LOW PRECISION INFERENCE ON GPU

Hao Wu, NVIDIA

OUTLINE

- Performance motivation for quantization
- Quantization details
- Post-training quantization accuracy
- Training for quantization

INFERENCE

(sometimes called serving)

- **Inference: using a trained model to make predictions**
  - Much of inference is fwd pass in training
- **Inference engines**
  - Apply optimizations not common in training frameworks
    - Layer fusion, batch normalization folding
    - Memory management optimized for inference
    - Quantization
  - TensorRT: NVIDIA's platform for inference
    - https://developer.nvidia.com/tensorrt
    - Available as a stand-alone and in TensorFlow
      - S9431 - TensorRT Inference with Tensorflow (Wednesday, Mar 20, 10:00 AM)

QUANTIZED INFERENCE

- **Quantization:**
  - Using lower precision to represent weights and activations
  - Using lower precision math
- **Benefits:**
  - Speed up inference:
    - Math limited layers due to higher throughput math
    - Memory limited layers due to bandwdith savings
  - Reduce resource requirements: memory footprint, etc.
- **Challenge:**
  - Maintaining model accuracy

TURING MATH THROUGHPUT

Relative to fp32 math

Input Type	Accumulation Type	Relative math throughput	Bandwidth savings
FP16	FP16	8x	2x
INT8	INT32	16x	4x
INT4	INT32	32x	8x
INT1	INT32	128x	32x

INFERENCE SPEEDUPS OVER FP32

TensorRT on Tesla T4 GPU

Input size 224x224 for all, except 299x299 for Inception networks

	Batch size 1			Batch size 8			Batch size 128		
	FP32	FP16	Int8	FP32	FP16	Int8	FP32	FP16	Int8
MobileNet v1	1	1.91	2.49	1	3.03	5.50	1	3.03	6.21
MobileNet v2	1	1.50	1.90	1	2.34	3.98	1	2.33	4.58
ResNet50 (v1.5)	1	2.07	3.52	1	4.09	7.25	1	4.27	7.95
VGG-16	1	2.63	2.71	1	4.14	6.44	1	3.88	8.00
VGG-19	1	2.88	3.09	1	4.25	6.95	1	4.01	8.30
Inception v3	1	2.38	3.95	1	3.76	6.36	1	3.91	6.65
Inception v4	1	2.99	4.42	1	4.44	7.05	1	4.59	7.20
ResNext101	1	2.49	3.55	1	3.58	6.26	1	3.85	7.39

INFERENCE THROUGHPUT IN IMAGES/S

Input size 224x224 for all, except 299x299 for Inception networks

Image/s	Batch size 1			Batch size 8			Batch size 128		
	FP32	FP16	Int8	FP32	FP16	Int8	FP32	FP16	Int8
MobileNet v1	1509	2889	3762	2455	7430	13493	2718	8247	16885
MobileNet v2	1082	1618	2060	2267	5307	9016	2761	6431	12652
ResNet50 (v1.5)	298	617	1051	500	2045	3625	580	2475	4609
VGG-16	153	403	415	197	816	1269	236	915	1889
VGG-19	124	358	384	158	673	1101	187	749	1552
Inception v3	156	371	616	350	1318	2228	385	1507	2560
Inception v4	76	226	335	173	768	1219	186	853	1339
ResNext101	84	208	297	200	716	1253	233	899	1724

INFERENCE IN FP16

- Training in fp32 and inference in fp16 is expected to get same accuracy as in fp32 most of the time
- Add normalization if it overflows (>65504)
  - Add batch normalization to activation
  - If it is integer RGB input (0~255), normalize it to be float (0~1)

QUANTIZATION DETAILS

- Terminology
- Choices:
  - Scale vs scale+shift (symmetric vs asymmetric quantization)
  - Signed vs unsigned integer quantized representation
  - Scaling factor
  - Scaling granularity
  - Operations to quantize

TERMINOLOGY

- Quantize: convert from full precision (FP32) to quantized integer representation (e.g. int8)
- Dequantize: convert from quantized representation to full precision
- Requantize: convert from one quantized representation to another
  - Effectively dequantize then quantize to a different quantized representation
  - Useful when output is being converted for quantized input of another operation

SCALE VS SCALE+SHIFT QUANTIZATION

Symmetric vs Asymmetric quantization

- **Determined by the range of real values being quantized**
- **Scale(Symmetric) quantization:**
  - Quantize a range symmetrically centered at 0
  - Examples: [-3.2, 3.2], [-100.0, 100.0]
- **Scale+Shift(Asymmetric) quantization:**
  - Quantize an arbitrary range
  - Examples: [-5.1, 8.3], [0.0, 20.0]

SCALE QUANTIZATION

Also known symmetric quantization

- Quantized range represents a 0 centered real range
- Given tensor $\mathbf{y}$, quantized tensor $\mathbf{y_q}$ is defined as

$$\mathbf{y_q} = rn\left(s \cdot clip(\mathbf{y}, -\alpha, \alpha)\right)$$

where:

rn() is round to nearest

s is scaling factor

α is clipping threshold

$$clip(x) = \begin{cases} -\alpha & , x \in (-\infty, -\alpha) \\ x & , x \in [-\alpha, \alpha) \\ \alpha & , x \in [\alpha, \infty] \end{cases}$$

Example:

Quantize to 4 bit with $\alpha = 2$

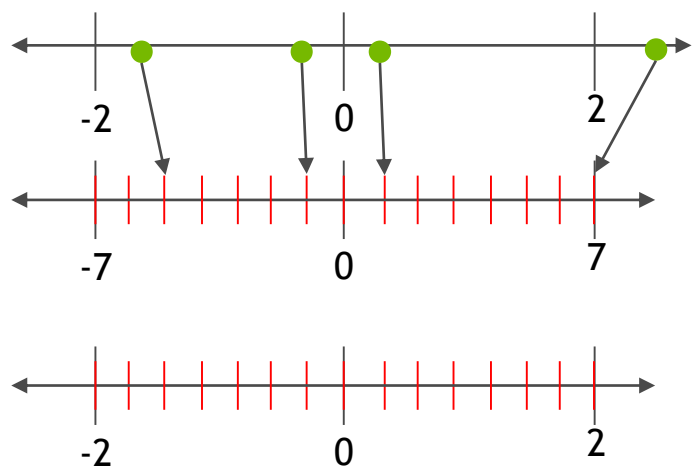
$$\begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 2.5 \end{pmatrix}$$

Quantize

$$\begin{pmatrix} -5 & 1 \\ -1 & 7 \end{pmatrix}$$

Dequantize

$$\begin{pmatrix} -1.43 & 0.28 \\ -0.28 & 2 \end{pmatrix}$$



SCALE+SHIFT QUANTIZATION

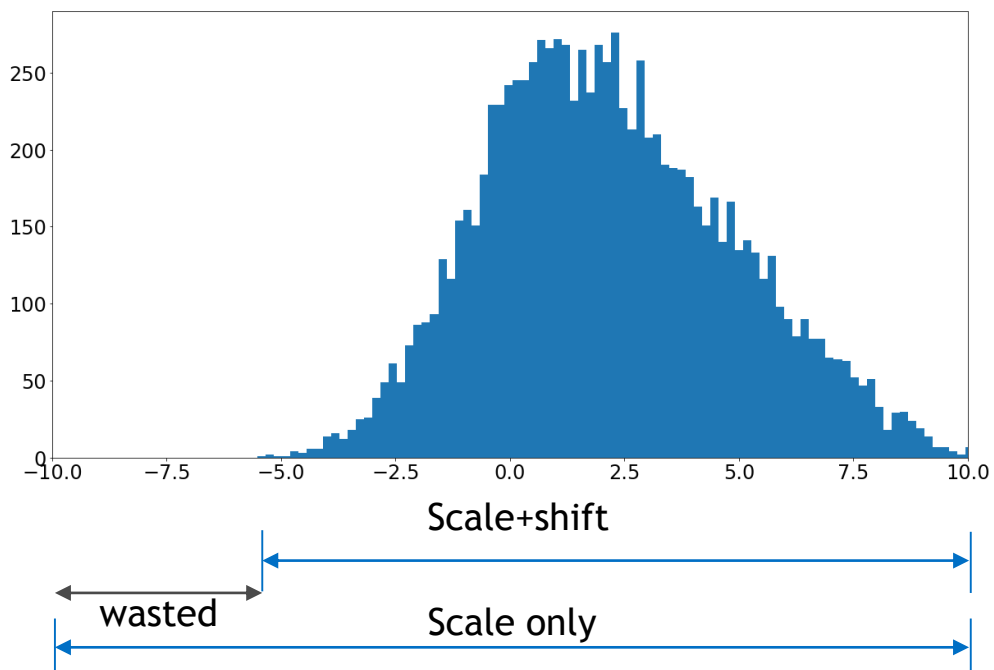
Also known as asymmetric quantization

- Quantized range represents a non 0-centered real range
- Given tensor $\mathbf{y}$, quantized tensor $\mathbf{y_q}$ is defined as
- $$\mathbf{y_q} = rn(s \cdot (clip(\mathbf{y}, \beta, \alpha) + z))$$

  where:
  rn() is round to nearest
  s is scaling factor
  z is shift (zero point)
  α and β are clipping threshold

$$clip(\mathbf{y}, \beta, \alpha) = \begin{cases} \beta & , x \in (-\infty, \beta) \\ x & , x \in [\beta, \alpha) \\ \alpha & , x \in [\alpha, \infty] \end{cases}$$

Could use bits more efficiently when distribution is not 0-centered



SCALE+SHIFT OFFERS LITTLE ACCURACY BENEFIT

Image Classification, top-1 accuracy

	FP32	Int8 Scale	Int8 Scale+Shift
Mobilenet-v1_1_224	70.90	70.70	70.00
Mobilenet-v2_1_224	71.90	71.10	70.90
Nasnet-Mobile	74.00	73.00	73.00
Mobilenet-v2_1.4_224	74.90	74.50	73.50
Inception-v3	78.00	78.00	78.00
Resnet-v1_50	75.20	75.00	75.00
Resnet-v2_50	75.60	75.00	75.00
Resnet-v1_152	76.80	76.20	76.50

Classificatoin data from https://arxiv.org/abs/1806.08342

Object Detection, mAP

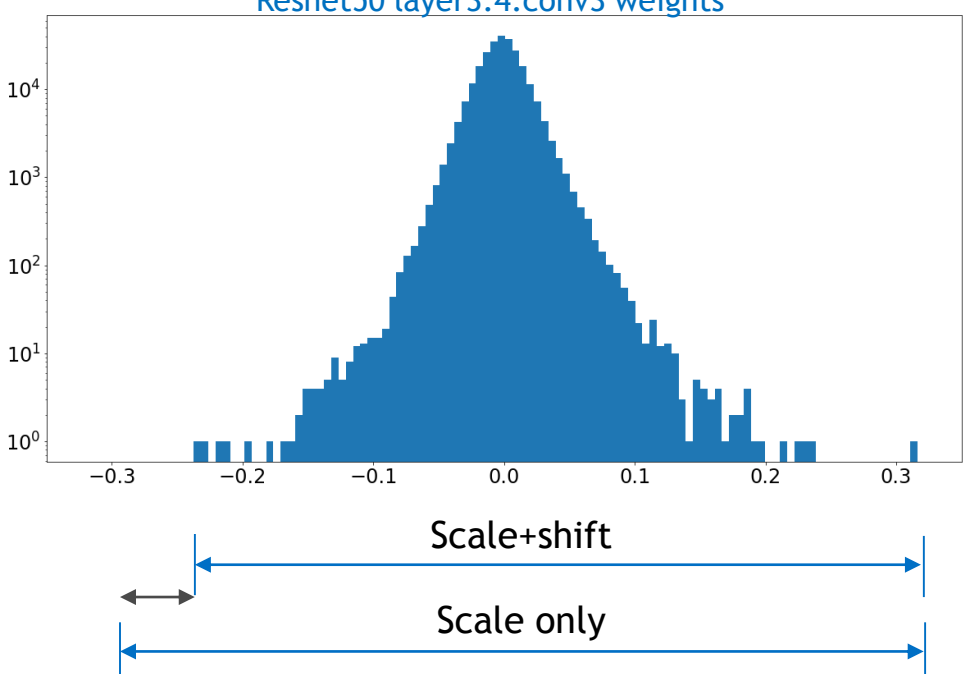
	FP32	Int8 Scale	Int8 Scale+Shift
faster_rcnn_resnet101_coco*	0.38	0.37	0.38
faster_rcnn_nas_coco*	0.56	0.55	0.55
faster_rcnn_inception_v2_coco	0.28	0.28	0.279

SCALE+SHIFT OFFERS LITTLE ACCURACY BENEFIT

Tensors with positive and negative values:

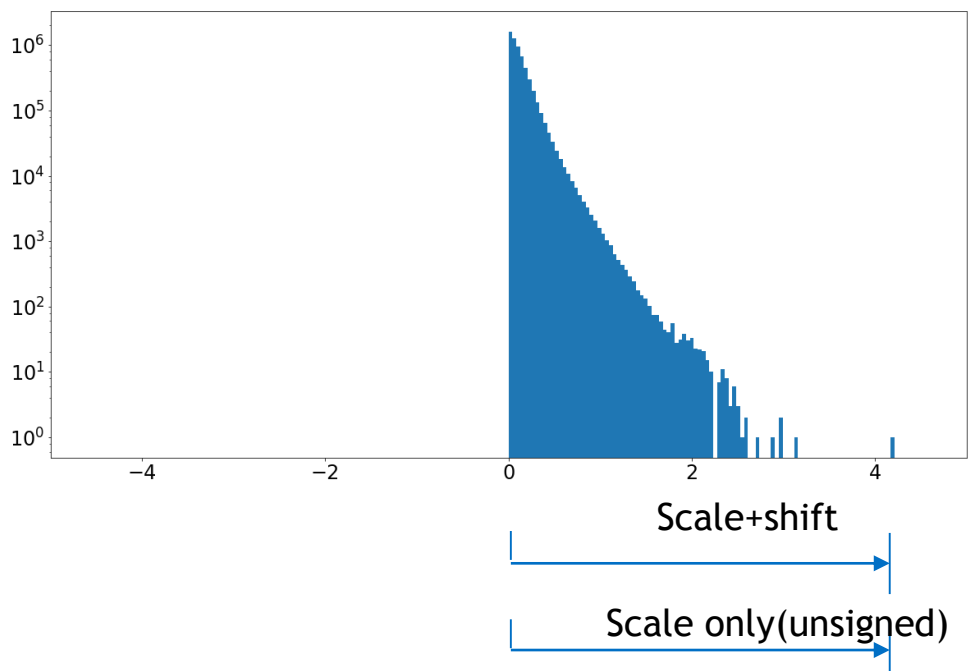
- Typically centered near 0
- Outliers cause assymetry of range,

Resnet50 layer3.4.conv3 weights



Tensors with only positive values:

- Scale-only with unsigned int is just as efficient



SCALE+SHIFT IS MORE EXPENSIVE

- With scale quantization, output is simply a scaled version of “true” output:
  - $s_A A * s_B B = s_A s_B AB$
- For scale+shift quantization, the output contains *four* distinct terms (t = shift):
  - $(s_A A + t_A) * (s_B B + t_B) = s_A s_B AB + s_A(A + t_B) + s_B(B + t_A) + t_A t_B$
  - The operations involved to compute 3 additional terms may eliminate the performance advantage of 8bit quantization over fp16
    - At least 1 more path to go through entire activation tensor
    - Detail can be found at https://github.com/google/gemmlowp

CONCLUSION: USE SCALE QUANTIZATION

- Faster than scale+shift
- Accuracy within epsilon of scale+shift
  - higher for some networks, lower for some others
- Optionally use unsigned int for tensors with only positive values (doubles the sample points)
- Quantize to a symmetric range of integer values to avoid bias
  - Do not use the minimum negative value
  - Given *k* bits, use symmetric range $[-(2^{k-1}-1), 2^{k-1}-1]$, $s = \frac{2^{k-1}-1}{\alpha}$. E.g. [-127, 127] for 8-bit

MINIMUM QUANTIZED VALUE

- Integer range is not completely symmetric. E.g. in 8bit, [-128, 127]
  - If use [-127, 127], $s = \frac{127}{\alpha}$
    - Range is symmetric
    - 1/256 of int8 range is not used. 1/16 of int4 range is not used
  - If use full range [-128, 127], $s = \frac{128}{\alpha}$
    - Values should be quantized to 128 will be clipped to 127
    - Asymmetric range may introduce bias

EXAMPLE OF QUANTIZATION BIAS

Bias introduced when int values are in [-128, 127]

$$A = [-2.2 \quad -1.1 \quad 1.1 \quad 2.2] \text{ , } B = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.3 \\ 0.5 \end{bmatrix} \text{ , } AB = 0$$

8bit scale quantization, use [-128, 127]. s_A=128/2.2, s_B=128/0.5

$$[-128 \quad -64 \quad 64 \quad 127] * \begin{bmatrix} 127 \\ 77 \\ 77 \\ 127 \end{bmatrix} = -127$$

Dequantize -127 will get -0.00853. A small bias is introduced towards -∞

EXAMPLE OF QUANTIZATION BIAS

No bias when int values are in [-127, 127]

$$A = [-2.2 \quad -1.1 \quad 1.1 \quad 2.2] \text{ , } B = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.3 \\ 0.5 \end{bmatrix} \text{ , } AB = 0$$

8-bit scale quantization, use [-127, 127]. s_A=127/2.2, s_B=127/0.5

$$[-127 \quad -64 \quad 64 \quad 127] * \begin{bmatrix} 127 \\ 76 \\ 76 \\ 127 \end{bmatrix} = 0$$

Dequantize 0 will get 0

MATRIX MULTIPLY EXAMPLE

Scale Quantization

$$\begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 0.65 \end{pmatrix} * \begin{pmatrix} 0.35 \\ -0.51 \end{pmatrix} = \begin{pmatrix} -0.651 \\ -0.423 \end{pmatrix}$$

MATRIX MULTIPLY EXAMPLE

Scale Quantization

$$\begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 0.65 \end{pmatrix} * \begin{pmatrix} 0.35 \\ -0.51 \end{pmatrix} = \begin{pmatrix} -0.651 \\ -0.423 \end{pmatrix}$$

8bit quantization

choose [-2, 2] fp range (scale 127/2=63.5) for first matrix and [-1, 1] fp range (scale = 127/1=127) for the second

$$\begin{pmatrix} -98 & 14 \\ -17 & 41 \end{pmatrix} * \begin{pmatrix} 44 \\ -65 \end{pmatrix} = \begin{pmatrix} -5222 \\ -3413 \end{pmatrix}$$

MATRIX MULTIPLY EXAMPLE

Scale Quantization

$$\begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 0.65 \end{pmatrix} * \begin{pmatrix} 0.35 \\ -0.51 \end{pmatrix} = \begin{pmatrix} -0.651 \\ -0.423 \end{pmatrix}$$

8bit quantization

choose [-2, 2] fp range (scale 127/2=63.5) for first matrix and [-1, 1] fp range (scale = 127/1=127) for the second

$$\begin{pmatrix} -98 & 14 \\ -17 & 41 \end{pmatrix} * \begin{pmatrix} 44 \\ -65 \end{pmatrix} = \begin{pmatrix} -5222 \\ -3413 \end{pmatrix}$$

The result has an overall scale of 63.5*127 . We can *dequantize* back to float

$$\begin{pmatrix} -5222 \\ -3413 \end{pmatrix} * \frac{1}{63.5 * 127} = \begin{pmatrix} -0.648 \\ -0.423 \end{pmatrix}$$

REQUANTIZE

Scale Quantization

$$\begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 0.65 \end{pmatrix} * \begin{pmatrix} 0.35 \\ -0.51 \end{pmatrix} = \begin{pmatrix} -0.651 \\ -0.423 \end{pmatrix}$$

8bit quantization

choose [-2, 2] fp range for first matrix and [-1, 1] fp range for the second

$$\begin{pmatrix} -98 & 14 \\ -17 & 41 \end{pmatrix} * \begin{pmatrix} 44 \\ -65 \end{pmatrix} = \begin{pmatrix} -5222 \\ -3413 \end{pmatrix}$$

Requantize output to a different quantized representation with fp range [-3, 3]:

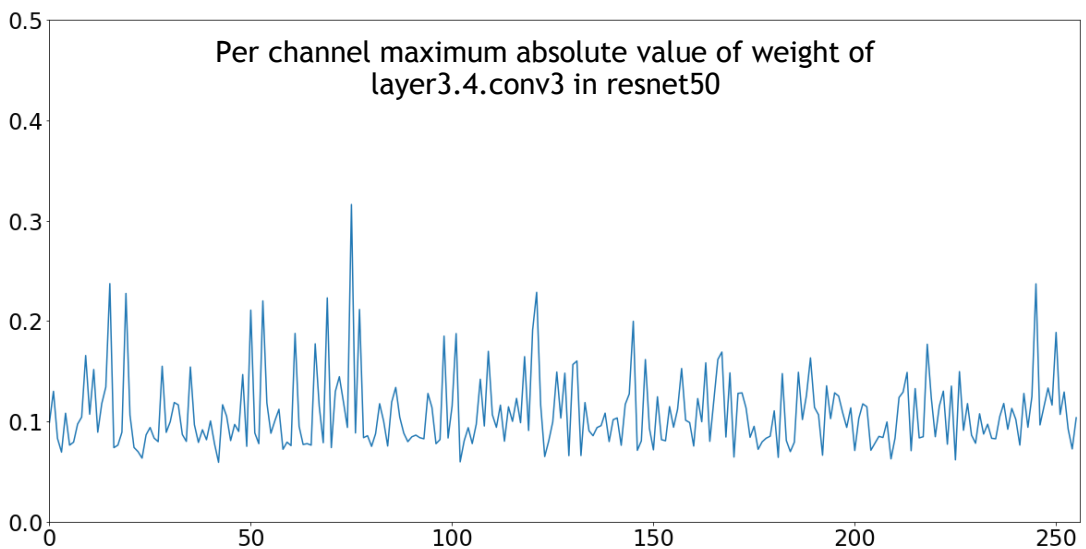
$$\begin{pmatrix} -5222 \\ -3413 \end{pmatrix} * \frac{127/3}{63.5 * 127} = \begin{pmatrix} -27 \\ -18 \end{pmatrix}$$

CHOOSING SCALE GRANULARITY

- Granularity for scaling choices:
  - Per tensor scale: all values in a tensor share a range
  - Fine-grain scale:
    - Values in a channel share scale
    - Different channels can have different scales
    - Can be extended to any axis of a tensor has its own scale

FINE GRAINED SCALE QUANTIZATION

- Why do we need fine scale?
  - Weight distribution varies per channel/neuron

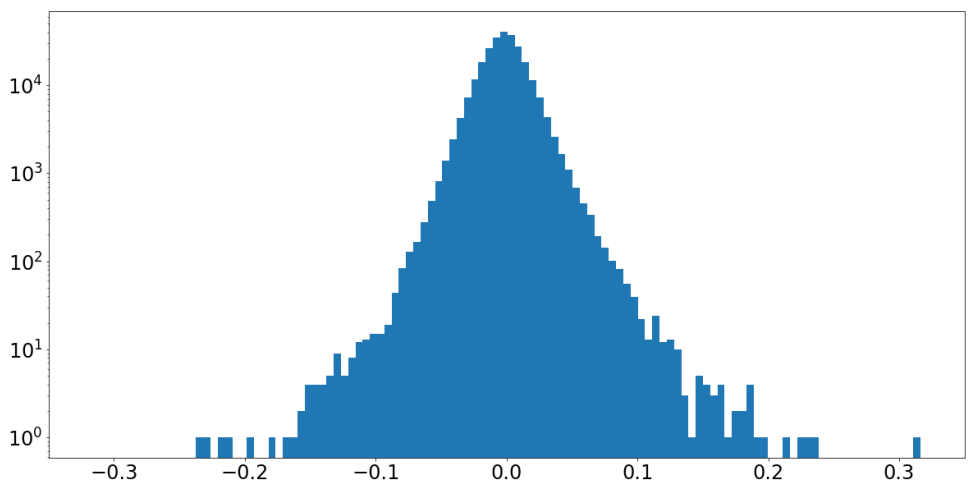


CHOOSING SCALE GRANULARITY (CONT.)

- Scale must be decided offline. Computing scale inflight will eliminate the performance advantage of int8 over fp16
- Per tensor (matrix) scale for activations
  - Each input in a batch can have different scale, can’t be decided offline
  - Each input feature map of activation must have same scale to do dot product
- Fine grained scale for weight
  - Can be decided offline
  - Per channel scale for convolution weight
  - Per neuron scale for fully connected weight

CHOOSING THE SCALE/RANGE

Using full range may not be the best choice for quantization – large outliers reduce resolution around 0.



$\alpha = 0.23$, clip outliers, has more precision close to 0

$\alpha = 0.31$, covers the full range

CALIBRATION

- Calibration
  - Feed data samples to the network, decide scaling factor for each activation tensor
  - Data samples must be representative of inference workload. A subset of training set is usually used
- Calibration method
  - Max value
    - Use the global maximum absolute value of all tensors seen in calibration
    - If activation is clipped during training, use the clipping threshold. E.g. ReLU6
  - Entropy. Developed by TensorRT for CNNs
    - Minimize the information loss between the original tensor and quantized tensor by KL-divergence
    - See http://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf

EXAMPLE OF FINE GRAINED SCALE

$$s_A \begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 0.65 \end{pmatrix} * s_B \begin{pmatrix} 0.35 \\ -0.51 \end{pmatrix} = s_A s_B \begin{pmatrix} -0.65 \\ -0.42 \end{pmatrix}$$

As written, each row (“neuron” / “channel”) of A acts independently on output. We can use a distinct scale for each:

$$\begin{pmatrix} s_{A1} \\ s_{A2} \end{pmatrix} \begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 0.65 \end{pmatrix} * s_B \begin{pmatrix} 0.35 \\ -0.51 \end{pmatrix} = \begin{pmatrix} s_{A1} s_B \\ s_{A2} s_B \end{pmatrix} \begin{pmatrix} -0.65 \\ -0.42 \end{pmatrix}$$

Small increase in bookkeeping math, usually a few percent performance overhead

Extends naturally to convolution as well as matrix multiply

CHOOSING OPERATIONS TO QUANTIZE

- Quantize:
  - Math-intensive operations: Matrix Multiply (fully-connected layers), Convolution
  - Other operations can be done in quantized space
    - Avoids dequantization and quantization
    - Example operations: ReLU, Pooling
- Do not quantize:
  - Computation of nonlinear operations, e.g. Softmax, tanh, sigmoid, GeLU etc.
  - Inexpensive layers

SUMMARY AND RECOMMENDATION

- Use scale only quantization, no shift
- Do not use the minimum negative value in quantized range
  - Use symmetric range $[-2^{k-1}-1, 2^{k-1}-1]$, $s = \frac{2^{k-1}-1}{\alpha}$, where k is number of bits used in quantized representation. E.g. [-127, 127] for 8-bit
- Per tensor activation scale
  - Run calibration to chose best scaling factor
- Fine grained weight scale
  - Use maximum absolute value (full range) to compute scaling factor for 8-bit quantization

POST TRAINING QUANTIZATION RESULTS

Int8 quantization

- Different task types:
  - Classification
  - Regression
- Different tasks:
  - Images: classification, detection, segmentation
  - Language translation

IMAGE CLASSIFICATION

Model	FP32	Int8 (max)	Rel Err %
MobileNet v1	71.01	69.46	2.18%
MobileNet v2	74.08	73.96	0.16%
NASNet (large)	82.72	82.09	0.76%
NASNet (mobile)	73.97	12.95	82.49%
ResNet50 (v1.5)	76.51	76.11	0.52%
ResNet50 (v2)	76.37	75.73	0.84%
ResNet152 (v1.5)	78.22	5.29	93.24%
ResNet152 (v2)	78.45	78.05	0.51%
VGG-16	70.89	70.75	0.20%
VGG-19	71.01	70.91	0.14%
Inception v3	77.99	77.7	0.37%
Inception v4	80.19	1.68	97.90%

- With max calibration, some networks have outliers which ruin quantization completely

All results percentage top-1 accuracy on Imagenet validation set. Measured by TFTRT
Models are from https://github.com/tensorflow/models/tree/master/research/slim and https://github.com/tensorflow/models/tree/master/official/resnet

CLASSIFICATION

Model	FP32	Int8 (max)	Int8 (entropy)	Rel Err (entropy)
MobileNet v1	71.01	69.43	69.46	2.18%
MobileNet v2	74.08	73.96	73.85	0.31%
NASNet (large)	82.72	82.09	82.66	0.07%
NASNet (mobile)	73.97	12.95	73.4	0.77%
ResNet50 (v1.5)	76.51	76.11	76.28	0.30%
ResNet50 (v2)	76.37	75.73	76.22	0.20%
ResNet152 (v1.5)	78.22	5.29	77.95	0.35%
ResNet152 (v2)	78.45	78.05	78.15	0.38%
VGG-16	70.89	70.75	70.82	0.10%
VGG-19	71.01	70.91	70.85	0.23%
Inception v3	77.99	77.7	77.85	0.18%
Inception v4	80.19	1.68	80.16	0.04%

- With max calibration, some networks have outliers which ruin quantization completely
- With entropy calibration, accuracy drops are below 1% relative, except MobileNet V1.

OBJECT DETECTION

COCO

Model	Backbone	FP32	INT8	Rel Err
SSD-300	MobileNet v1	26	25.8	0.77%
SSD-300	MobileNet v2	27.4	26.8	2.19%
Faster RCNN	ResNet-101	33.7	33.4	0.89%

All results COCO mAP on COCO 2017 validation, higher is better

Pascal VOC

Model	Backbone	FP32	INT8	Rel Err
SSD-300	VGG-16	77.7	77.6	0.13%
SSD-512	VGG-16	79.9	79.9	0.0%

All results VOC mAP on VOC 07 test, higher is better

IMAGE SEGMENTATION

Model	Backbone	FP32	INT8	Relative Error
NV-ADLR Mask RCNN*	ResNet-101	39.6	39.0	1.52%

	mAP	APs	APm	APl
FP32	39.6	11.4	34.8	65.9
INT8	39.0	9.88	34.6	65.4

** 4th place in https://www.cityscapes-dataset.com/benchmarks/#instance-level-scene-labeling-task*
All results Cityscapes mask mAP on val_fine dataset, higher is better

LANGUAGE TRANSLATION

- GNMT: LSTM, 8 layer encoder, 8 layer decoder (https://github.com/tensorflow/nmt)
  - BLEU De→En, newstest2015
    - FP32: 29.89
    - Int8: 29.97

LANGUAGE MODEL

- BERT (Deep Bidirectional Transformers) large uncased in Pytorch
- Fine tuned for
  - Classification: MRPC of GLUE dataset
  - Question answering: SQuAD 1.1
- Accuracy measured in Pytorch. Max calibration

LANGUAGE MODEL

Bert large uncased	FP32	Int8	Rel Err %
MRPC	0.855	0.823	3.74%
SQuAD 1.1 (F1)	91.01	85.16	6.43%

Out of the box, BERT loss accuracy significantly

LANGUAGE MODEL

Bert large uncased	FP32	Int8	Rel Err %
MRPC	0.855	0.823	3.74%
SQuAD 1.1 (F1)	91.01	85.16	6.43%

Out of the box, BERT loss accuracy significantly

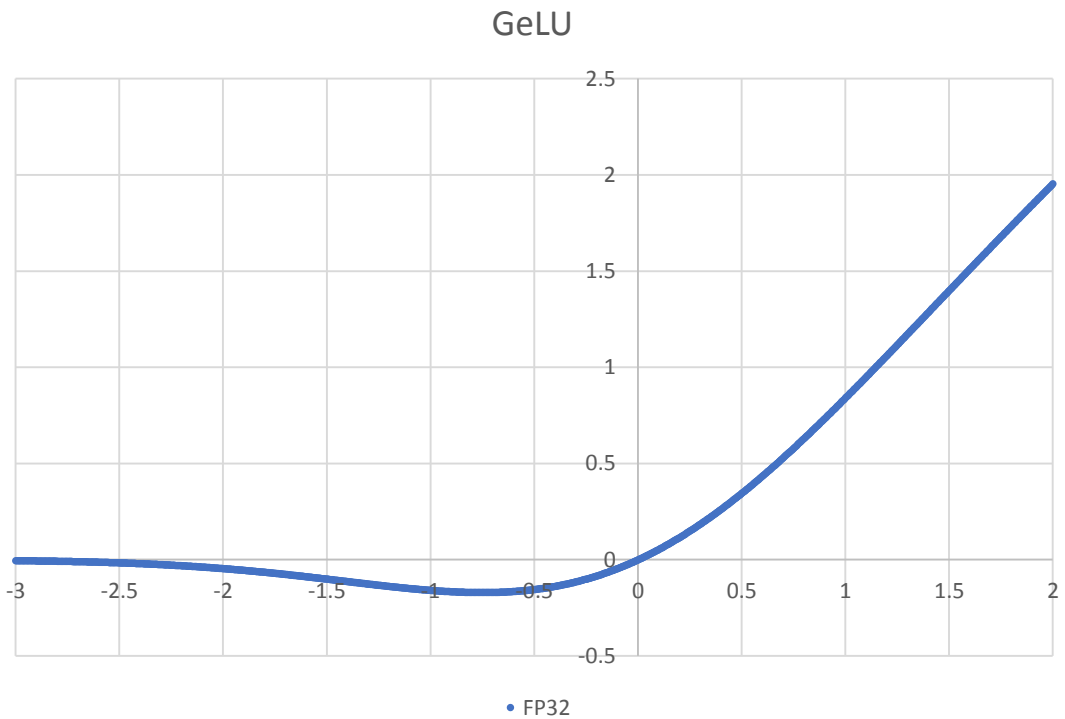
Bert large uncased	FP32	Int8 (GeLU10)	Rel Err %
MRPC	0.855	0.843	0.70%
SQuAD 1.1 (F1)	91.01	90.40	0.67%

With the right clipped GeLU, relative error is within 1%.

GeLU

BERT uses GeLU which produces asymmetric range.

Negative values generated by GeLU are between [-0.17, 0].



$$f(x) = \frac{x}{2}(1 + erf(\frac{x}{\sqrt{2}}))$$

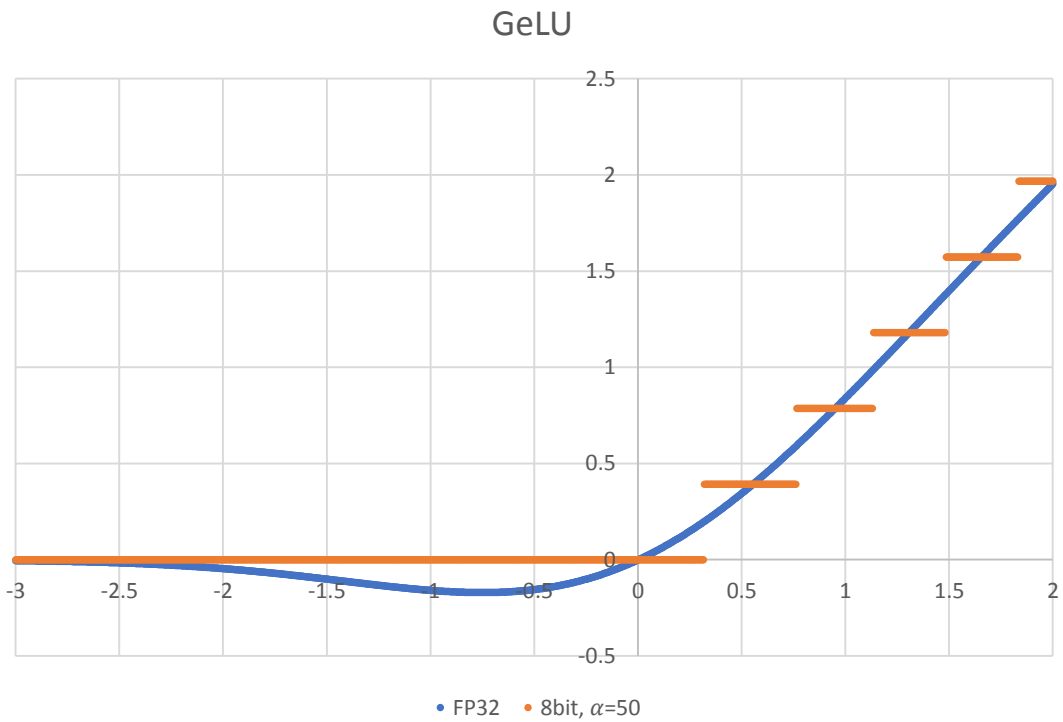
GeLU

BERT uses GeLU which produces asymmetric range.

Negative values generated by GeLU are between [-0.17, 0].

If α >= (0.5/0.17) ~= 43.18, all the negative values will be quantized to 0.

Maximum absolute values encountered are >50



$$f(x) = \frac{x}{2}(1 + erf(\frac{x}{\sqrt{2}}))$$

GeLU

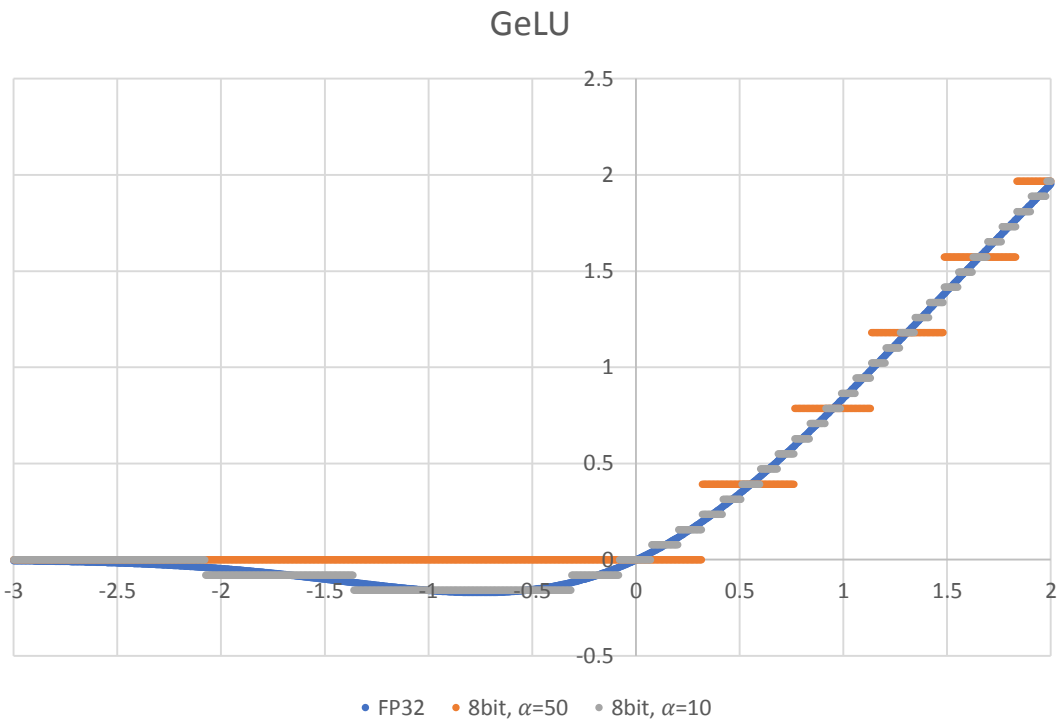
BERT uses GeLU which produces asymmetric range.

Negative values generated by GeLU are between [-0.17, 0].

If alpha >= (0.5/0.17) ~= 43.18, all the negative values will be quantized to 0.

Maximum absolute values encountered are >50

Clip GeLU output to 10 will have 2 negative quantized values



$$f(x) = \frac{x}{2}(1 + erf(\frac{x}{\sqrt{2}}))$$

SUMMARY OF POST TRAINING QUANTIZATION

- Manually add clip at the right place can help quantization
- Mobilenetv1(with relu6) and some other networks still losses >1% relative

REASONS QUANTIZATION MAY LOSE ACCURACY

- Outlier in the tensor
  - Example: BERT, Inception V4
  - Solution: Clip. Tighten the range, use bits more efficiently
- Not enough precision in quantized representation
  - Example: Int8 for MobileNet V1
  - Example: Int4 for Resnet50
  - Solution: Train/fine tune for quantization

TRAIN FOR QUANTIZATION

- Why do we need to train (fine tune) for quantization?
  - Some networks lose >1% accuracy with best post training quantization when quantizing to 8bit
  - It is much harder to post training quantize with fewer than 8 bits

TRAIN FOR QUANTIZATION TECHNIQUES

- Making range more quantization friendly, get rid of outliers
  - Clip
  - PACT (Parameterized Clipping Activation)[1].
- Adding quantization to training
  - Challenge: Quantization is a nondifferentiable function
  - Approximate derivative: STE (Straight-Through Estimator)[2]
  - Other methods also exist

1. https://arxiv.org/abs/1805.06085
2. https://arxiv.org/abs/1308.3432

CLIP

- Differentiability of clip is similar as ReLU, can back propagate
- Choosing clip threshold
  - Arbitrarily chosen fixed number, e.g. ReLU6
  - Arbitrarily chosen percentile
- Example:
  - BERT, SQuAD 1.1 (F1), clip GeLU output to 10

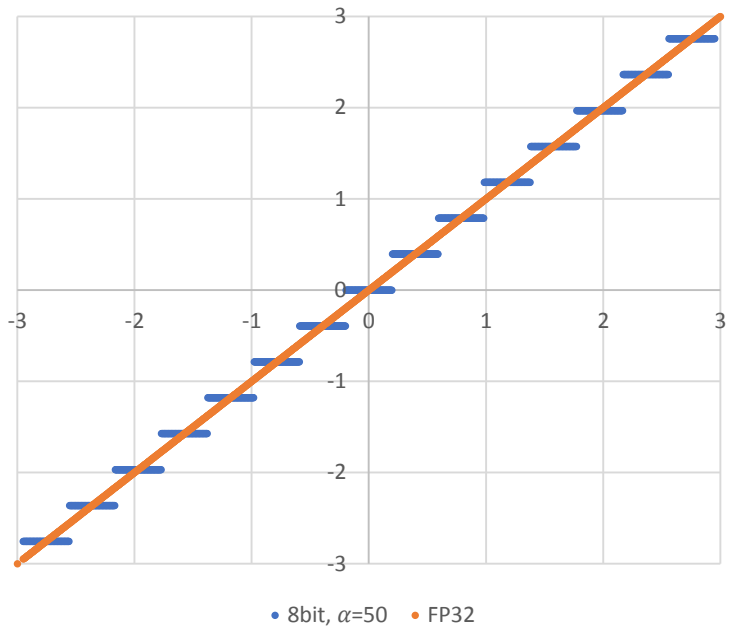
Bert large uncased	FP32	Int8 (GeLU10)	Rel Err %
Post training GeLU10	91.01	90.40	0.67%
Finetue with GeLU10	90.95	90.71	0.33%

PACT

- Learning clip threshold
- Requires its own hyper-parameter choices:
  - learning rate and decay of clipping threshold instead of arbitrarily picking threshold
- Originally designed for activation with quantization
  - Can be used independently to quantization
  - Can be applied to weight as well
- Results will come later in the 4bit section

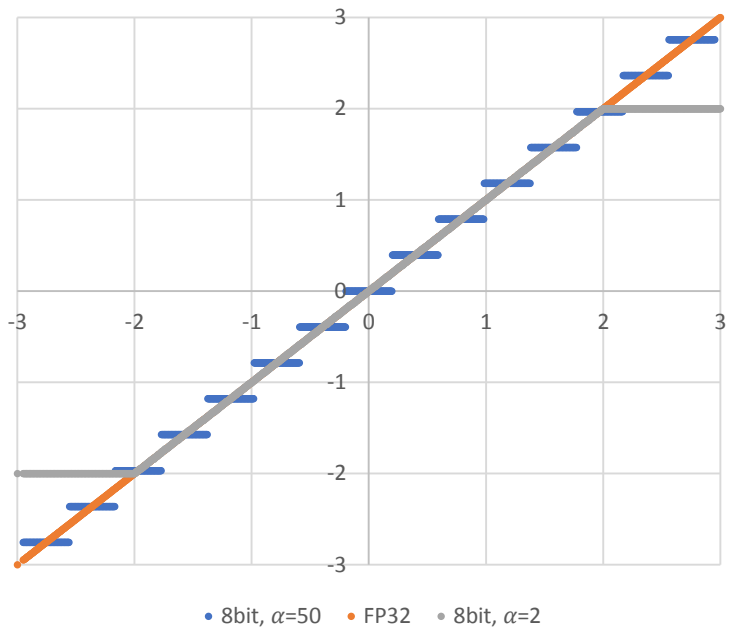
STE (STRAIGHT-THROUGH ESTIMATOR)

- Quantization is a step function which is not differentiable
  - $\frac{dy_q}{dy} = 0$
- Commonly used approximation is STE
  - Back propagate with $\frac{dy_q}{dy} = 1$



STE (STRAIGHT-THROUGH ESTIMATOR)

- Quantization is a step function which is not differentiable
  - $\frac{dy_q}{dy} = 0$
- Commonly used approximation is STE
  - Back propagate with $\frac{dy_q}{dy} = 1$
  - Works better when step size is small



MOBILENET WITH STE

- Fine tune MobileNet V1 with STE

MobileNet V1	FP32	Int8 (max)	Rel Err %
Post training quantization	70.90	68.9	2.82%
Fine tune with STE	70.90	70.60	0.42%

Pytorch version which gets slightly different fp32 accuracy compare to Tensorflow version

4-BIT QUANTIZATION

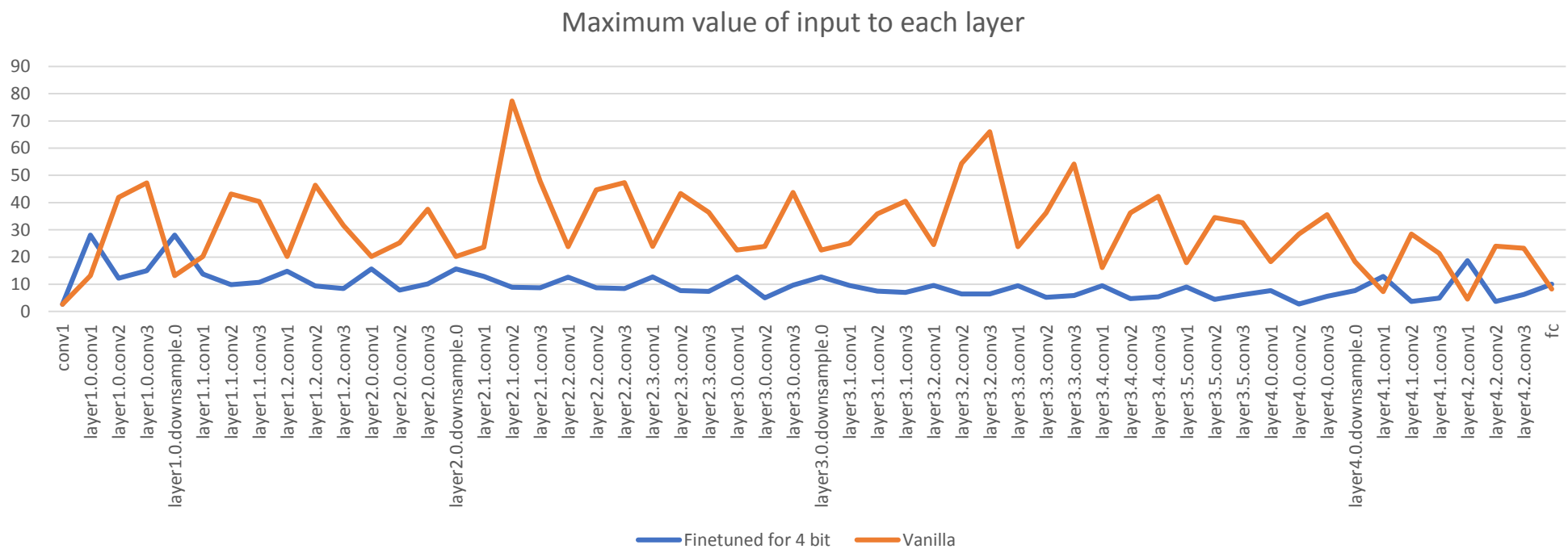
- Post-training 4-bit quantization loses a lot of accuracy
- Solution:
  - Use mixed precision quantization. e.g. 8bit + 4bit
  - Fine tune for quantization
    - Clip activation
    - Clip weight
    - STE with small learning rate works OK for CNN

4BIT RESNET50 V1.5

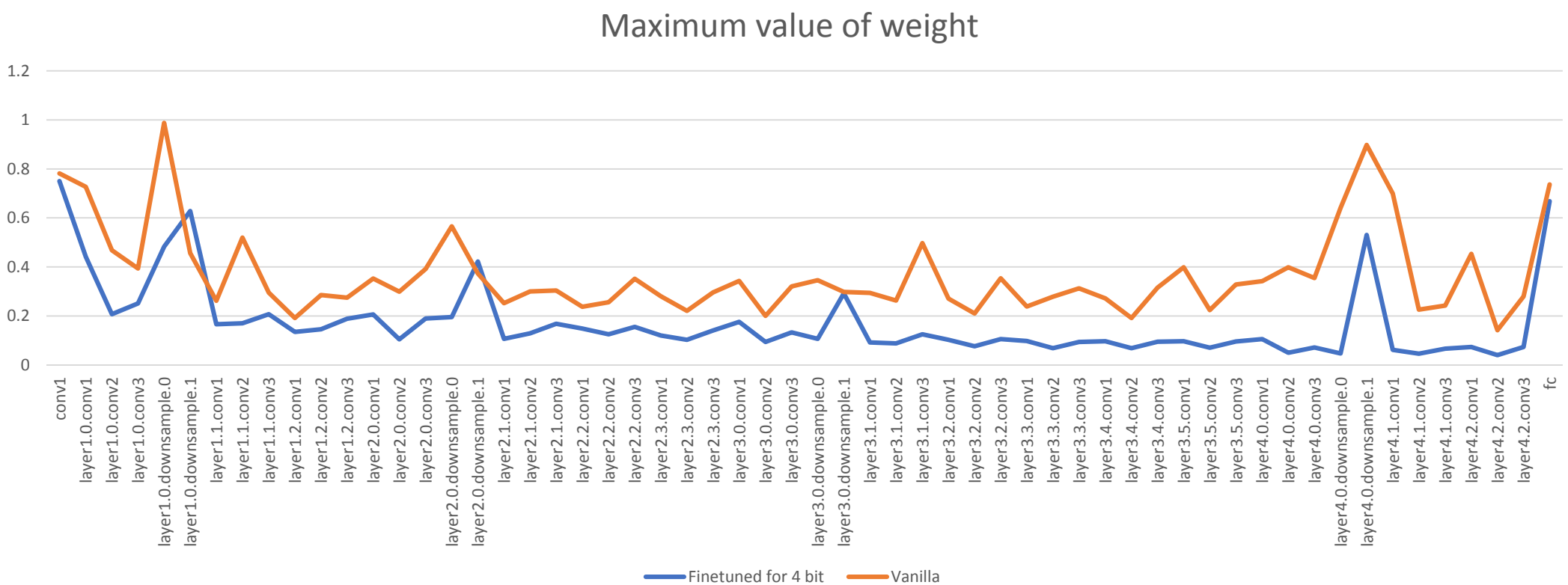
- 76.3% top-1 with mixed precision quantization
  - 8bits activation and weights for the first and downsample convolution and the last fully connected layer
  - Unsigned 4bit for activation generated by ReLU, 8bit for the rest.

Resnet50 V1.5	FP32	Int4 + Int8	Rel Err %
Post training quantization	0.761	0.576	24.31%
STE finetune, no clip	0.762	0.693	9.06%
STE finetune, clip	0.764	0.763	0.13%
STE finetune, PACT	0.763	0.762	0.13%

ACTIVATION CLIP IN 4BIT RESNET50



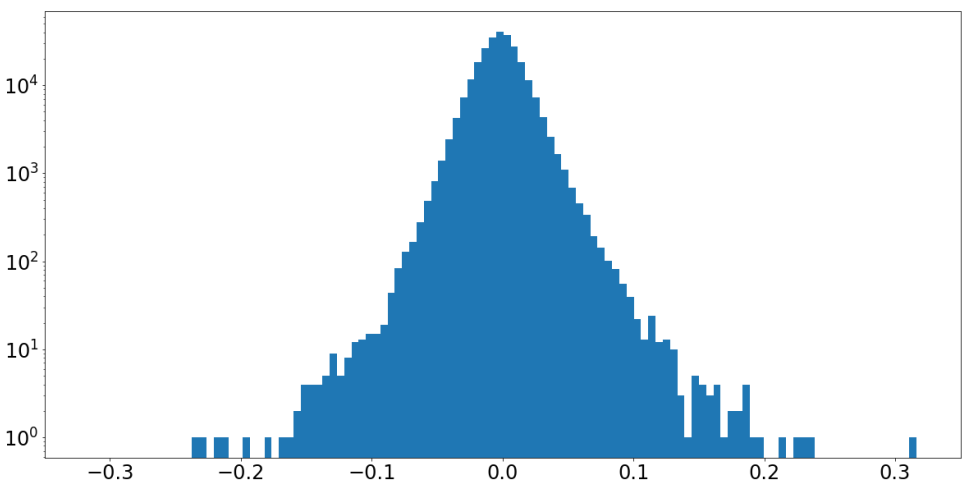
WEIGHT CLIP IN 4BIT RESNET50



WEIGHT CLIP IN 4BIT RESNET50

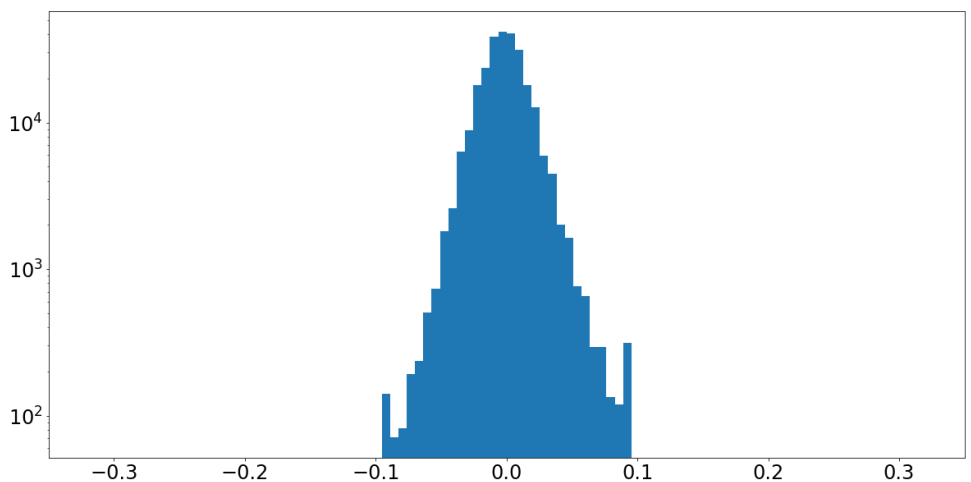
example

weight of layer3.4.conv3 in resnet50(torchvision)



weight of layer3.4.conv3 in resnet50(torchvision), fine tuned for 4bit

Much tight range, no outliers



SUMMARY

- Int8 quantized inference can be 4~8x faster than FP32
- Use scale only quantization, don’t use shift
- Per tensor scale for activation, fine grained scale for weight
- Most networks can be quantized to 8 bit by post training quantization
- Must train for 4bit quantization


NVIDIA