

低精度表示用于深度学习 训练与推断

Brian Liu 刘建航, Intel AIPG

2018年3月

目录

1. 为什么需要低精度
2. 低精度带来的问题
3. FP16/FP32混合精度训练
4. 更多的创新 (FlexPoint16, DFP-16, bfloat16)
5. 小结

更多的数据



更大的模型



更强的计算能力

更强的计算能力

架构改进

- 增加核数
- 提高频率
- 单时钟周期指令数
- **低精度**
- 内存层次
- 节点互联

软件优化

- 提高并行性
- 改善负载均衡
- 数据布局格式
- 数据重用和预取
- 编译优化
- 框架优化

深度学习使用的数值精度

当前主流

训练/推断 32bit

(FP32)

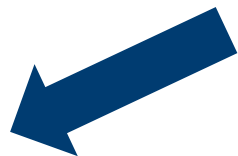


走向成熟

16bit - 训练 (FP16/DFP16)

8bit - 推断 (INT8)

我们这里主要讨论



训练

- 16bit混合精度逐渐成熟，商用硬件开始出现（如Intel的NNP，NVIDIA的Tensor Core）

推断

- 8bit推断已经成熟，商用硬件大规模使用（如Google的TPUv1）
- 4bit甚至1bit在探索中

低精度带来

更少内存

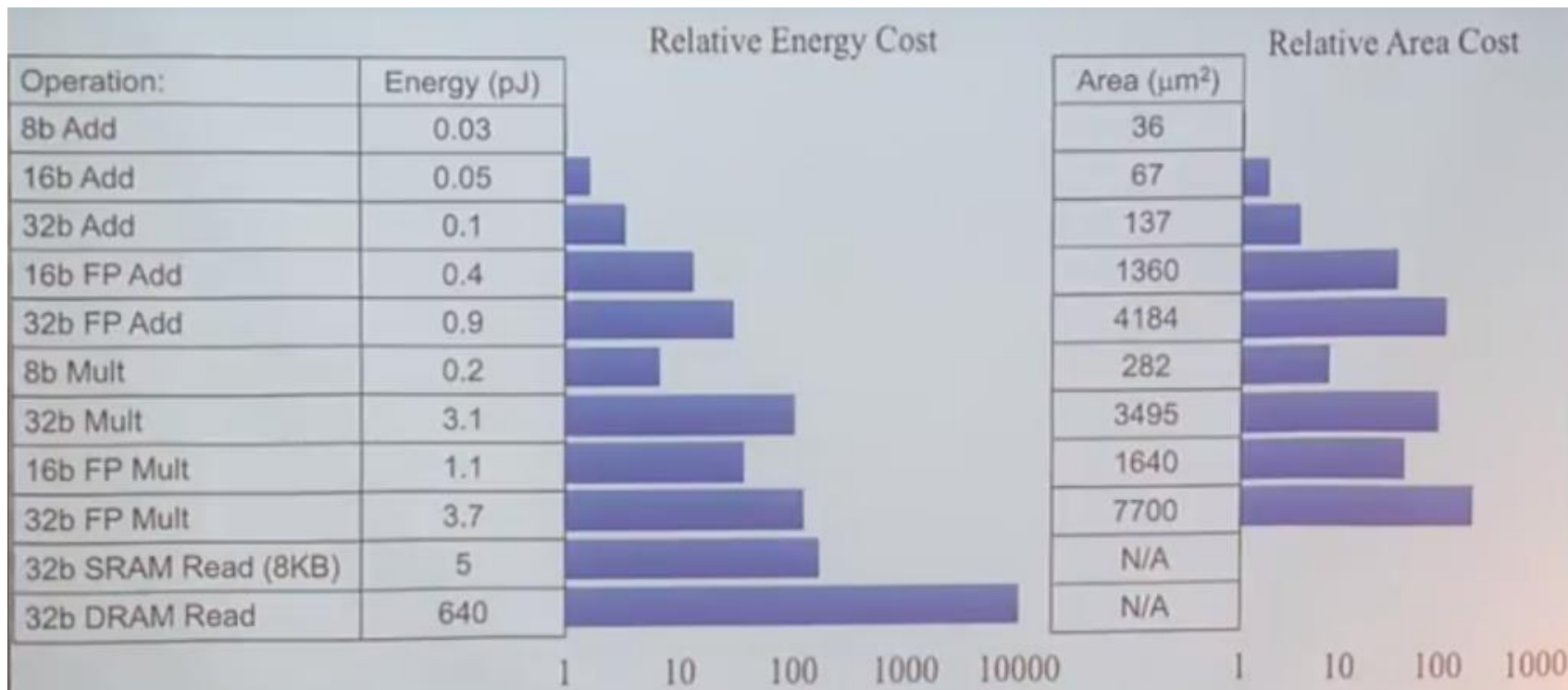
- 减少内存访问
- 更好匹配缓存
- 数据快速搬移

更小硅片面积

- 减少晶体管数量
- 减少能耗
- 更高的每秒操作数(OPS)



不同精度计算的消耗能量和硅片面积








Credit: Bill Dally ACMMM 2017 Keynote "Efficient Methods and Hardware for Deep Learning"

目录

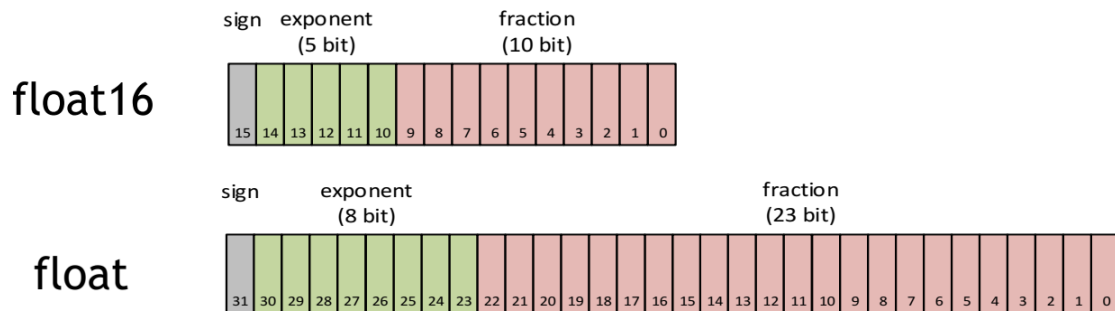
1. 为什么需要低精度
2. 低精度带来的问题
3. FP16/FP32混合精度训练
4. 更多的创新 (FlexPoint16, DFP-16, bfloat16)
5. 小结

不同精度表示的范围 (Range) 和精度 (Precision)

		Range	Accuracy
FP32		$10^{-38} - 10^{38}$.000006%
FP16		$6 \times 10^{-5} - 6 \times 10^4$.05%
Int32		$0 - 2 \times 10^9$	$\frac{1}{2}$
Int16		$0 - 6 \times 10^4$	$\frac{1}{2}$
Int8		$0 - 127$	$\frac{1}{2}$

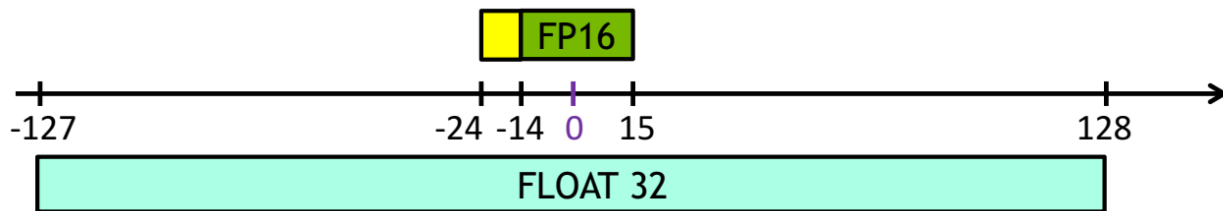
Credit: NIPS 2015 tutorial "High-performance hardware for machine learning"

IEEE754 – 单精度浮点数 (FP32) 和半精度浮点数 (FP16)



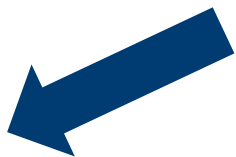
FP16的动态范围($6 \times 10^{-8} \sim 65504$) **远低于** FP32的动态范围($1.4 \times 10^{-45} \sim 1.7 \times 10^{+38}$)

FP16的精度(2^{-10}) **远粗于** FP32的精度(2^{-23})

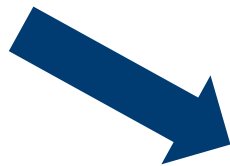


深度学习中使用FP16表示/计算带来的主要问题

量化误差 Quantization Error



狭窄的表示范围带来的溢出错误
(Overflow/Underflow)



精度不足带来的舍入错误
(Rounding Error)

溢出错误

OK FP32 weight = 2^{-25} (约等于 2.98×10^{-8})

Error FP16 weight = 2^{-25} (约等于 2.98×10^{-8})



下溢出 (Underflow) : FP16能表示的最小数为 2^{-24}

舍入错误

OK FP16 weight = 2^{-3} (0.125)

OK FP16 gradient = 2^{-14} (约等于0.000061)

FP16 weight = weight + gradient

$$= 2^{-3} + 2^{-14}$$

Error

$$= 2^{-3}$$

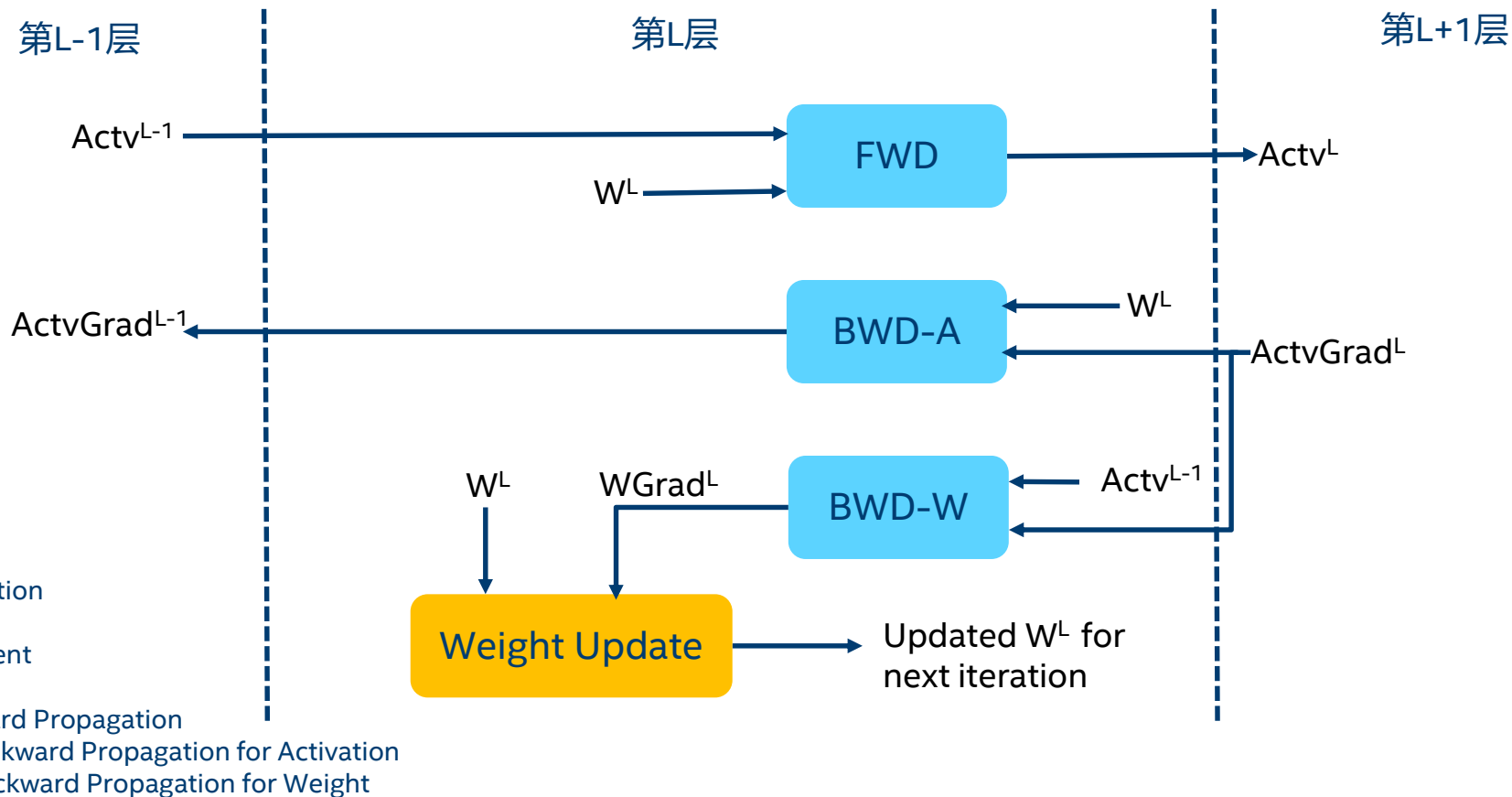


舍入错误 (Rounding Error) :

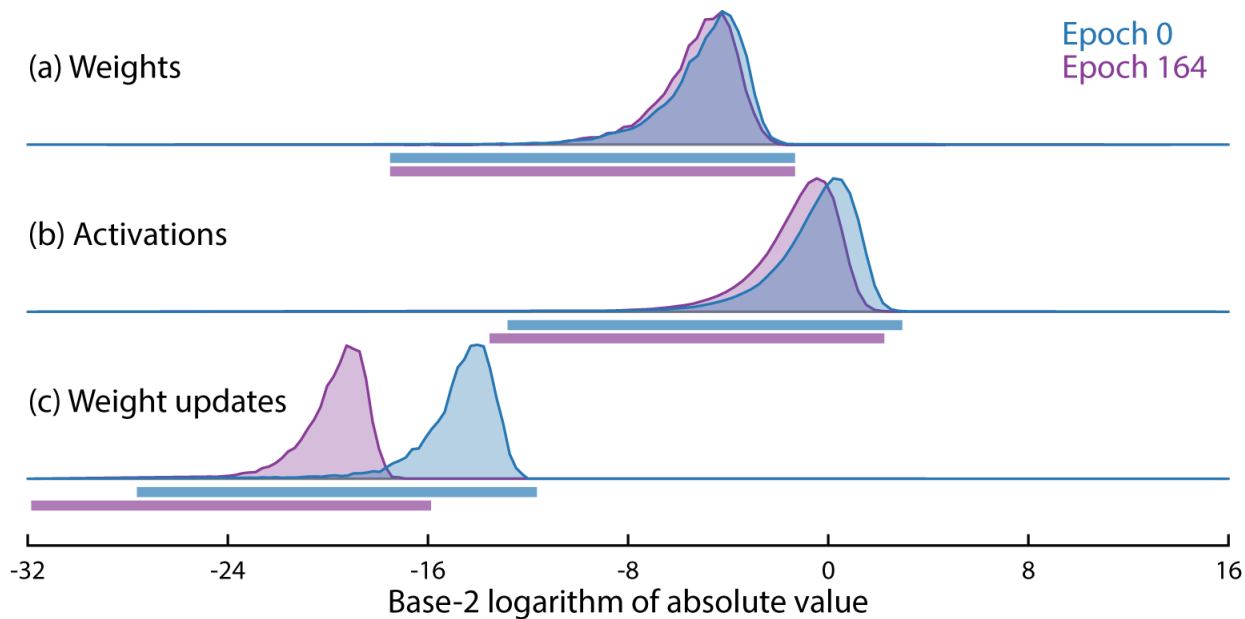
$[2^{-3}, 2^{-2}]$ 间, FP16表示的固定间隔为 2^{-13}

即比 2^{-3} 大的下一个数为 $2^{-3} + 2^{-13}$

深度学习的训练过程



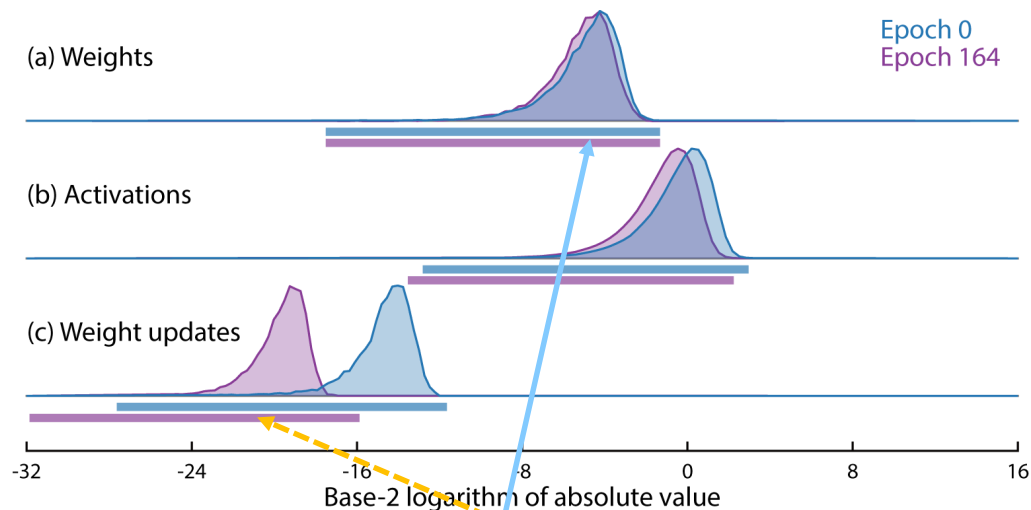
观察深度学习训练过程中的数值分布变化



图是使用FP32在CIFAR-10数据集上训练ResNet-101模型的过程，一共165个epochs。随着训练的进行，权重梯度（Weights gradient/updates）逐渐变小，其和权重本身的比例也在不断扩大

Credit: ai.intel.com "Flexpoint: numerical innovation" Xin Wang et al.

SGD更新权重(FP16)时，发生舍入错误



$$W(t+1) = W(t) - \lambda * \Delta W(t)$$

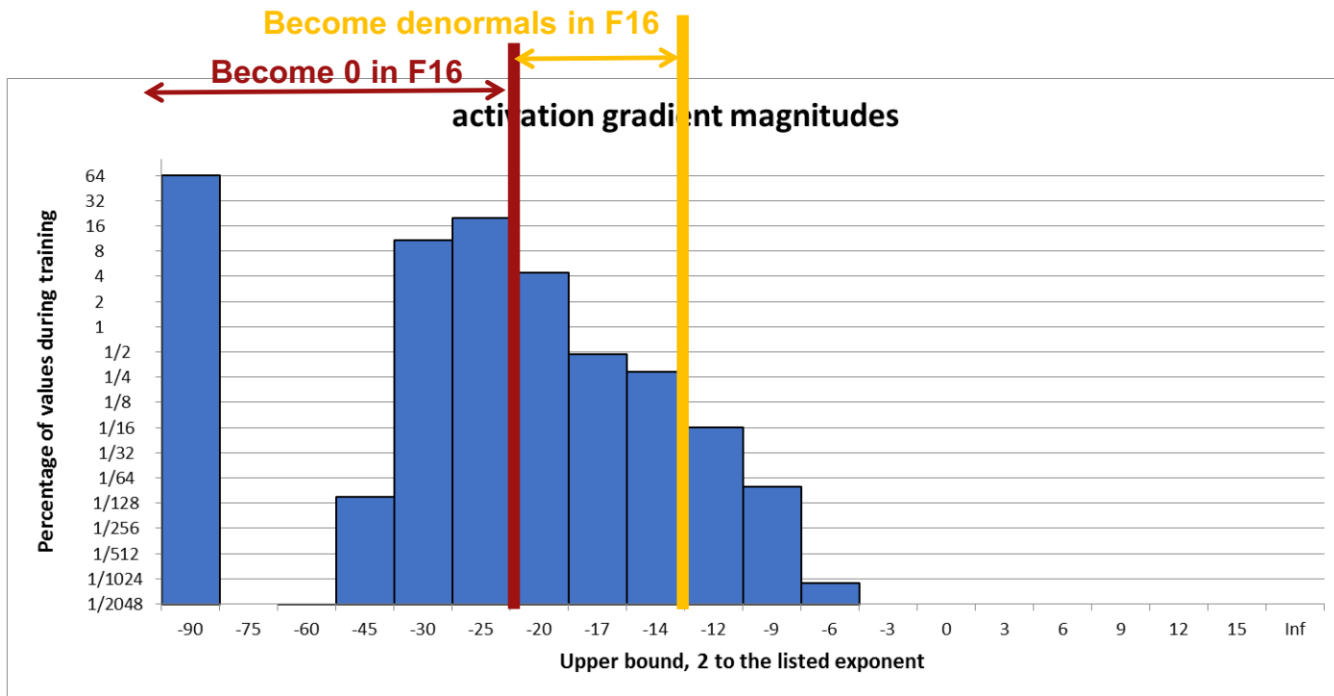
$$(\lambda * \Delta W(t)) / W(t) < 2^{-10}$$

舍入错误

权重未得到更新，训练失败

Credit: ai.intel.com "Flexpoint: numerical innovation" Xin Wang et al.

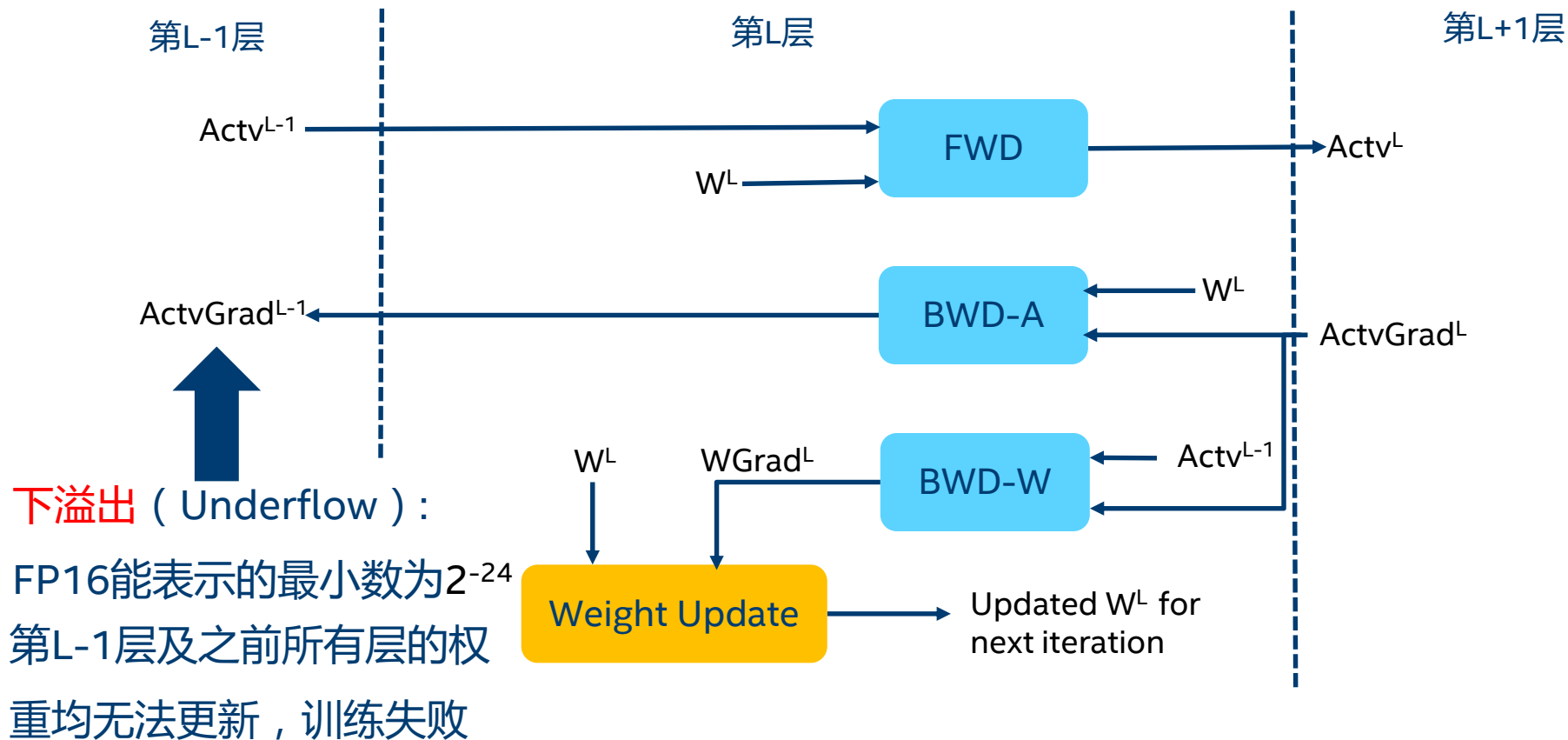
训练过程中激活梯度要远小于权重梯度



图为使用FP32进行Multibox SSD with VGG-D训练过程中，激活梯度（Activation Gradient）的幅度直方图

Credit: NVIDIA "Training with mixed precision" Ginsburg et al.

当激活梯度过小时



目录

1. 为什么需要低精度
2. 低精度带来的问题
3. FP16/FP32混合精度训练
4. 更多的创新 (FlexPoint16, DFP-16, bfloat16)
5. 小结

使用FP16/FP32混合精度进行训练

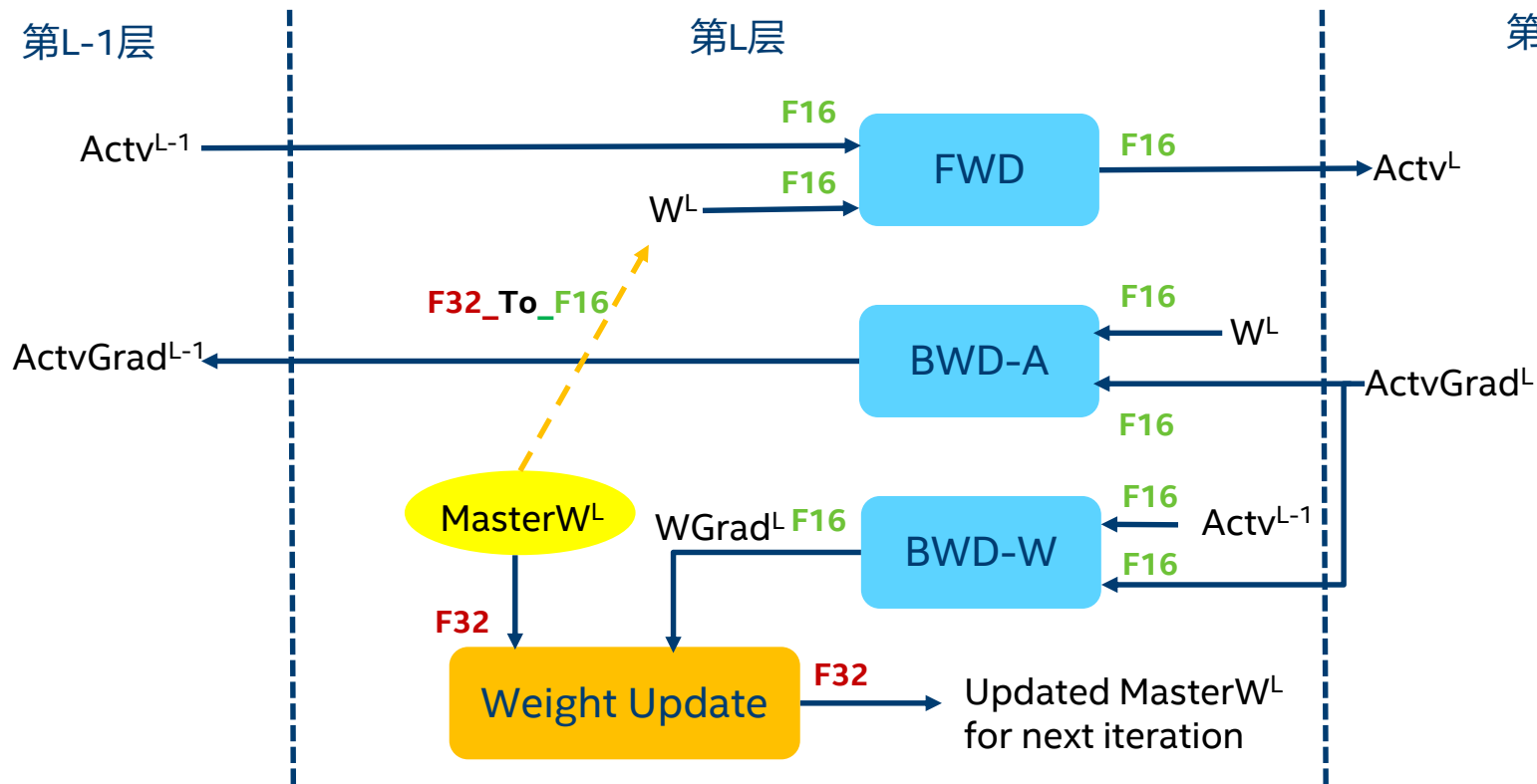
混合表示 (Representation)

- 所有权重、激活、梯度均使用FP16表示 (存储)
- 使用FP32作为累加器 (Accumulator) 用于累加FP16的乘积，只在写入内存前转换为FP16
- 为每一个权重，保留一个高精度 (FP32) 的主备份 (Master Copy)

混合计算 (Math)

- 区别对待不同层的计算需求
- 对计算敏感型的层，如全连接层、卷积层等，直接使用FP16进行计算 (当然还需要用FP32做累加器)
- 对BatchNorm、SoftMax等需要对整个Tensor进行统计操作 (如SUM) 的，可从内存读入FP16值后，进行FP32运算以保持精度

为权重增加用FP32表示的主备份



MasterW – 权重主备份，用FP32表示

$$W(t+1) = W(t) - \lambda * \Delta W(t) \text{ 避免了舍入错误}$$

为权重增加FP32主备份的代价

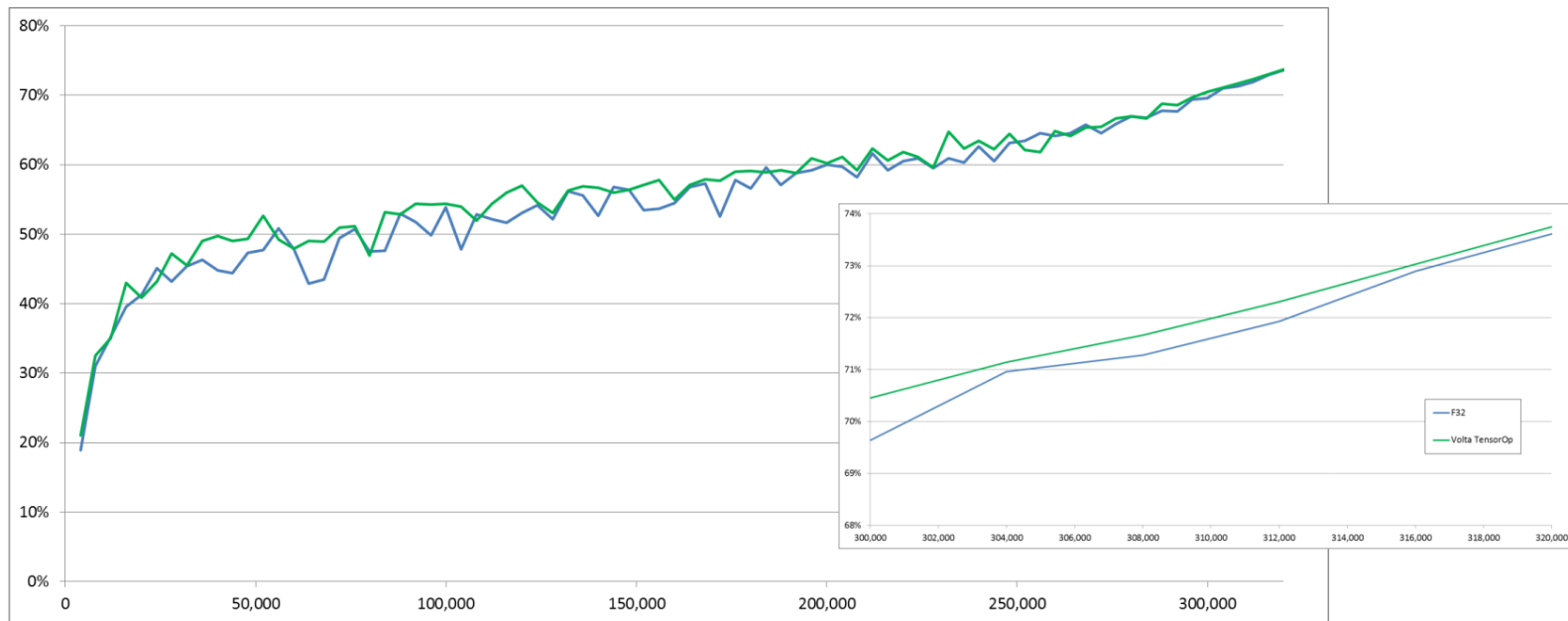
存储

- 相比单纯使用FP32，权重增加了50%
- 整体影响很小，内存占用主要由激活（及激活梯度）决定

计算

- 仅在权重更新时使用FP32运算
- 耗时的前向、后向计算均使用FP16运算

使用权重FP32主备份后 ResNet-50混合精度训练



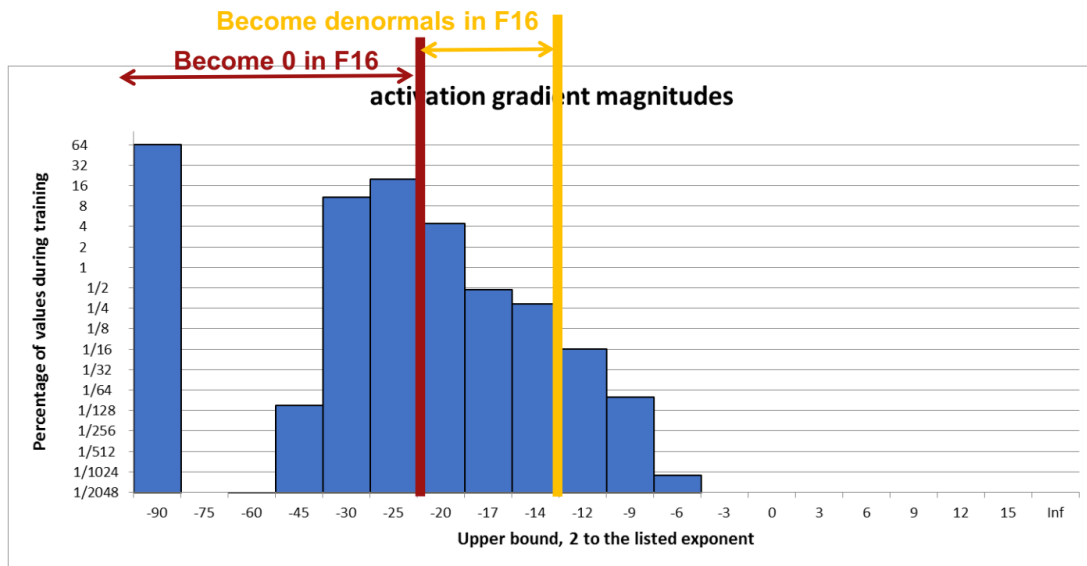
图为使用FP16的GPU TensorCore加上FP32权重主备份进行ResNet-50训练，使用与FP32训练相同的超参数。

达到和FP32训练相同的准确性

Credit: NVIDIA "Training with mixed precision" Ginsburg et al.

另一个问题

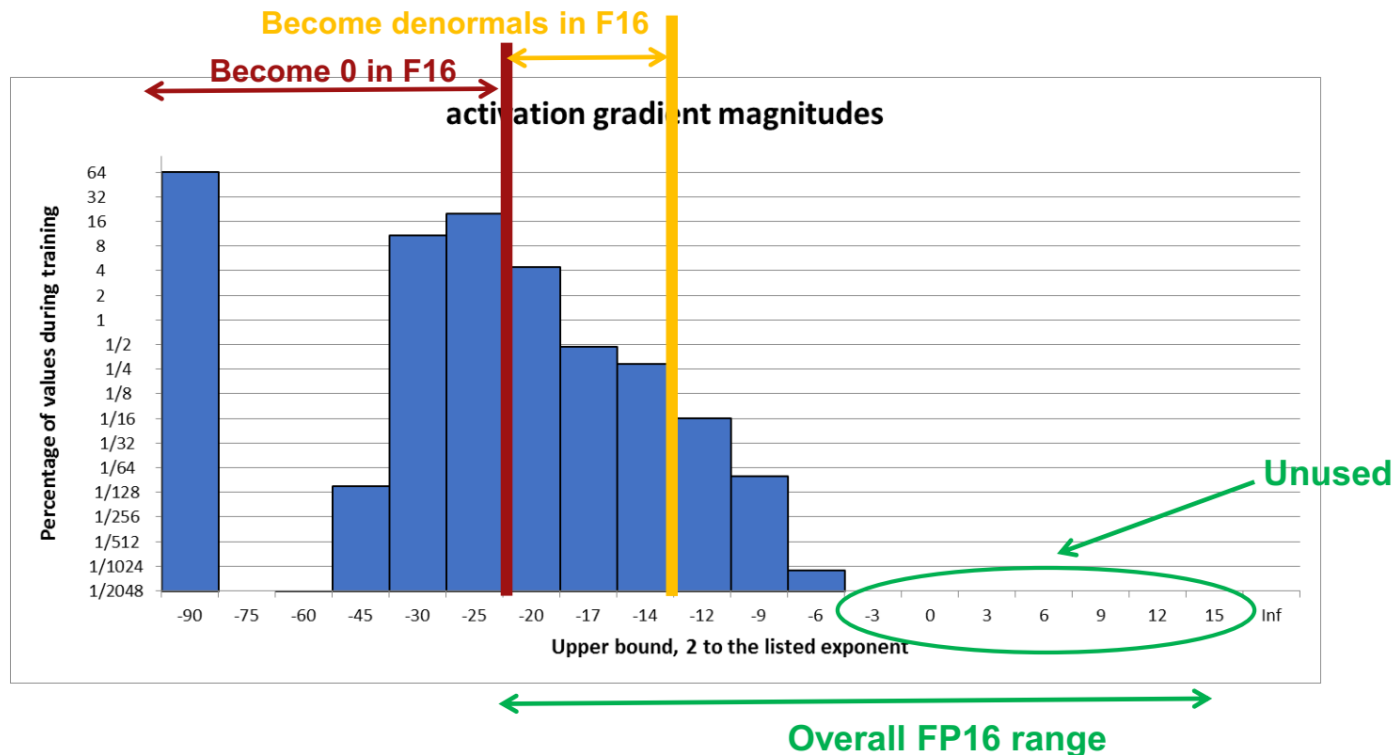
- 现象：有些模型仍无法收敛（如Multibox SSD with VGG-D）或准确率很差（如Seq2Seq）
- 原因：激活梯度的值很小，超出FP16表示的范围（Underflow）



图为使用FP32进行Multibox SSD with VGG-D训练过程中，激活梯度（Activation Gradient）的幅度直方图

Credit: NVIDIA "Training with mixed precision" Ginsburg et al.

激活梯度的数值分布并不对称，未充分利用FP16的动态范围



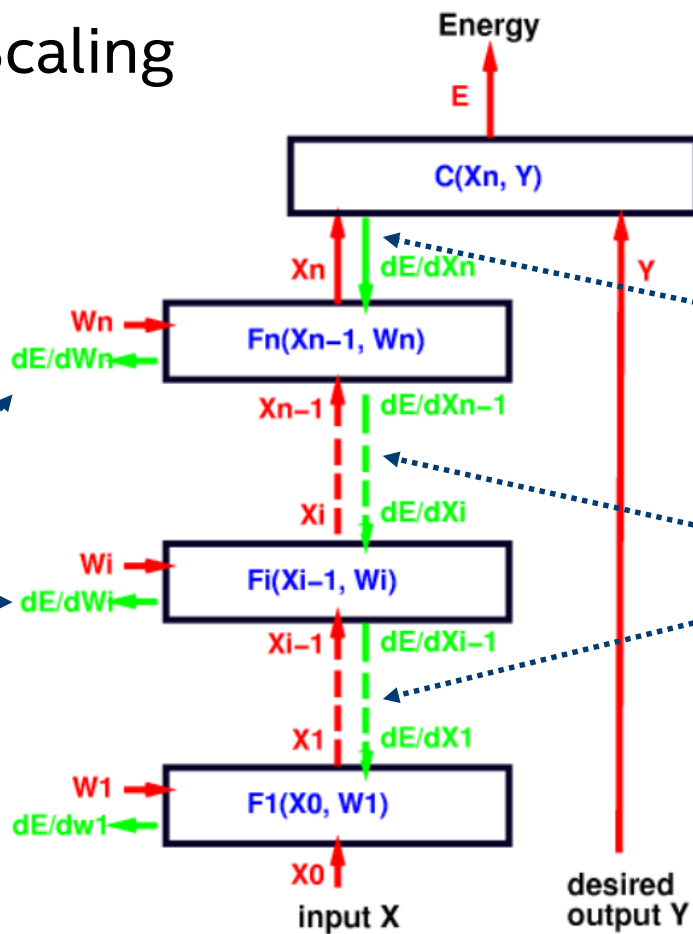
Credit: NVIDIA "Training with mixed precision" Ginsburg et al.

解决激活梯度下溢出的思路

- 做什么：将所有的激活梯度都放大一定倍数（如256倍）
- 如何做：
 - 在反向计算开始前，将损失变化（ $dLoss$ ）人为增大
 - 基于Chain Rule，反向计算得到的所有梯度（激活/权重）均放大了相同倍数
 - 在权重更新前，将权重梯度缩小成正常值

损失放大 Loss Scaling

基于Chain Rule，反向计算得到的权重梯度也放大了 2^K 倍，在进行权重更新前需要将其缩小 2^K 倍，确保权重的正确更新

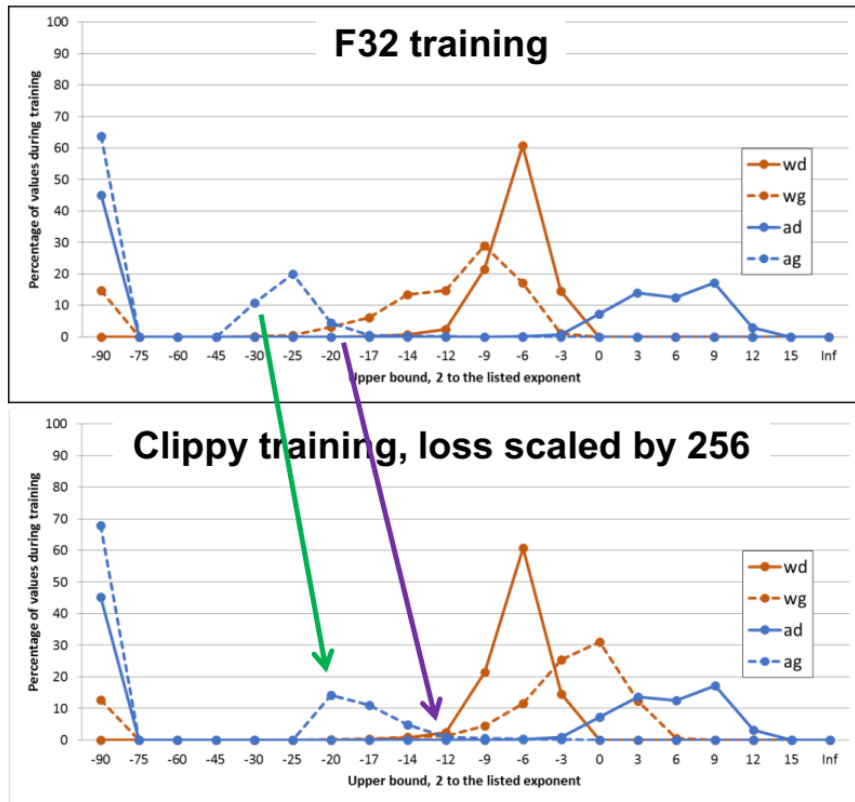


反向传播前，将损失变化增大 2^K 倍

基于Chain Rule，反向计算得到的激活梯度放大了 2^K 倍，从而落入FP16的动态范围

Credit: Yann LeCun ICML'13 tutorial

观察使用损失放大后梯度数值范围的变化



使用256 (2^8) 倍的损失放大后，激活梯度和权重梯度都放大了256倍。激活梯度不再因为超出FP16的动态范围而下溢出

Credit: NVIDIA "Training with mixed precision" Ginsburg et al.

使用损失放大的效果

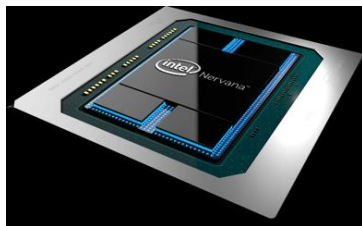
模型	FP32基准	FP16混合精度 (无损失放大)	FP16混合精度 (有损失放大)
Faster R-CNN	69.1%	68.6%	69.7%
Multibox SSD	76.9%	无法收敛	77.1%

目标检测任务。Faster R-CNN和Multibox SSD均采用VGG-16。mAP使用Pascal VOC 2007测试集。

Credit: Paulius Micikevicius et al. (2017)

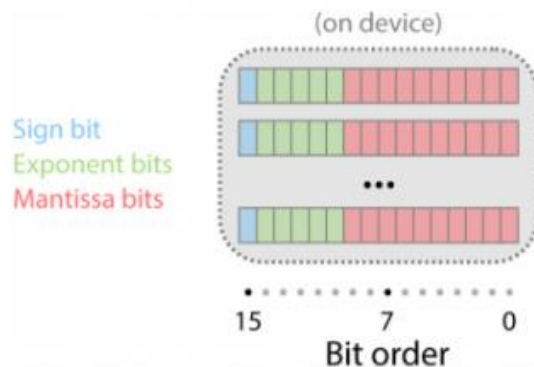
目录

1. 为什么需要低精度
2. 低精度带来的问题
3. FP16/FP32混合精度训练
4. 更多的创新 (FlexPoint16, DFP-16, bfloat16)
5. 小结

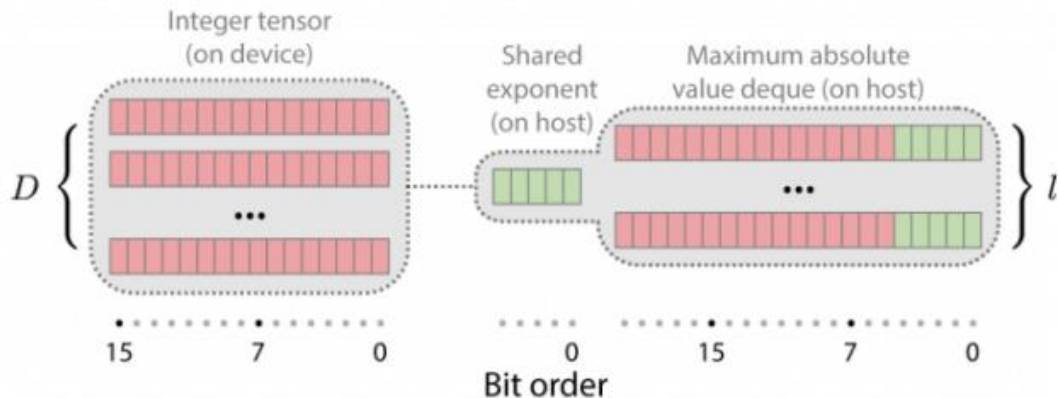


Intel NNP ASIC中使用的Flexpoint格式

(a) float16 tensor



(b) flex16+5 tensor

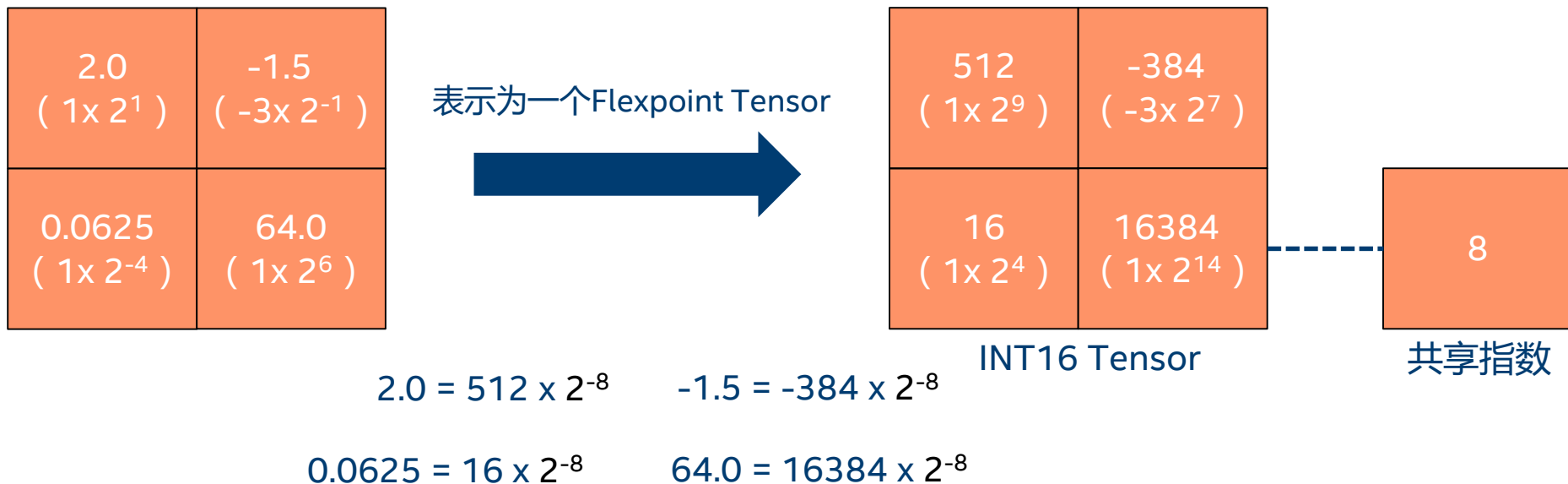


一个Flexpoint Tensor (flex16+5) = 一个INT16 Tensor + 一个5 bit的指数 + 一个队列 (记录该Tensor最大值历史数据)
(全Tensor共享)

Credit: ai.intel.com "Flexpoint: numerical innovation" Xin Wang et al.

Flexpoint实质

是**定点数** (Fixed Point) ,
而非浮点数 (Floating Point)



优势：Flexpoint Tensor运算都是整形运算

$$\begin{array}{|c|c|} \hline I_a \\ \hline E_a \end{array} \times \begin{array}{|c|c|} \hline I_b \\ \hline E_b \end{array} = \begin{array}{|c|c|} \hline I_a \times I_b \\ \hline E_a + E_b \end{array}$$

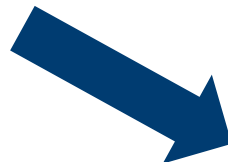
$$\begin{array}{|c|c|} \hline I_a \\ \hline E_a \end{array} + \begin{array}{|c|c|} \hline I_b \\ \hline E_b \end{array} = \begin{array}{|c|c|} \hline I_a + (I_b \gg (E_a - E_b)) \\ \hline \text{Max}(E_a, E_b) \end{array} \quad \text{当 } E_a > E_b$$

$$\text{or} \quad \begin{array}{|c|c|} \hline I_b + (I_a \gg (E_b - E_a)) \\ \hline \text{Max}(E_a, E_b) \end{array} \quad \text{当 } E_b > E_a$$

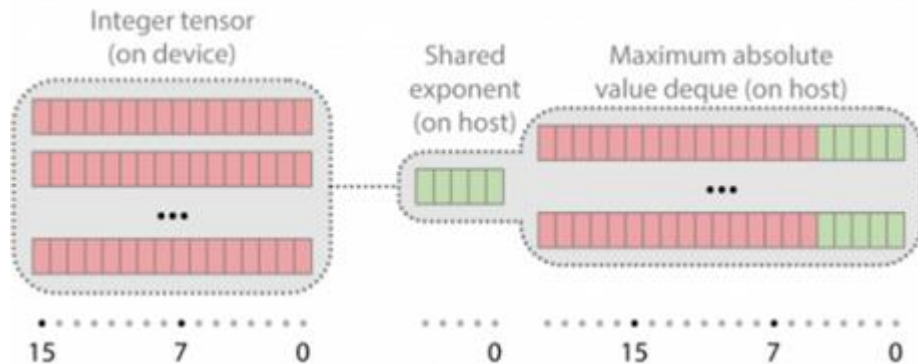
没有免费的午餐



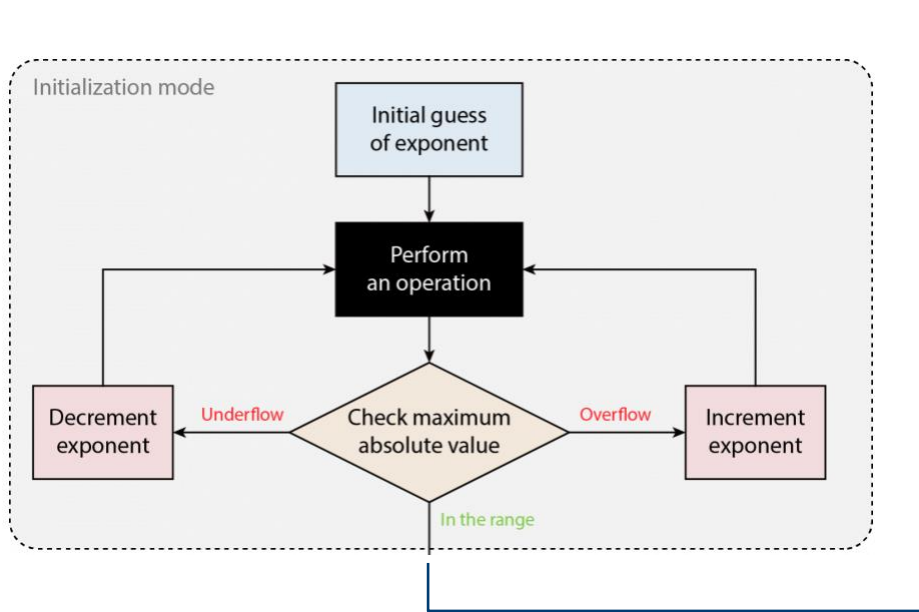
由于指数(exponent)部分由Tensor的所有元素共享，Tensor的动态范围受到限制（最大值和最小值之间不能超过 2^{15} 倍）



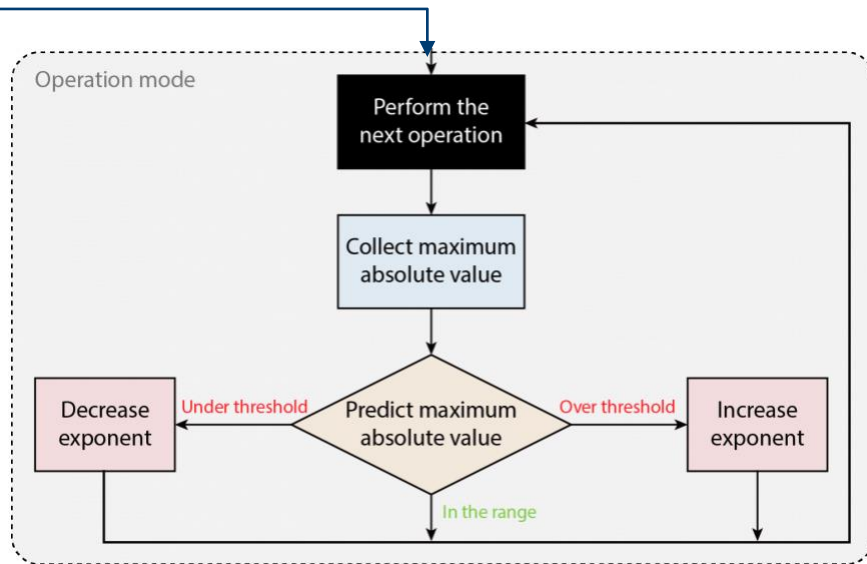
当Tensor元素的数值范围发生变化而无法用当前指数表示时，要能够动态对指数进行调整



Autoflex – 指数管理算法



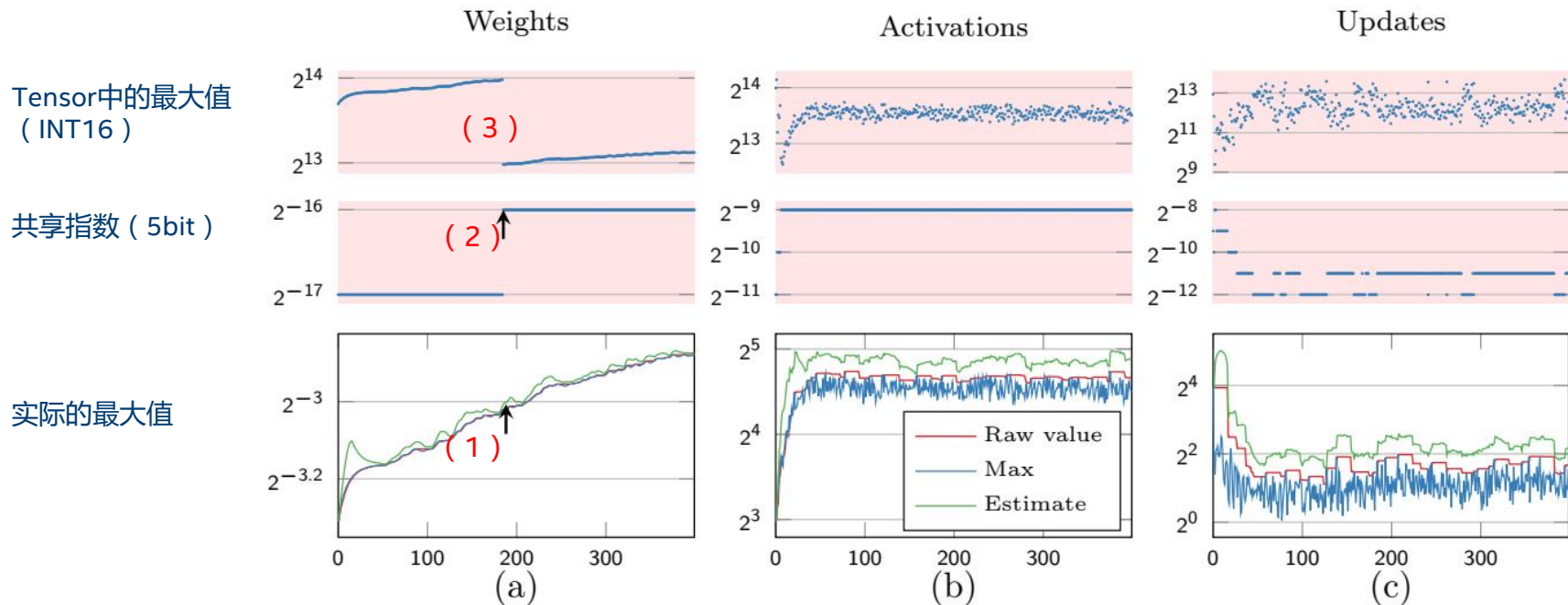
初始化：开始时，用于记录Tensor中最大值（绝对值）历史的队列为空。通过猜测-调整将共享指数初始化为“合适”的值（尽量利用16位manissa，但留有2~3bit的空间）



操作：在每次Tensor的写入后，根据最近的最大值历史信息（如反应变化剧烈有否的方差）预测下一个最大值的可能范围，从而决定是否对当前指数进行调整

Autoflex – 预测最大值变化并提前调整指数

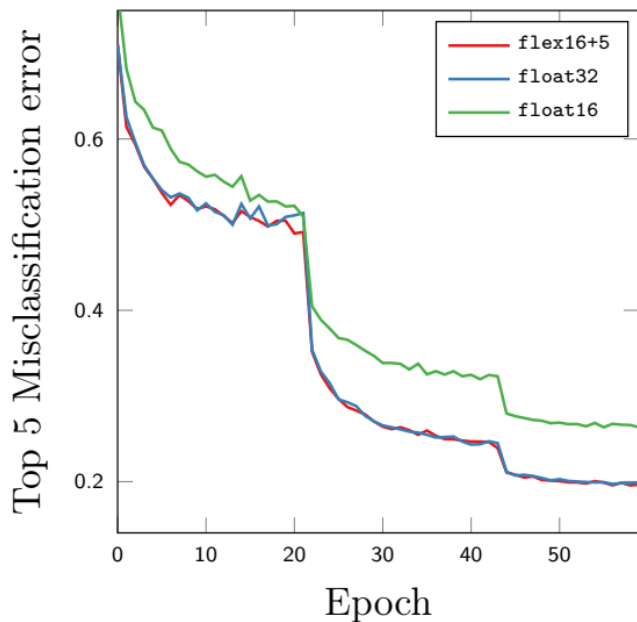
(1) 根据历史信息，预测下一个最大值即将超过阈值 (2) 增大共享指数 (3) 调整Tensor里的每一个整形值



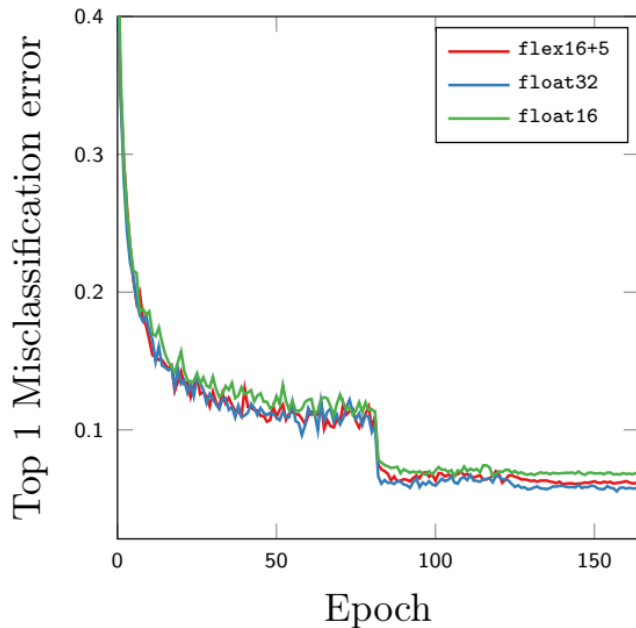
图为使用Flexpoint在CIFAR-10数据集上训练一个小的2层感知机

Credit: Urs Koster et al. (2017)

使用Flexpoint训练CNN



(a) ImageNet1k AlexNet



(b) CIFAR-10 ResNet

图为使用Flexpoint在

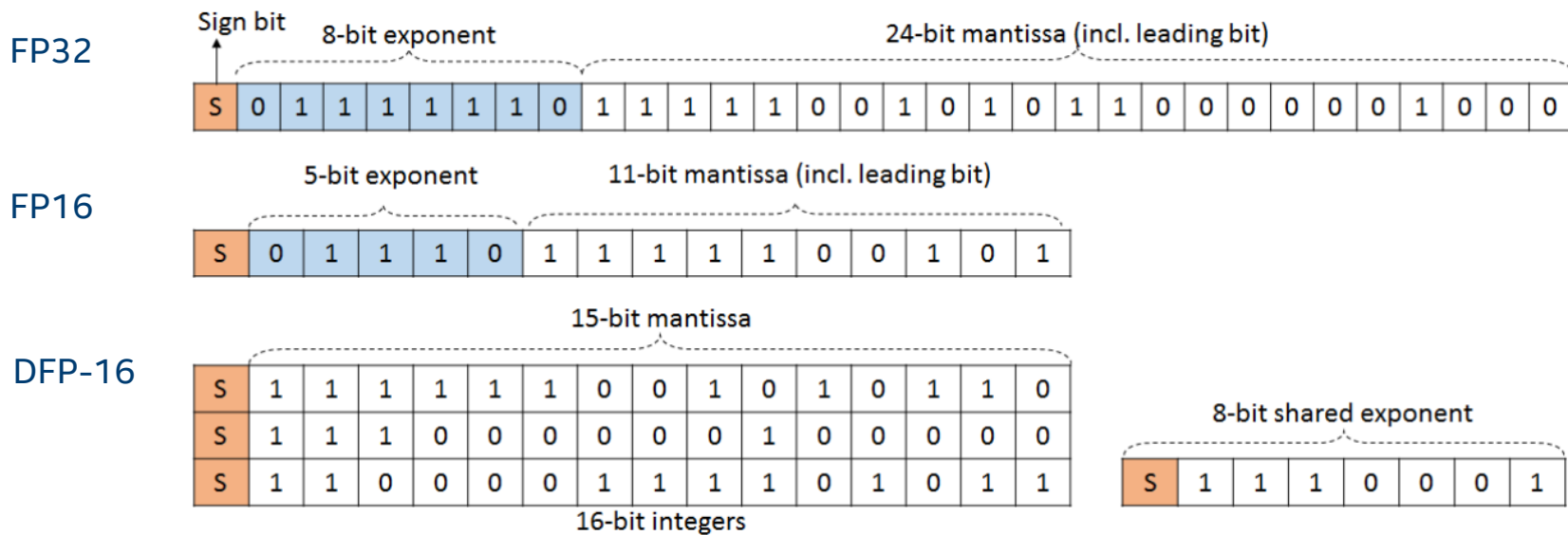
(a) ImageNet1K数据集上训练AlexNet后的Top-5误判率

(b) CIFAR-10数据集上训练ResNet-110后的Top-1误判率

Credit: Urs Koster et al. (2017)

其它的16bit格式：DFP-16

在通用CPU上使用DFP-16进行深度学习训练

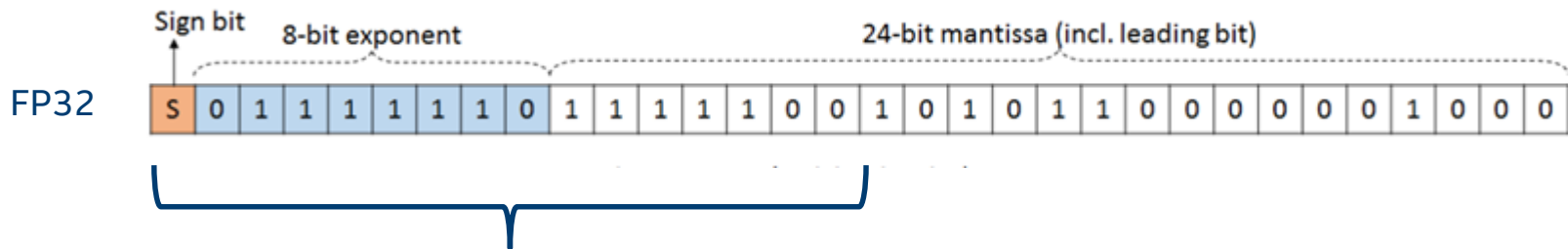


DFP-16(Dynamic Fix Point)和Flexpoint类似，都是定点数，也都是整个Tensor共享一个指数。不同的是DFP-16采用了8bit的指数位，从而保持了和FP32相同的动态范围，模型训练相对更容易收敛

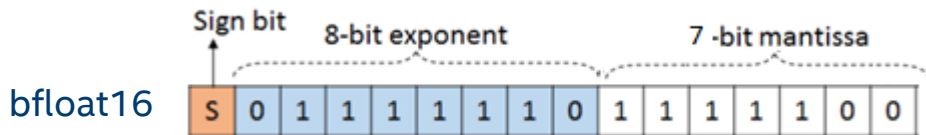
Credit: Dipankar Das et al. ICLR 2018

其它的16bit格式： bfloat16

Google TPU / TensorFlow支持的16位格式



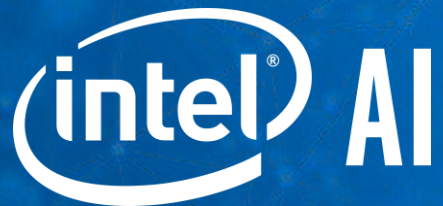
直接截取(Truncate)前16位



不同于IEEE754的标准FP16，bfloat16直接截取FP32的前16位，保持了相同的动态范围（8bit指数）但更多地牺牲精度（mantissa只有7bit）。更适用于表示（动态范围大，不易溢出），而不是运算（精度低，容易发生舍入错误）。

小结

- 深度学习训练使用16bit表示/运算正逐渐成为主流
- 低精度带来了性能、功耗优势，但需要解决量化误差（溢出、舍入）
- 常见的避免量化误差的方法
 - 为权重保持高精度（FP32）备份
 - 损失放大，避免梯度的下溢出
 - 一些特殊层（如BatchNorm）仍使用FP32运算
- 使用16位定点数（如Flexpoint16, DFP-16）的关键是指数管理（避免Tensor最大值溢出）



参与大会现场
互动赢取礼品



扫码参与英特尔
全程参与奖



欢迎参观英特尔的展览 展位号：100