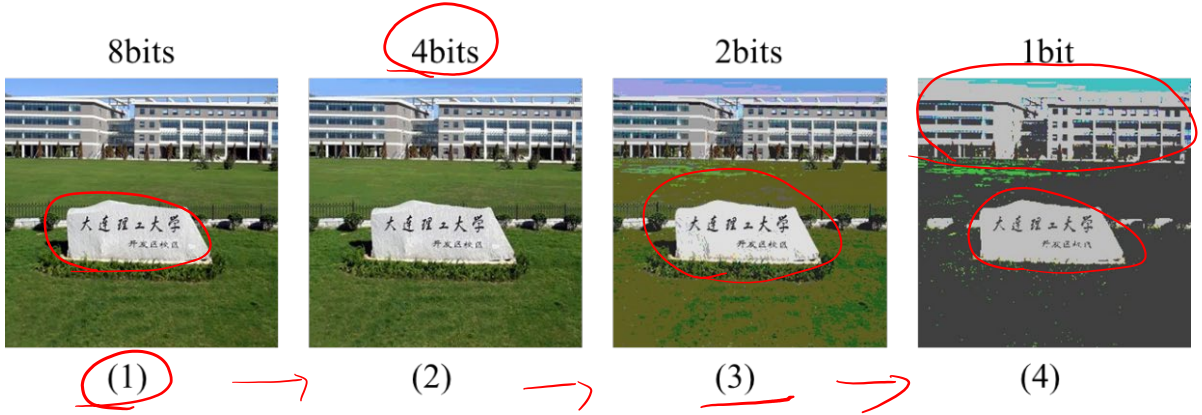
FP16 和 INT8 类型的运算次数 大于 FP32 类型的运算次数。

# 为什么INT8量化不会大幅损失精度？

神经网络的特性：具有一定的鲁棒性。

原因：训练数据一般都是有噪声的，神经网络的训练过程就是从噪声中识别出有效的信息。

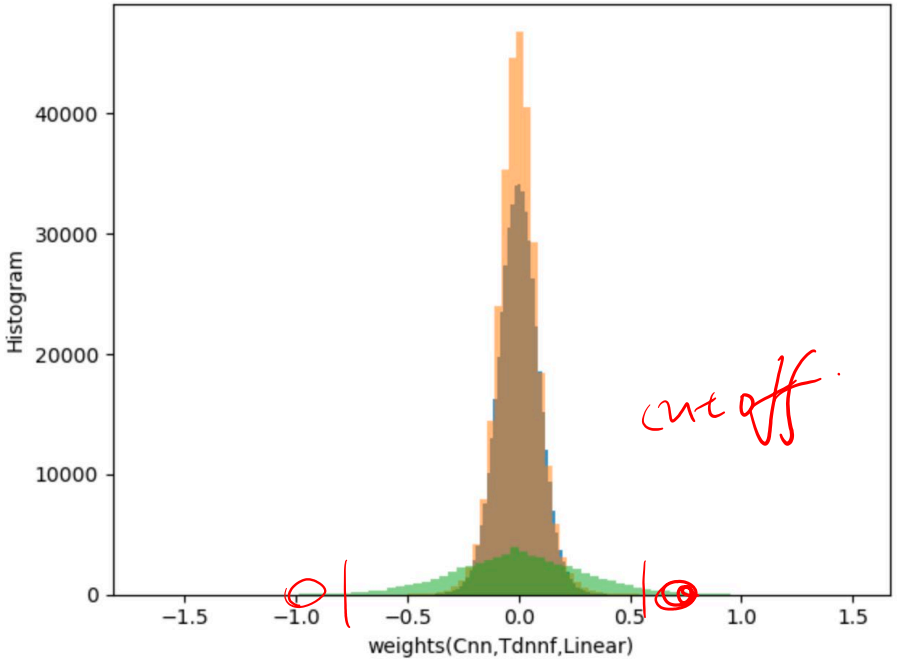
思路：可以将低精度计算造成的损失理解为另一种噪声。




(1)为原图，使用8bits存储；(2) (3) (4)为将原图量化为4bits、2bits、1bit后进行反量化的结果

# 为什么INT8量化不会大幅损失精度?

神经网络权值，大部分是正态分布，值域小且对称



# INT8量化算法

动态对称量化算法 使用于:

（1）PyTorch dynamic quantization

（2）ONNX quantization

server，接大

动态非对称量化算法 使用于

（1）Google Gemmlowp

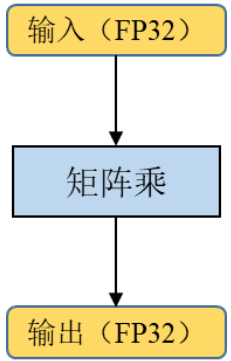
ARM 接小

静态对称量化算法 使用于

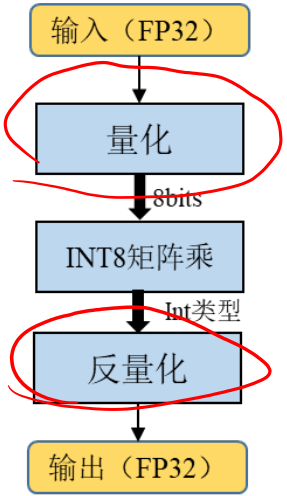
（1）PyTorch static quantization

（2）TensorRT

（3）NCNN

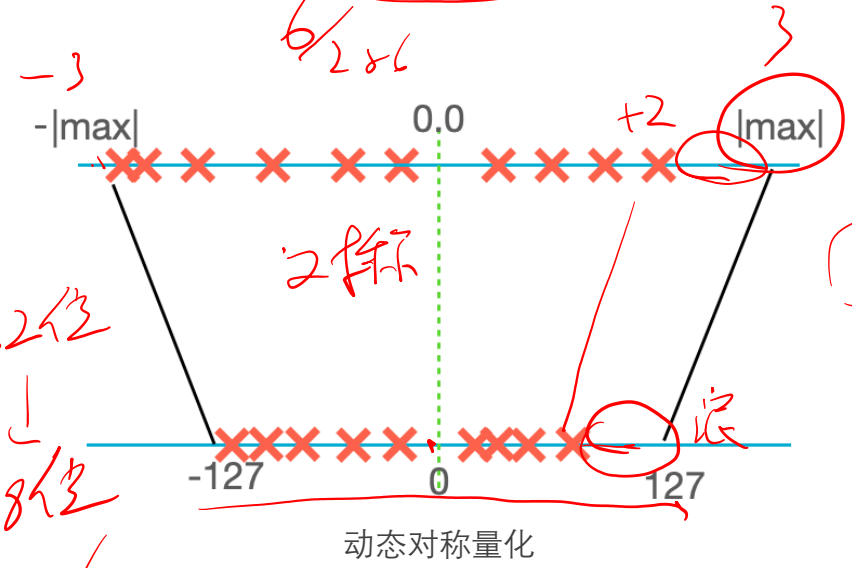


(a)



(b)

# INT8量化算法-动态对称量化算法



动态对称量化

$$scale = |max| * \frac{2}{256}$$

$$real\_value = scale * quantized\_value$$

其中，

$real_value$为真实值，即float类型；

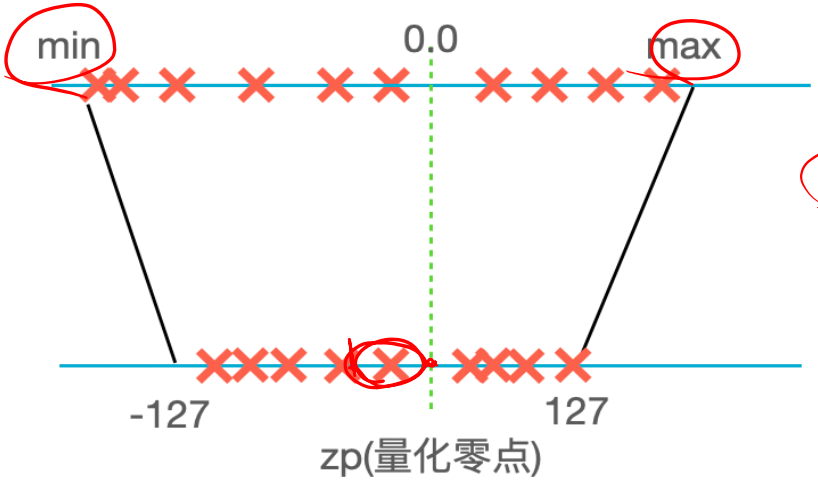
$quantized_value$为INT8量化的结果，即char类型；

$|max|$为输入的模最大值，由$|max|$可求出$scale$ 。

优点：算法简单，量化步骤耗时时间短；

缺点：会造成位宽浪费，影响精度。

# INT8量化算法-动态非对称量化算法




动态不对称量化

$$scale = \frac{|max - min|}{256}$$

$$real\_value = scale * (quantized\_value - zero\_point)$$

其中，

$real_value$为真实值，即float类型；

$quantized_value$为INT8量化的结果，即char类型；

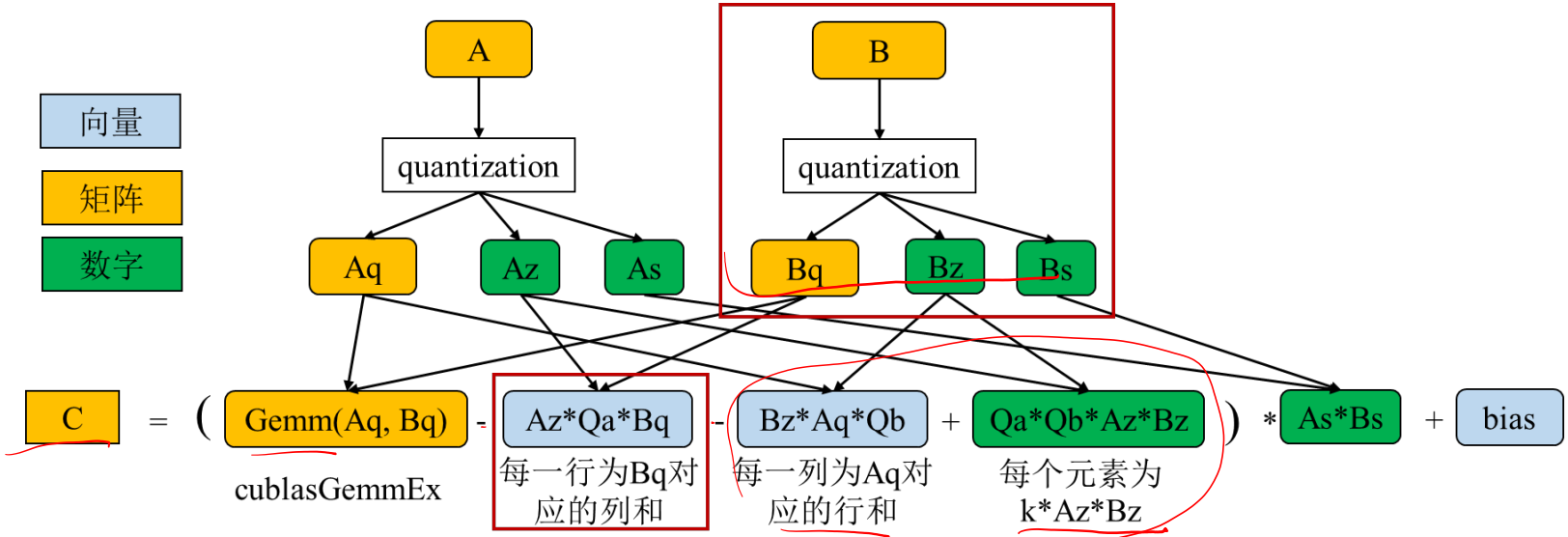
$max$为输入的模最大值，$min$为最小值；

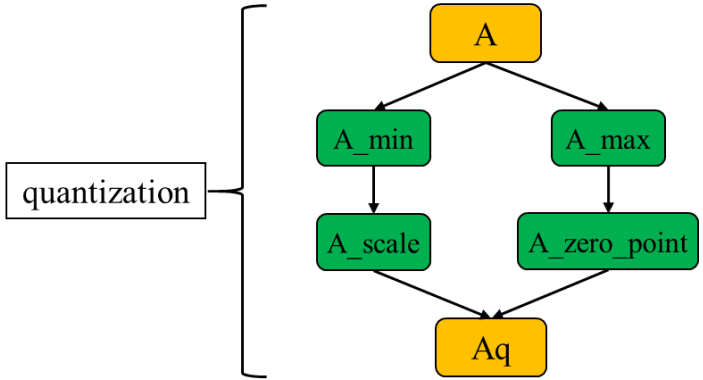
$zero_point$为零点值；

由$max$和$min$可求出倍数$scale$ 。

优点：不会造成bit位宽浪费，精度有保证；

缺点：算法较复杂，量化步骤耗时时间长。

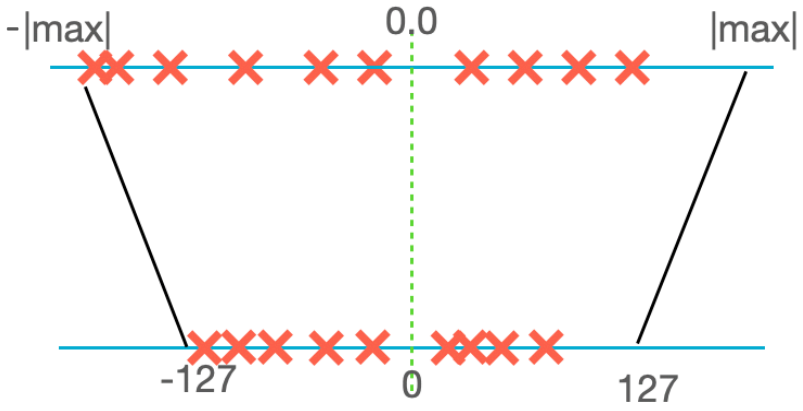


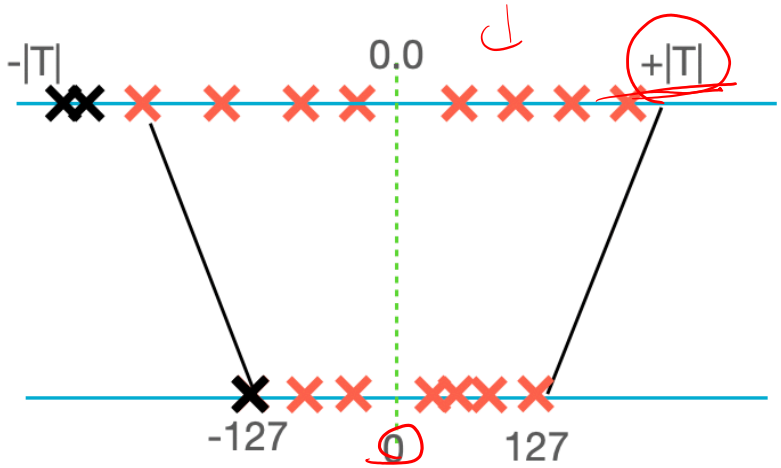

2行

1、weights先计算出来，包括Bq、Bz、Bs和Bq的列和，然后存储到文件中。
2、Gemm(Aq, Bq)，Bz*Aq*Qb可以并行算
3、Az、As、Bz、Bs放到Constant memory中

# INT8量化算法-静态对称量化算法




动态对称量化




静态对称量化

- 动态量化，推理时实时统计数值|max|
- 静态量化，推理时使用预先统计的缩放阈值，截断部分阈值外的数据

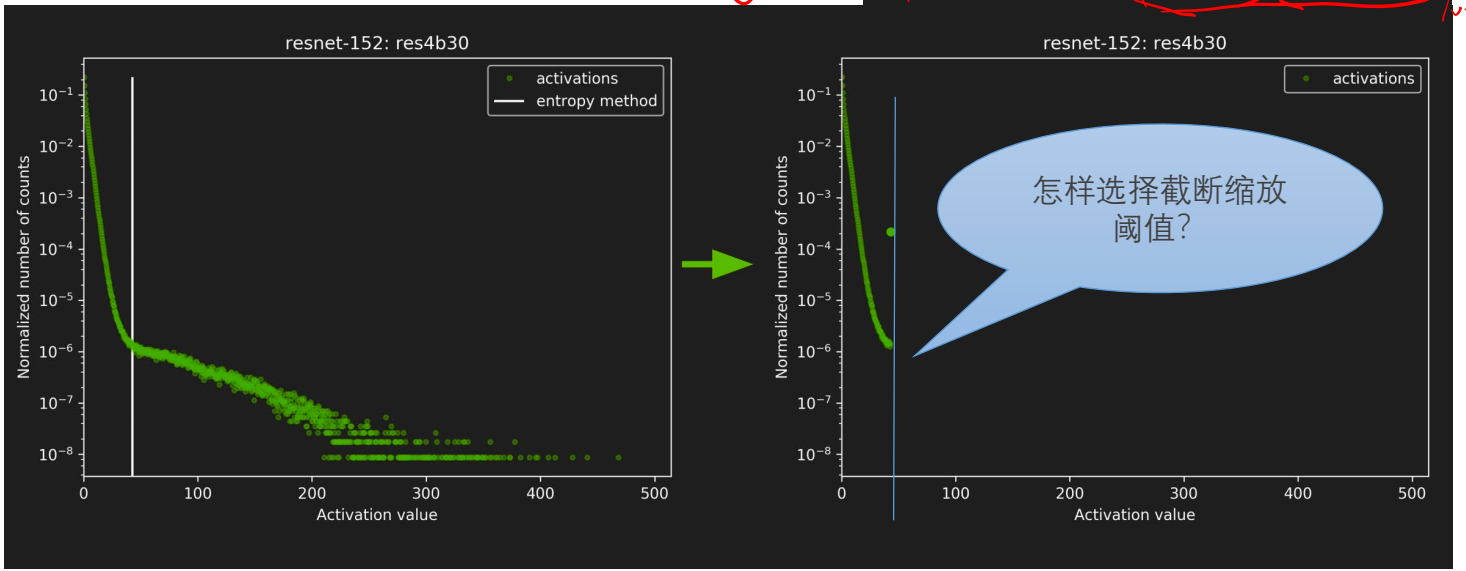
# INT8量化算法-静态对称量化算法

优点：算法最简单，量化耗时时间最短，精度也有所保证；

缺点：构建量化网络比较麻烦。

Nvidia: 8-bit Inference with TensorRT

- Run FP32 inference on Calibration Dataset.
- For each Layer:
  - collect histograms of activations.
  - generate many quantized distributions with different saturation thresholds.
  - pick threshold which minimizes KL_divergence(ref_distr, quant_distr).



# 怎么用TensorRT INT8 量化?

TensorRT-7.2.3.4\samples\sampleINT8\sampleINT8.cpp

```
    if (dataType == DataType::kINT8)
        config->setFlag(BuilderFlag::kINT8);
    builder->setMaxBatchSize(mParams.batchSize);

    if (dataType == DataType::kINT8) {
        MNISTBatchStream calibrationStream(mParams.calBatchSize,
                                           mParams.nbCalBatches,
                                           "train-images-idx3-ubyte",
                                           "train-labels-idx1-ubyte",
                                           mParams.dataDirs);
        calibrator.reset(new Int8EntropyCalibrator2<MNISTBatchStream>(
                         calibrationStream, 0,
                         mParams.networkName.c_str(),
                         mParams.inputTensorNames[0].c_str()));
        config->setInt8Calibrator(calibrator.get());
        // void IBuilderConfig::setInt8Calibrator(IInt8Calibrator* calibrator)
    }
```

# 怎么用TensorRT INT8 量化?

TensorRT-7.2.3.4\include\NvInfer.h

```
class IInt8Calibrator {
public:
    virtual int32_t getBatchSize() const = 0;

    //! \brief Get a batch of input for calibration.
    virtual bool getBatch(void* bindings[], const char* names[], int32_t nbBindings) = 0;

    //! \brief Load a calibration cache.
    virtual const void* readCalibrationCache(std::size_t& length) = 0;

    //! \brief Save a calibration cache.
    virtual void writeCalibrationCache(const void* ptr, std::size_t length) = 0;

    //! \brief Get the algorithm used by this calibrator.
    virtual CalibrationAlgoType getAlgorithm() TRTNOEXCEPT = 0;
};
```

# 怎么用TensorRT INT8 量化?

TensorRT-7.2.3.4\include\NvInfer.h

```
    //!
    //! \brief Get a batch of input for calibration.
    //!
    //! The batch size of the input must match the batch size returned by getBatchSize().
    //!
    //! \param bindings An array of pointers to device memory that must be updated to point
to device memory
    //! containing each network input data.
    //! \param names The names of the network input for each pointer in the binding array.
    //! \param nbBindings The number of pointers in the bindings array.
    //! \return False if there are no more batches for calibration.
    //!
    //! \see getBatchSize()
    //!
    virtual bool getBatch(void* bindings[], const char* names[], int32_t nbBindings)
TRTNOEXCEPT = 0;
```

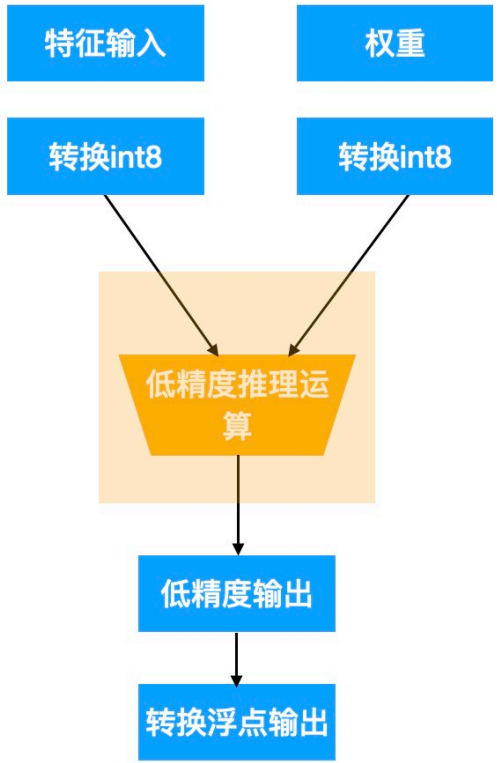
# 怎么用TensorRT INT8 量化?

- TensorRT-7.2.3.4\samples\common\BatchStream.h
- TensorRT-7.2.3.4\samples\common\EntropyCalibrator.h
- TensorRT-7.2.3.4\samples\sampleINT8\sampleINT8.cpp

# 进阶：INT8 一定快么？

- Float 运算时间 Tfloat
- 低精度运算时间Tint8
- 输入量化运算时间 Tquant
- 输出反量化时间 Tdequant

INT8 性能收益 = Tfloat – Tint8 – Tquant – Tdequant

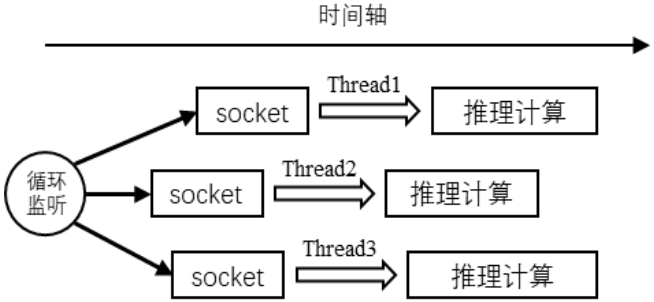



经验：

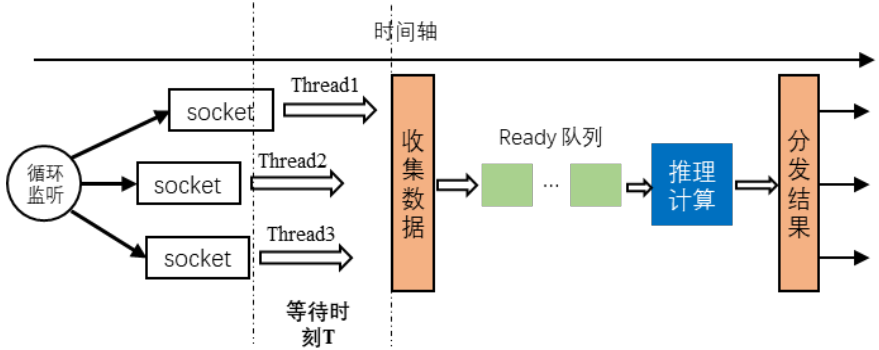
权值越大，输入越小，加速比越大。

输入越大，收益越小，甚至负收益。

# 进阶：如何使用TensorRT进行大规模上线

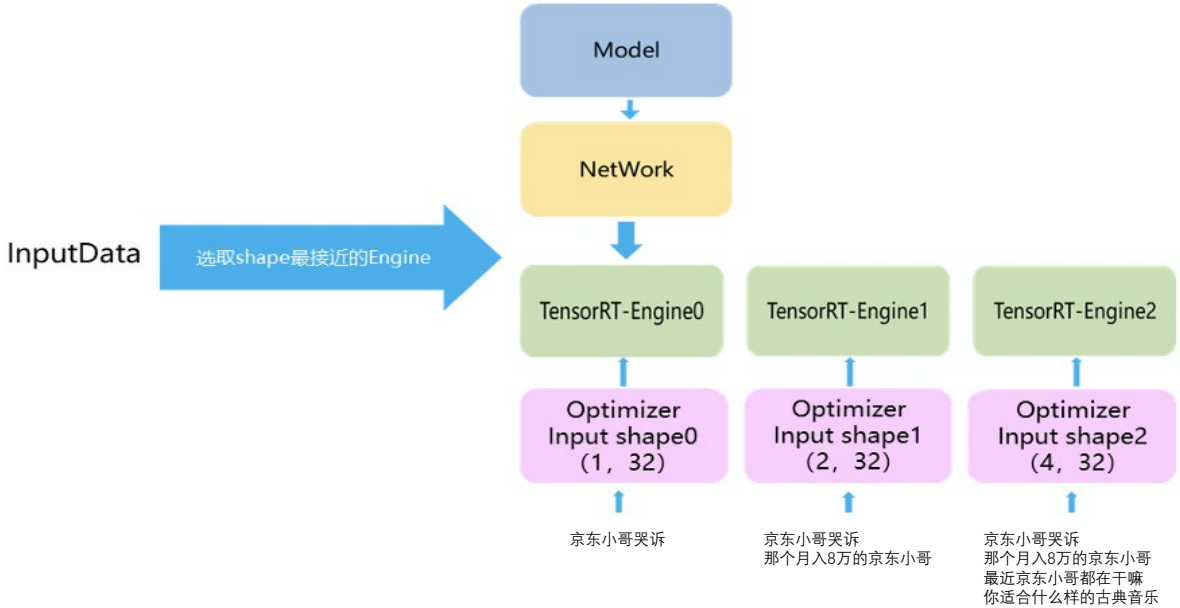



线性调度算法流程




批量调度算法流程

# 进阶：如何使用TensorRT进行大规模上线




多Engine定制

# 总结与建议

（1）对于深度神经网络的推理，TRT可以充分发挥GPU计算潜力，以及节省GPU存储单元空间。

（2）对于初学者，建议先从Sample入手，尝试替换掉已有模型，再深入利用网络定义API尝试搭建网络。

（3）如果需要使用自定义组件，建议至少先了解CUDA基本架构以及常用属性。

（4）推荐使用FP16/INT8计算模式

- FP16只需定义很少变量，明显能提高速度，精度影响不大；
- Int8有更多的潜力，但是可能会导致精度下降。

（5）如果不是非常了解TRT，也可以尝试使用集成了TRT的框架，但是如果不支持的网络层太多，会导致速度下降明显。

（6）在不同架构的GPU或者不同的软件版本的设备上，引擎不能通用，要重新生成一个。

深蓝学院
shenlanxueyuan.com

# 感谢聆听！

Thanks for Listening